# EE412 Foundation of Big Data Analytics, Fall 2019
# HW2

Name: 오승준

Student ID: 20160381

Discussion Group (People with whom you discussed ideas used in your answers):
손채연(numpy methods, collaborative filtering algorithm, improve performance)

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

## Answer to Problem 1

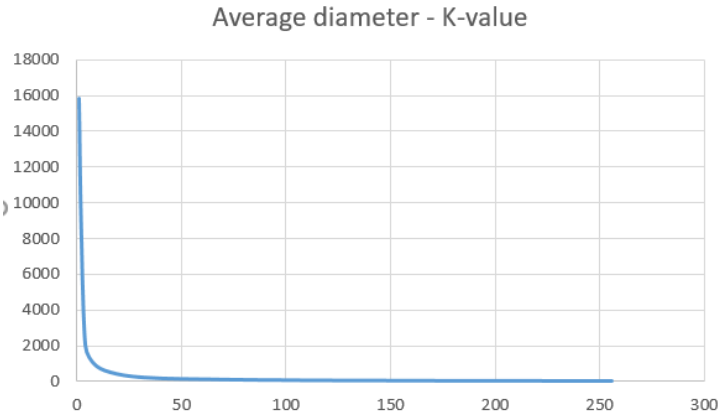
**(a) Exercise 7.2.6**

Answer: a, ab, abc, abcd, abcdefg

If we choose the clusteroid by minimizing the sum of the distances to the other points, we get ‘abc’ as clusteroid. However, if we choose the clusteroid by minimizing the maximum distance to the other points, we get ‘abcd’ as clusteroid.

**(b) Implement the k-Means algorithm using Spark**

| K value | 1 | 2 | 4 | 8 | |
|---|---|---|---|---|---|
| Average Diameter | 15840.015935 | 8712.758001 | 2018.267391 | 1003.322574 | |
| K value | 16 | 32 | 64 | 128 | 256 |
| Average Diameter | 504.611513 | 216.904215 | 110.928037 | 48.644531 | 19.191004 |



The best range of k value is between 16 and 32, because from 32 to 64 average diameter decrease little. Therefore, best k should be between 16 and 32.

If k = 24, average diameter is 298.600990. From 24 to 32, average diameter decrease more slowly, so k should be between 16 to 24.

If k = 20, average diameter is 392.245073. The change between 16 and 20 and between 20 and 24 are similar.

Therefore, we can choose best k value as 20.

# Answer to Problem 2

- **Exercise 11.1.7**

---- CODE ----

```
import numpy as np

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

# Finding eigenpairs by power iteration.
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
```

---- ANSWER ----

(a) Approximate value of the principal eigenvector

[[ 0.19382449]
 [ 0.4722482 ]
 [ 0.85989168]]

(b) The principal eigenvalue of for the matrix

eigenvalue: 7.872983

(c) Construct a new matrix

[[ 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**

---- CODE ----

```
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])
for i in range(rank):
    index = values.index(max(values))
    values[index] = 0
    U[:,i] = MMT_pair[1][:,index]
if U[0,0] < 0:
    U = -U

sigma = np.sqrt(np.diag(eig_values))

print "U,V,S of M"
print U
print V
print sigma

# (d) Set smaller singular value to 0
U_1 = U[:,:-1]
V_1 = V[:,:-1]
sigma_1 = sigma[:-1,:-1]

print "Approximated U,V,S"
print U_1
print V_1
print sigma_1

# (e) Compare energy of the original and approximation
print "Original energy: %f" % (np.sum(np.square(sigma)))
print "Approximated energy: %f" % (np.sum(np.square(sigma_1)))
```

---- ANSWER ----

(a) M'M and MM'

| | | | |
|---|---|---|---|
| M'M | [[36 37 38]<br>[37 49 61]<br>[38 61 84]] | MM' | [[14 26 22 16 22]<br>[26 50 46 28 40]<br>[22 46 50 20 32]<br>[16 28 20 20 26]<br>[22 40 32 26 35]] |

(b) Eigenpairs of M'M and MM'

- 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%

- **Exercise 11.4.2**

---- CODE ----

```
import numpy as np

# Matrix of users and items
M = np.array([[1,1,1,0,0],[3,3,3,0,0],[4,4,4,0,0],[5,5,5,0,0],
              [0,0,0,4,4],[0,0,0,5,5],[0,0,0,2,2]])
M_norm = float(np.sum(np.square(M)))

# select row and columns
# (a) [1,2], [0,1] (b) [3,4] [1,2] (c) [0,6] [0,4]
row_select = [1,2]
col_select = [0,1]
r = np.linalg.matrix_rank(M)

# Initialize C,U,R
C = np.zeros((M.shape[0], r))
U = np.zeros((r,r))
R = np.zeros((r, M.shape[1]))

# Compute matrix C
for i in range(r):
    q = float(np.sum(np.square(M[:,col_select[i]])))
    weight = (r * (q/M_norm)) ** 0.5
    C[:,i] = M[:,col_select[i]] / weight
print C

# Compute matrix R
for i in range(r):
    q = float(np.sum(np.square(M[row_select[i],:])))
    weight = (r * (q/M_norm)) ** 0.5
    R[i,:] = M[row_select[i],:] / weight
print R

# Compute matrix U
W = np.zeros((r,r))
for i in range(r):
    for j in range(r):
        W[i,j] = M[row_select[i], col_select[j]]
W_rank = np.linalg.matrix_rank(W)
WTW_pair = np.linalg.eig(np.dot(np.transpose(W), W))
WWT_pair = np.linalg.eig(np.dot(W, np.transpose(W)))
X = np.zeros((r,W_rank))
Y = np.zeros((r,W_rank))
eig_values = []
```

```
values = list(WTW_pair[0])
for i in range(W_rank):
    index = values.index(max(values))
    eig_values.append(values[index])
    values[index] = 0
    Y[:,i] = WTW_pair[1][:,index]
if Y[0,0] < 0:
    Y = -Y

values = list(WWT_pair[0])
for i in range(W_rank):
    index = values.index(max(values))
    values[index] = 0
    X[:,i] = WWT_pair[1][:,index]
if X[0,0] < 0:
    X = -X

sigma = np.sqrt(np.diag(eig_values))
inverse_sigma = np.linalg.pinv(sigma)

U = np.dot(np.dot(np.dot(Y,inverse_sigma), inverse_sigma), np.transpose(X))
print U
```

---- ANSWER ----

(a) The columns for The Matrix and Alien and the rows for Jim and John.

- C

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

- U

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

- R

```
[[6.36396103  6.36396103  6.36396103  0.  0.  ]
 [6.36396103  6.36396103  6.36396103  0.  0.  ]]
```

(b) The columns for Alien and Star Wars and the rows for Jack and Jill.

- C

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

- U

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

- R

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

(c) The columns for The Matrix and Titanic and the rows for Joe and Jane.

- C

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

- U

```
[[1.    0.  ]
[0.    0.25]]
```

- R

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

# 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.

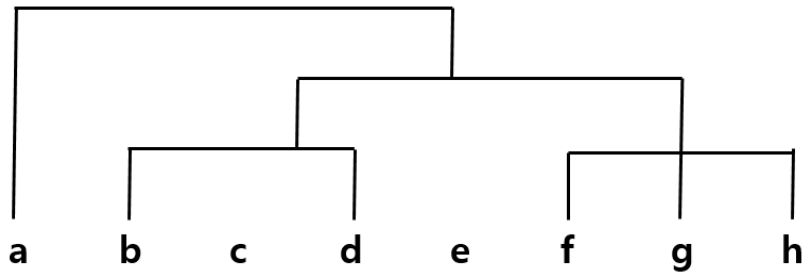
New matrix

| | 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)

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

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

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

- Exercise 9.4.3

a. Find best value of u11

Figure 9.16

$$\begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ 1.178 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1.617 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x & 1 \\ 1 & 1 \\ 1.178 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1.617 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1.617x+1 & x+1 & x+1 & x+1 & x+1 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.905 & 2.178 & 2.178 & 2.178 & 2.178 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.617 & 2 & 2 & 2 & 2 \end{bmatrix}$$

The sum of the squares of the errors

$$(5-(1.617x+1))^2+(2-(x+1))^2+(4-(x+1))^2+(4-(x+1))^2+(3-(x+1))^2$$

Its derivative

$$13.2294\text{x} - 30.936 = 0, when\ minimum$$

$$\therefore \text{x} = u_{11} = 2.338$$

b. Find best value of u52

$$\begin{bmatrix} 2.338 & 1 \\ 1 & 1 \\ 1.178 & 1 \\ 1 & 1 \\ 1 & x \end{bmatrix} \times \begin{bmatrix} 1.617 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 4.781 & 3.338 & 3.338 & 3.338 & 3.338 \\ 2.617 & 2 & 2 & 2 & 2 \\ 2.905 & 2.178 & 2.178 & 2.178 & 2.178 \\ 2.617 & 2 & 2 & 2 & 2 \\ x + 1.617 & x + 1 & x + 1 & x + 1 & x + 1 \end{bmatrix}$$

The sum of the squares of the errors

$$(4 - (x + 1.617))^2 + (4 - (x + 1))^2 + (5 - (x + 1))^2 + (4 - (x + 1))^2$$

Its derivative

$$8(\mathrm{x} - 3.09575) = 0, when\ minimum$$

$$\therefore \mathrm{x} = u_{52} = 3.096$$

c. Find best value of u22

$$\begin{bmatrix} 2.338 & 1 \\ 1 & 1 \\ 1.178 & 1 \\ 1 & 1 \\ 1 & 3.096 \end{bmatrix} \times \begin{bmatrix} 1.617 & 1 & 1 & 1 & 1 \\ 1 & x & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 4.781 & x + 2.338 & 3.338 & 3.338 & 3.338 \\ 2.617 & x + 1 & 2 & 2 & 2 \\ 2.905 & x + 1.178 & 2.178 & 2.178 & 2.178 \\ 2.617 & x + 1 & 2 & 2 & 2 \\ 4.713 & 3.096x + 1 & 4.096 & 4.096 & 4.096 \end{bmatrix}$$

The sum of the squares of the errors

$$(2 - (x + 2.338))^2 + (1 - (x + 1))^2 + (5 - (x + 1))^2 + (4 - (3.096x + 1))^2$$

Its derivative

$$25.1704\mathrm{x} - 25.9 = 0, when\ minimum$$

$$\therefore \mathrm{x} = v_{22} = 1.029$$

**(b) Implement collaborative filtering**

Output of the top-5 movies among movies 1 to 1000 for U with the highest predicted ratings using the *user-based* method.

The result of prediction user-based

| | |
|---|---|
| 261 | 5.969583 |
| 899 | 5.969583 |
| 440 | 5.561315 |
| 527 | 5.371794 |
| 7 | 5.265449 |

Output of the top-5 movies among movies 1 to 1000 for U with the highest predicted ratings using the *item-based* method.

The result of prediction item-based

| | |
|---|---|
| 1 | 5.000000 |
| 5 | 5.000000 |
| 364 | 5.000000 |
| 785 | 5.000000 |
| 16 | 4.500000 |

**(c) Movie Recommendation Challenge**

Command-line: `python hw2_3c.py path/to/ratings/txt path/to/ratings_test.txt`

---- TECHNIQUES ----

I use two methods.

1. Better normalization

   I use other normalization method. In HW2_3b, we normalized utility matrix by subtract from each rating the average rating of that user. However, in this time, I use standard normalization, which is also called Z-score. I compute average and standard deviation of rating of that user. I ignore blank ratings when computing average and standard deviation. Then, I normalize ratings by subtract average, then divide with standard deviation.

   $$Z = \frac{ratings - average}{standard\ deviation}$$

   After I predict ratings, I restore it. I multiply standard deviation and add average.

2. UV decomposition

   I use UV decomposition when predict ratings. There are two functions: optimize_u() and optimize_v(). optimize_u() is optimizing $u_{ij}$ by minimizing difference between normalized matrix and U*V, while increasing $u_{ij}$. Similarly, optimize_v() is optimiziong $v_{ij}$ by minimizing difference between normalized matrix and U*V. Here, I use error() function to get sum of square of differences. Also, I do not compute difference of whole matrix. I just compute where $u_{ij}$ or $v_{ij}$ affects. For example, when I optimize $u_{1,2}$, I compute difference between i-row of normalized matrix and i-row of U*V.

   (한국어로 간단히 말하자면, $u_{ij}$ 를 최적화한다고 했을 때, $u_{ij}$ 를 조금씩 증가시키면서 normalized matrix 와의 차이가 최소가 되는 지점을 찾았습니다.)

   First, I initialize U and V matrices with 0 because matrix is already normalized. Then, I do optimize all elements in U and V.

   When we do UV computation, we need permutation for random selection. For this, I make train_data which have all index pair of U and V matrices. I sort train_data randomly, then I train data according to its order.