EE412 Foundation of Big Data Analytics, Fall 2022
HW2

Name: 고건호

Student ID: 20160025

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

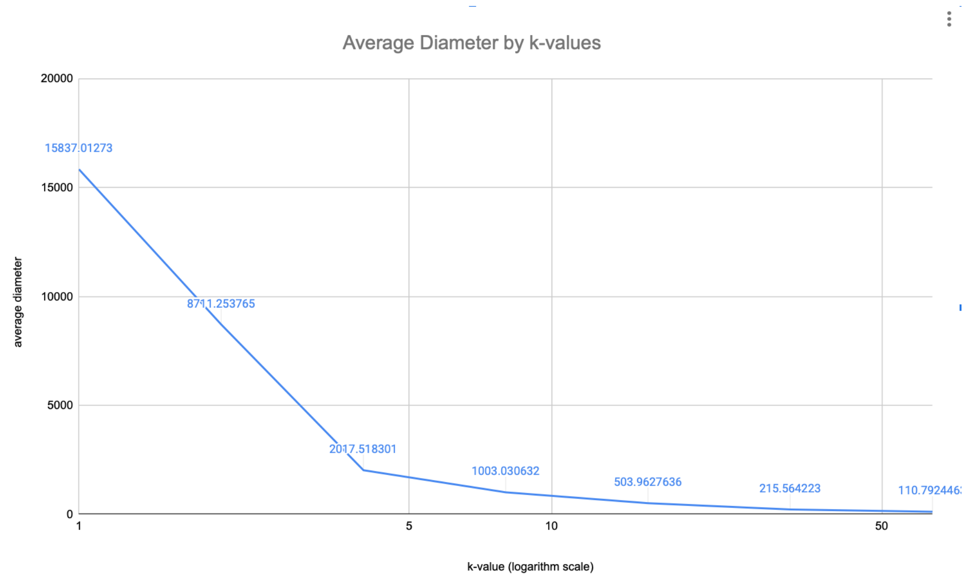
On-line or hardcopy documents used as part of your answers:

Answer to Problem 1

(a) [5 pts] Solve the following problem, which is based on the exercises in the Mining of Massive Datasets 2nd edition (MMDS) textbook.

(b) [20 pts] Implement the k-Means algorithm using Spark

- The number-of-clusters vs. average diameter plot.



- The k value with an explanation why it is good for this data.
  Please choose the best k value based on your thought. You will get full points if the reasoning is logical.

  Changing the x-axis to logarithmic scale results in a point where the graph is clearly bent. The point is where **k = 4**. And when k is larger than 4, as k is doubled, the average diameter is halved, which means total diameter is not changing that much.

Answer to Problem 2

Exercise 11.1.7

```
import numpy as np

THRESHOLD = 1e-5

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

eig_vec1 = np.array([1, 1, 1]).reshape((3, 1))
diff = 1
while diff > THRESHOLD:
   new_eig_vec = A @ eig_vec1
   new_eig_vec = new_eig_vec / np.linalg.norm(new_eig_vec)
   diff = np.linalg.norm(eig_vec1 - new_eig_vec)
   eig_vec1 = new_eig_vec

print("(a) first eigen vector")
print(eig_vec1, end="\n\n")

eig_val1 = (eig_vec1.T @ A @ eig_vec1)[0,0]

print("(b) first eigen value")
print(eig_val1, end='\n\n')

A2 = A - eig_val1 * (eig_vec1 @ eig_vec1.T)

print("(c) new matrix")
print(A2, end='\n\n')

def get_eig_pair(M):
   eig_vec = np.array([1, 1, 1]).reshape((3, 1))
   diff = 1
   while diff > THRESHOLD:
       new_eig_vec = M @ eig_vec
       new_eig_vec = new_eig_vec / np.linalg.norm(new_eig_vec)
       diff = np.linalg.norm(eig_vec - new_eig_vec)
       eig_vec = new_eig_vec
   eig_val = eig_vec.T @ M @ eig_vec
   new_M = M - eig_val * (eig_vec @ eig_vec.T)
   return eig_vec, eig_val[0,0], new_M
```

```python
eig_vec2, eig_val2, A3 = get_eig_pair(A2)

print("(d) second eigen pair")
print(eig_vec2)
print(eig_val2, end='\n\n')

eig_vec3, eig_val3, A4 = get_eig_pair(A3)

print("(e) third eigen pair")
print(eig_vec3)
print(eig_val3, end='\n\n')

V = np.concatenate([eig_vec1, eig_vec2, eig_vec3], axis=1)
U = np.diag([eig_val1, eig_val2, eig_val3])

print("Verification: reconstructed matrix")
print(V @ U @ V.T)
```

(a) first eigen vector

```
[[0.19382289]
 [0.4722474 ]
 [0.85989248]]
```

(b) first eigen value

7.872983346206856

(c) New matrix

```
[[ 0.70423318  0.27936729 -0.31216529]
 [ 0.27936729  0.24418608 -0.19707675]
 [-0.31216529 -0.19707675  0.1785974 ]]
```

(d) **second eigen pair**

```
[[ 0.81649634]
 [ 0.40824695]
 [-0.40825011]]
1.0000000000042972
```

(e) third eigen pair

```
[[ 0.54384155]
 [-0.78122828]
 [ 0.30646167]]
0.1270166537934505
```

Exercise 11.3.1

```
import numpy as np

RANK = 2

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

S1 = M @ M.T
S2 = M.T @ M

print("(a) M@M.T and M.T@M")
print("- M@M.T", S1, sep='\n')
print("- M.T@M", S2, sep='\n', end='\n\n')

w1, v1 = np.linalg.eig(S1)
w2, v2 = np.linalg.eig(S2)

print("(b) eigenpairs for matrices of part (a)")
print("- w1", w1, "- v1", v1, "- w2", w2, "- v2", v2, sep='\n', end='\n\n')

U = v1[:, np.argsort(w1)][:, -RANK:]
V = v2[:, np.argsort(w2)][:, -RANK:]
S = np.sqrt(np.diag(sorted(w1)[-RANK:]))

print("(c) SVD for the matrix M")
print("- U", U, "- V", V, "- S", S, sep="\n", end='\n\n')

sv2 = S[0,0]
S[0,0] = 0
approximate = U @ S @ V.T

print("(d) approximation to the matrix M")
print(approximate, end='\n\n')

sv1 = S[-1,-1]
retained_energy = sv1**2 / (sv1**2 + sv2**2)
```

```python
print("(e) retained energy")
print(retained_energy)
```

(a) M@M.T and M.T@M

- M@M.T

```
[[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.T@M

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

(b) eigenpairs for matrices of part (a)

- w1

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

- v1

```
[[ 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]]
```

- w2

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

- v2

```
[[-0.40928285 -0.81597848  0.40824829]
 [-0.56345932 -0.12588456 -0.81649658]
 [-0.7176358   0.56420935  0.40824829]]
```

(c) SVD for the matrix M

- U

```
[[-0.15906393  0.29769568]
 [ 0.0332003   0.57050856]
 [ 0.73585663  0.52074297]
 [-0.5103921   0.32257847]
 [-0.41425998  0.45898491]]
```

- V

```
[[-0.81597848 -0.40928285]
```

[-0.12588456 -0.56345932]
[0.56420935 -0.7176358]]

- S

[[3.92848616 0.]
[0. 12.39221516]]

(d) 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) **retained energy**

0.9086804524257934

- **The reason approximation of M is negative**
  Since we collect vectors U and V through eigen pairs from $M^T M$ and MM^T, it is impossible to distinguish the sign of matrix M. Which means, we will get the same SVD decomposition result from matrix M and -M.

Answer to Problem 3

(a) [15 pts] Solve the following problems, which are based on the exercises in the MMDS textbook.

Exercise 9.3.1

(a) Jaccard distance

J(A, B) = ½ = 0.5
J(A, C) = ½ = 0.5
J(B, C) = ½ = 0.5

(b) Cosine distance

$$C(A, B) = \frac{5\times3 + 5\times3 + 1\times1 + 3\times1}{\sqrt{4^2 + 5^2 + 5^2 + 1^2 + 3^2 + 2^2} \times \sqrt{3^2 + 4^2 + 3^2 + 1^2 + 2^2 + 1^2}} = 0.601$$

$$C(A, C) = \frac{4\times2 + 5\times3 + 3\times5 + 2\times3}{\sqrt{4^2 + 5^2 + 5^2 + 1^2 + 3^2 + 2^2} \times \sqrt{2^2 + 1^2 + 3^2 + 4^2 + 5^2 + 3^2}} = 0.615$$

$$C(B, C) = \frac{4\times1 + 3\times3 + 2\times4 + 1\times5}{\sqrt{3^2 + 4^2 + 3^2 + 1^2 + 2^2 + 1^2} \times \sqrt{2^2 + 1^2 + 3^2 + 4^2 + 5^2 + 3^2}} = 0.514$$

(c) Rounded Jaccard distance

	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

J(A, B) = ⅗ = 0.6
J(A, C) = 4/6 = 0.66
J(B, C) = ⅚ = 0.833

(d) Rounded Cosine distance

$$C(A, B) = \frac{2}{\sqrt{4} \times \sqrt{3}} = 0.577$$

$$C(A, C) = \frac{2}{\sqrt{4} \times \sqrt{4}} = 0.5$$

$$C(B, C) = \frac{1}{\sqrt{3} \times \sqrt{4}} = 0.289$$

(e) Normalized Matrix

	a	b	c	d	e	f	g	h
A	0.67	1.67		1.67	-2.33		-0.33	-1.33
B		0.67	1.67	0.67	-1.33	-0.33	-1.33	
C	-1		-2	0		1	2	0

(f) Normalized Cosine distance

$$C(A, B) = \frac{1.67\times0.67 + 1.67\times0.67 + 2.33\times1.33 + 0.33\times1.33}{\sqrt{0.67^2 + 1.67^2 + 1.67^2 + 2.33^2 + 0.33^2 + 1.33^2} \times \sqrt{0.67^2 + 1.67^2 + 0.67^2 + 1.33^2 + 0.33^2 + 1.33^2}} = 0.584$$

$$C(A, C) = \frac{-0.67\times1 - 0.33\times2}{\sqrt{0.67^2 + 1.67^2 + 1.67^2 + 2.33^2 + 0.33^2 + 1.33^2}\times\sqrt{1^2+2^2+1^2+2^2}} = -0.115$$

$$C(B, C) = \frac{-1.67\times2 - 0.33\times1 - 1.33\times2}{\sqrt{0.67^2 + 1.67^2 + 0.67^2 + 1.33^2 + 0.33^2 + 1.33^2}\times\sqrt{1^2+2^2+1^2+2^2}} = -0.740$$

Exercise 9.3.2

(a) Clustering

i. (f, h) : J(f,h) = 0

ii. (b,d) : J(b,d) = ⅓

iii. (b,d,g) : J(d,g) = ⅓

iv. (b,d,g,a) : J(a,g) = ½

Final clusters : (f,h), (b,d,g,a), (c), (e)

(b) User-Cluster Matrix

	(f,h)	**(b,d,g,a)**	**(c)**	**(e)**
A	1	4.25		
B	1	1.75	4	
C	3.5	2.5	1	

(c) Cosine distance

$$C(A, B) = \frac{1\times1 + 4.25\times1.75}{\sqrt{1^2 + 4.25^2}\times\sqrt{1^2+1.75^2+4^2}} = 0.431$$

$$C(A, C) = \frac{1\times3.5 + 4.25\times2.5}{\sqrt{1^2 + 4.25^2}\times\sqrt{3.5^2+2.5^2+1^2}} = 0.733$$

$$C(B, C) = \frac{1\times3.5 + 1.75\times4.25+4\times1}{\sqrt{1^2+1.75^2+4^2}\times\sqrt{3.5^2+2.5^2+1^2}} = 0.6$$

(b) [20 pts] Implement collaborative filtering

(c) [20 pts] Movie Recommendation Challenge (incomplete)

Intialization

- Train-Test Split

  To check the performance of the algorithm and prevent over-fitting, I splitted nonzero elements into a train and test(validation) set.

```
nonzeros = utility_matrix.nonzero()
random_idx = np.random.permutation(num_data)
splitting_point = int(num_data * 0.8)
train_set = (
   nonzeros[0][random_idx[:splitting_point]],
   nonzeros[1][random_idx[:splitting_point]]
)
validation_set = (
   nonzeros[0][random_idx[splitting_point:]],
   nonzeros[1][random_idx[splitting_point:]]
)
```

- Normalization

  Normalize utility matrix with both user-mean and movie-mean matrix.

```
user_mean_matrix = np.apply_along_axis(
   lambda row: row[row > 0].mean() if sum(row) > 0  else 0.,
   axis=1,
   arr=utility_matrix,
)
movie_mean_matrix = np.apply_along_axis(
   lambda col: col[col > 0].mean() if sum(col) > 0 else 0.,
   axis=0,
   arr=utility_matrix,
)
mean_matrix = (user_mean_matrix[:, None] + movie_mean_matrix[None, :]) / 2
normalized_matrix = (utility_matrix - mean_matrix)
M = normalized_matrix * create_mask(train_set, utility_matrix.shape)
validation_matrix = normalized_matrix * create_mask(validation_set,
utility_matrix.shape)
```

- Perturbation

  Each element of V and U is random normal distribution with mean 0 and standard deviation 1.

```
u, v = num_users+1, num_movies+1
U = np.random.normal(0, 1, (u, m))
V = np.random.normal(0, 1, (m, v))
```

Optimization

- Permutation

  In order to apply optimization randomly, I permute the indices of U and V

```
U_idx = np.random.permutation(np.array(U.nonzero()).T).T
V_idx = np.random.permutation(np.array(V.nonzero()).T).T
```

- Optimization

  Since the number of movies is much bigger than the number of users, optimization is conducted proportional to its size. (e.g. adjust U optimization 1 times, and then adjust V optimization 244 times, and repeat it)

```
for i in tqdm(range(u * m)):
   # Adjust U
   r, s = U_idx[:, i]
   x1 = (V[s, :] ** 2).sum()
   x2 = (V[s, :] * M[s, :]).sum()
   x3 = (U[r, :] * (V @ V[s, :])).sum()
   U[r, s] += (x2 - x3) / x1
   # Adjust V
   for j in range(uv_rate):
       r, s = V_idx[:, i * uv_rate + j]
       y1 = (U[:, r] ** 2).sum()
       y2 = (U[:, r] * M[:, s]).sum()
       y3 = (V[:, s] * (U.T @ U[:, r])).sum()
       V[r, s] += (y2 - y3) / y1
```