EE412 Foundation of Big Data Analytics, Fall 2021

HW2

Name: 함태욱

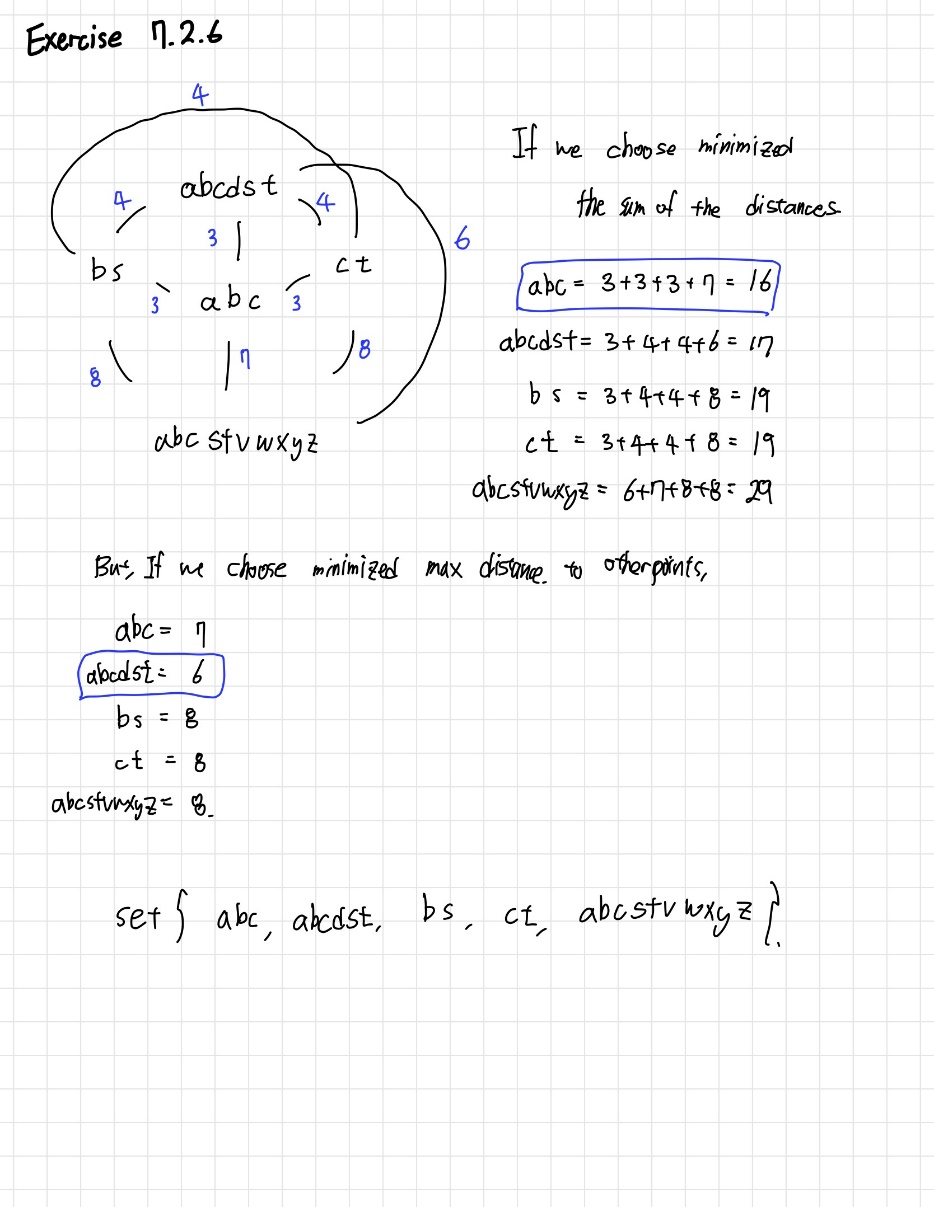
Student ID: 20180716

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

(a)



(b)

(1, 15840.015935162375)

(2, 8712.758000610926)

(3, 4225.773466007261)

(4, 2018.267391109598)

(5, 1526.2137994123182)

(6, 1420.7214130743844)

(8, 1003.3225742993732)

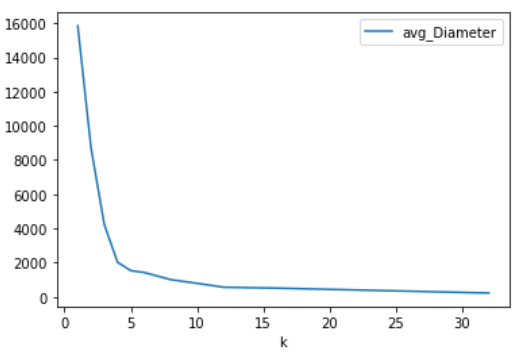
(10, 784.7882449282713)

(12, 557.6125972712002)

(16, 504.6115133174929)

(32, 216.9042145953651)

[Graph]



-The k value with an explanation why it is good for this data.

v = 10 일때, v(=10) ~ 2v(=20)간의 기울기 변화가 적고, v/2 < k < v 를 만족하는k 값은 6,7,8,9 중의 하나이다. 이때 binary search 를 사용한 결과 나는 k=8 이 ‘best value’ 라고 생각하여 선택하였다.

## Answer to Problem 2

(1)

**[Code Attachment]**

import sys

import numpy as np

M1 = np.array([[1,1,1],[1,2,3],[1,3,6]])

x0 = np.array([1,1,1]).T

# (a)

x = x0

prev\_x = x

while True:

    x = np.matmul(M1,x)

    x = x/np.linalg.norm(x)

    if np.linalg.norm(x-prev\_x)<0.000001:

            break

    prev\_x = x

# (b)

eig\_vec1 = x

eig\_val1=np.matmul(np.matmul(eig\_vec1.T,M1),eig\_vec1)

# (c)

M2 = M1 - np.dot(eig\_val1 , np.matmul(eig\_vec1.reshape(3,1),eig\_vec1.reshape(1,3)))

# (d)

x = eig\_vec1

prev\_x =x

while True:

    x = np.matmul(M2,x)

    x = x/np.linalg.norm(x)

    if np.linalg.norm(x-prev\_x)<0.000001:

            break

    prev\_x = x

eig\_vec2 = x

eig\_val2=np.matmul(np.matmul(eig\_vec2.T,M2),eig\_vec2)

M3 = M2 - np.dot(eig\_val2 , np.matmul(eig\_vec2.reshape(3,1),eig\_vec2.reshape(1,3)))

# (e)

x = eig\_vec2

prev\_x = x

while True:

    x = np.matmul(M3,x)

    x = x/np.linalg.norm(x)

    if np.linalg.norm(x-prev\_x)<0.000001:

            break

    prev\_x = x

eig\_vec3 = x

eig\_val3=np.matmul(np.matmul(eig\_vec3.T,M2),eig\_vec3)

Eigenvalues = [7.872983346207407, 1.0000000000000713, 0.12701665379258312]

Eigenvectors =

[0.19382269 0.4722473 0.85989258]

[-0.81649653 -0.40824816 0.40824854]

[ 0.54384382 -0.78122714 0.30646053]

(2)

**[Code Attachment]**

import numpy as np

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

row = 5

col = 3

rank = 2

# (a)

MTM = np.matmul(M.T,M)

MMT = np.matmul(M,M.T)

# (b)

val\_V,vec\_V = np.linalg.eig(MTM)

val\_U,vec\_U = np.linalg.eig(MMT)

# (c)

sigma = np.array([])

V = np.array([]) #col x rank

V = np.append(V,vec\_V.T[0])

V = np.append(V,vec\_V.T[1])

V = V.reshape(rank,col).T

U = np.array([]) # row x rank

U = np.append(U,vec\_U.T[0])

U = np.append(U,vec\_U.T[2])

U = U.reshape(rank,row).T

for val in val\_V:

    if val > 0.01:

        sigma = np.append(sigma,val)

sigma = np.sqrt(sigma)

sigma\_diag = np.diag(sigma)

energy = np.linalg.norm(sigma)

two\_dim\_M = np.matmul(np.matmul(U,sigma\_diag),-V.T)

# (d)

index = np.argmin(sigma)

sigma\_diag[index]=0

U.T[index]=0

V.T[index]=0

one\_dim\_M = np.matmul(np.matmul(U,sigma\_diag),-V.T)

# (e)

E\_retained = sigma[0]\*\*2/(np.sum(sigma\*\*2))

(a)

MTM =

[[36 37 38]

[37 49 61]

[38 61 84]]

MMT =

[[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)

Eigenpairs of MTM

[1.53566996e+02, 1.54330035e+01, 2.99519331e-15]

(third eigenvalue is almost zero)

[[-0.40928285 -0.81597848 0.40824829]

[-0.56345932 -0.12588456 -0.81649658]

[-0.7176358 0.56420935 0.40824829]]

Eigenpairs of MMT

[ 1.53566996e+02, -2.16919039e-15, 1.54330035e+01, -2.69964138e-15, -1.63115212e-16]

(second, fourth, fifth eigenvalue is almost zero)

[[ 0.29769568 0.94131607 -0.15906393 -0.57735012 -0.21094872]

[ 0.57050856 -0.17481584 0.0332003 -0.22666834 0.06716429]

[ 0.52074297 -0.04034212 0.73585663 0.10591706 -0.13512315]

[ 0.32257847 -0.18826321 -0.5103921 -0.27280206 -0.68074095]

[ 0.45898491 -0.21515796 -0.41425998 0.72776982 0.68507159]]

(c)

U =

[[ 0.29769568 -0.15906393]

[ 0.57050856 0.0332003 ]

[ 0.52074297 0.73585663]

[ 0.32257847 -0.5103921 ]

[ 0.45898491 -0.41425998]]

Sigma =

[[12.39221516 0. ]

[ 0. 3.92848616]]

V.T =

[[0.40928285 0.56345932 0.7176358 ]

[0.81597848 0.12588456 -0.56420935]]

(Here, I multiply -1 to V.T from my code output)

M = U\*Sigma\*(V.T)

= [[1.00000000e+00 2.00000000e+00 3.00000000e+00]

[3.00000000e+00 4.00000000e+00 5.00000000e+00]

[5.00000000e+00 4.00000000e+00 3.00000000e+00]

[6.66133815e-16 2.00000000e+00 4.00000000e+00]

[1.00000000e+00 3.00000000e+00 5.00000000e+00]]

I multiply -1 to the original V.T from my code because eigenvector multiplied by a constant is also an eigenvector.

And multiplying -1 to the V makes the right output of original M

(d)

One – dimensional approximation to the matrix M

[[1.509889 2.0786628 2.64743661]

[2.89357443 3.98358126 5.0735881 ]

[2.64116728 3.63609257 4.63101787]

[1.63609257 2.25240715 2.86872172]

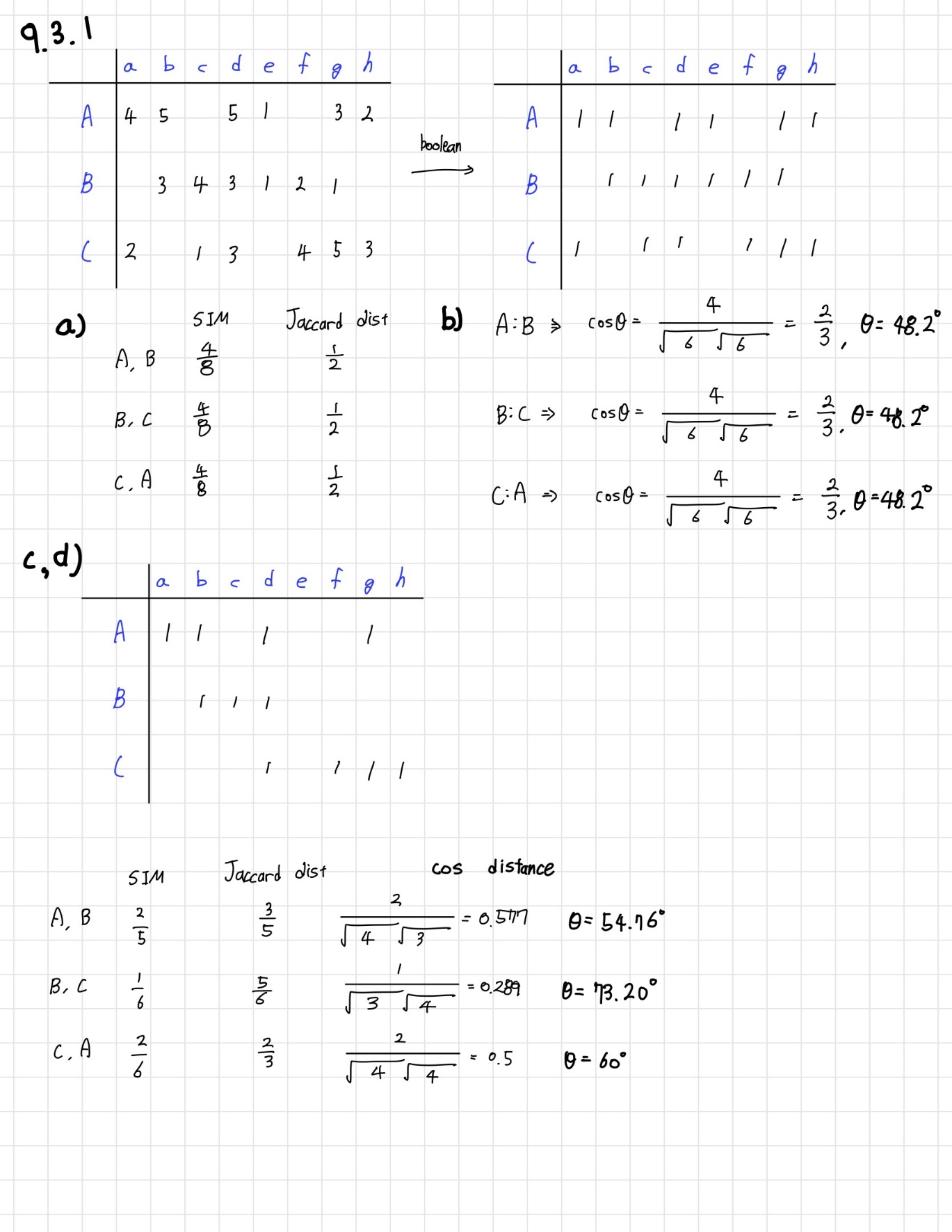
[2.32793529 3.20486638 4.08179747]]

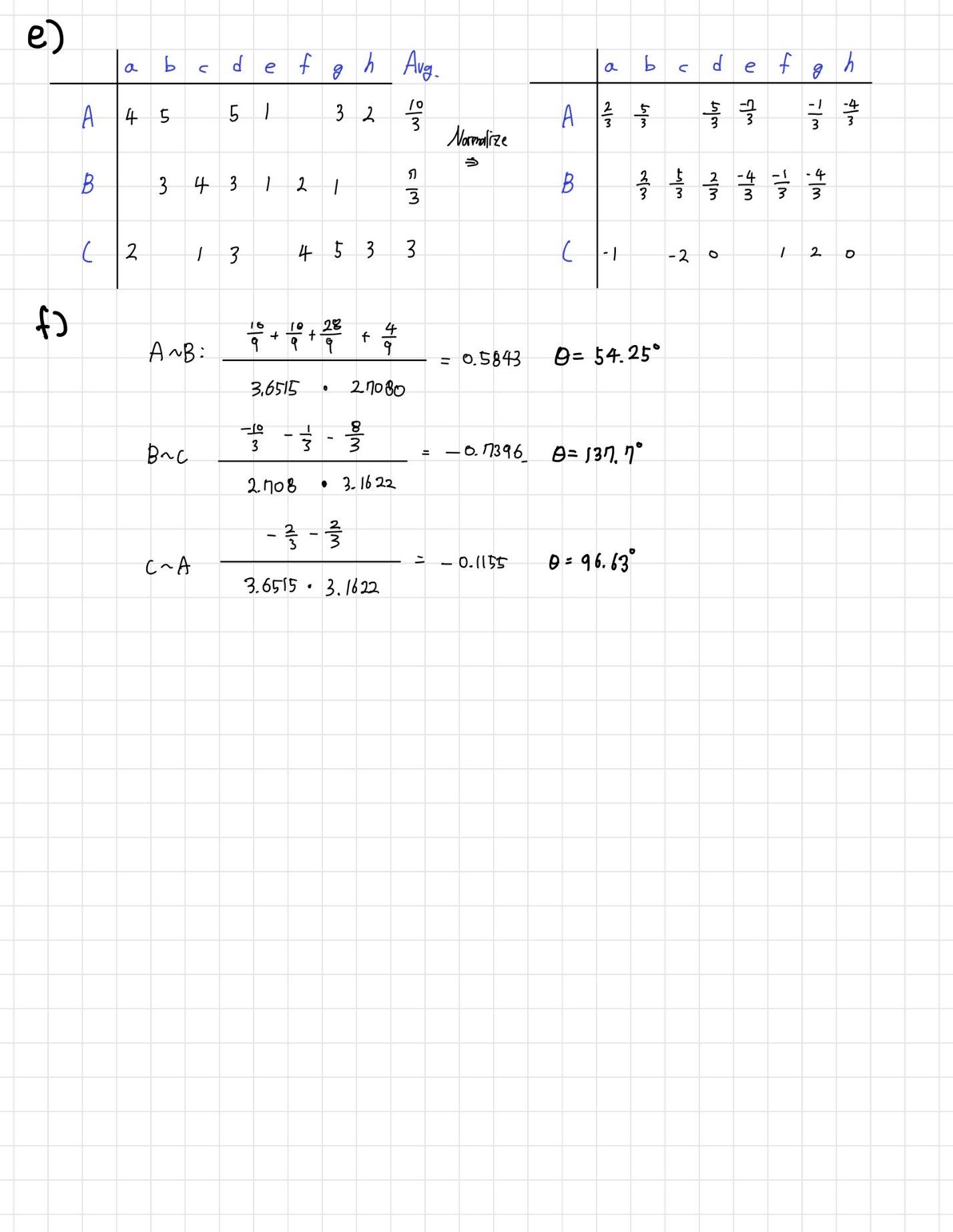
(e)

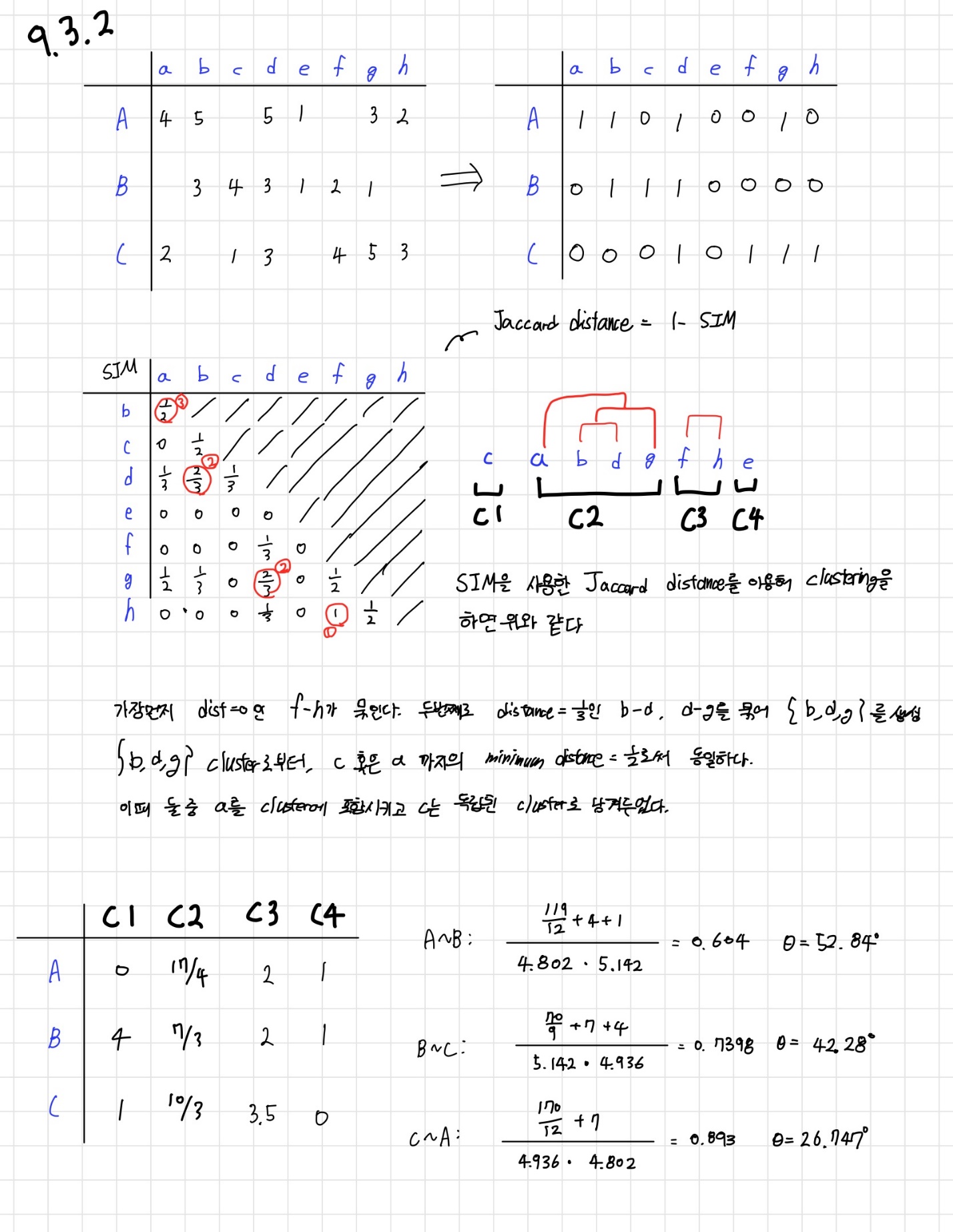
0.9086804524257934= about 90.87%

## Answer to Problem 3

(a)







(b)

175 5.0

261 5.0

440 5.0

480 5.0

527 5.0

5 5.0

318 5.0

364 5.0

785 5.0

1 4.5

(c)

임의의 사용자에 의한 평가되지 않은 영화에 대한 별점 예측을 하기위해서 UV decomposition을 사용.

Utility Matrix U(n x m) 에 대해 n by 2 matrix U, 2 by m matrix V 를 생성(m = rank로 설정 시 계산시간이 너무 오래걸려 확인이 힘들었음), 그리고 각 벡터값은 0부터 1사이의 임의의 값으로 채웠다. 직접 간단한 K-Fold method를 이용해 ratings를 4개의 set으로 나누어 진행. RMSE를 최소로 만들도록 U->V 순으로 최적화 진행, 이후 만들어진 P = UV matrix로부터 ratings\_test.txt의 user-movie ratings를 예측.