EE412 Homework 2

Quang Minh Nguyen
Student ID: 20200854

November 3, 2022

Contents

1 Clustering

(a)

We consider the set of five strings

$$\{\texttt{a},\ \texttt{aa},\ \texttt{ab},\ \texttt{ac},\ \texttt{a...a}\ \},$$

where the last element is the character a repeated 100 times. The edit distance matrix, together with the sum and maximum of each row, is included in Table 1, where the entry at position (x, y) is the distance between strings x and y.

	a	aa	ab	ac	a...a	SUM	MAX
a	0	1	1	1	99	102	99
aa	1	0	2	2	98	103	98
ab	1	2	0	2	100	105	100
ac	1	2	2	0	100	105	100
a...a	99	98	100	100	0	397	100

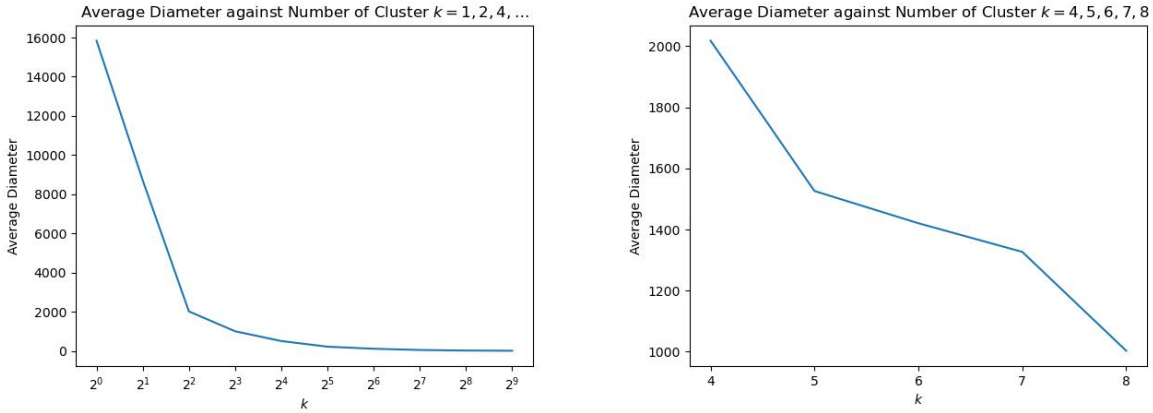
Table 1: The edit distance matrix of the chosen set for Problem 1(a)

Let us choose a clustroid. If the clustroid is to minimise the sum of distances to other points, we choose a, with $\text{SUM}(\texttt{a}) = 102$. However, if the clustroid is to minimise the maximum distance to other points, we choose aa instead, with $\text{MAX}(\texttt{aa}) = 98$.

(b) Implementation of k-Means algorithm using Spark

We set the threshold for a significant change of average diameters between two values of k to be 500.

Figure 1a shows how the number of clusters varies with $k = 1, 2, 4 \ldots$. The change between $k = 2^3$ and $k = 2^4$ is less than 500, so we focus on the range $k \in [4, 8]$. Within this range, we see that the change between $k = 6$ and $k = 8$ is less than 500, so we narrow down to $k \in [4, 6]$. The change from $k = 4$ to $k = 5$ is greater than the change from $k = 5$ to $k = 6$, so we identify $k = 5$ as the suitable value. For reference, the computed values are included in Table 2.



(a) $k = 1, 2, 4, \ldots$ (b) $k = 4, 5, 6, 7, 8$

Figure 1: Problem 1(b): average diameter vs. number of cluster plot for two ranges of k

k	Average diameter
1	15840.015935162375
2	8712.758000610926
4	2018.267391109598
5	1526.2137994123182
6	1420.7214130743844
7	1326.9455957023163
8	1003.3225742993731
16	504.61151331749284
32	216.90421459536506
64	110.92803717553373
128	48.64453075027023
256	19.19100439761892
512	8.172160832093697

Table 2: Problem 1(b): average diameter vs. number of cluster

2 Dimensionality Reduction

The answers are written down in formulas in subsections **Exercise 11.1.7** and **Exercise 11.3.1**. The more compact, raw output of the Python source code (subsection **Source code**) is included in subsection **Output**.

Exercise 11.1.7

(a) An approximate value of the principal eigenvector:

$$\begin{pmatrix} 0.19382266 \\ 0.47224729 \\ 0.8598926 \end{pmatrix} \tag{1}$$

(b) An estimate of the principal eigenvalue:

$$7.872983346207416 \tag{2}$$

(c) The newly constructed matrix is

$$\begin{pmatrix} 0.70423389 & 0.27936833 & -0.31216389 \\ 0.27936833 & 0.24418694 & -0.19707639 \\ -0.31216389 & -0.19707639 & 0.17859583 \end{pmatrix} \tag{3}$$

(d) Second eigenvalue:

$$1.0 \tag{4}$$

Second eigenvector:

$$\begin{pmatrix} 0.81649658 \\ 0.40824829 \\ -0.40824829 \end{pmatrix} \tag{5}$$

(e) New matrix:

$$\begin{pmatrix} 0.03756722 & -0.053965 & 0.02116944 \\ -0.053965 & 0.07752028 & -0.03040973 \\ 0.02116944 & -0.03040973 & 0.01192916 \end{pmatrix} \tag{6}$$

Third eigenvalue:

$$0.12701665379258315 \tag{7}$$

Third eigenvector:

$$\begin{pmatrix} 0.54384383 \\ -0.78122714 \\ 0.30646053 \end{pmatrix} \tag{8}$$

Exercise 11.3.1

(a)

$$M^\top M = \begin{pmatrix} 36 & 37 & 38 \\ 37 & 49 & 61 \\ 38 & 61 & 84 \end{pmatrix}, \quad MM^\top = \begin{pmatrix} 14 & 26 & 22 & 16 & 22 \\ 26 & 50 & 46 & 28 & 40 \\ 22 & 46 & 50 & 20 & 32 \\ 16 & 28 & 20 & 20 & 26 \\ 22 & 40 & 32 & 26 & 35 \end{pmatrix} \tag{9}$$

(b)

- For $M^\top M$:

  Eigenvalue 1:

  $$153.56699645995903 \tag{10}$$

  Eigenvector 1:

  $$\begin{pmatrix} -0.40928285 \\ -0.56345932 \\ -0.7176358 \end{pmatrix} \tag{11}$$

  Eigenvalue 2:

  $$15.433003540040923 \tag{12}$$

  Eigenvector 2:

  $$\begin{pmatrix} -0.81597848 \\ -0.12588456 \\ 0.56420935 \end{pmatrix} \tag{13}$$

- For $MM^\top$:

  Eigenvalue 1:

  $$153.56699645995909 \tag{14}$$

  Eigenvector 1:

  $$\begin{pmatrix} 0.29769568 \\ 0.57050856 \\ 0.52074297 \\ 0.32257847 \\ 0.45898491 \end{pmatrix} \tag{15}$$

  Eigenvalue 2:

  $$15.433003540040925 \tag{16}$$

  Eigenvector 2:

  $$\begin{pmatrix} -0.15906393 \\ 0.0332003 \\ 0.73585663 \\ -0.5103921 \\ -0.41425998 \end{pmatrix} \tag{17}$$

(c) The (compact) SVD for the original matrix M is $M = U\Sigma V^\top$, where

$$U = \begin{pmatrix} 0.29769568 & -0.15906393 \\ 0.57050856 & 0.0332003 \\ 0.52074297 & 0.73585663 \\ 0.32257847 & -0.5103921 \\ 0.45898491 & -0.41425998 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} 12.39221516 & 0. \\ 0. & 3.92848616 \end{pmatrix}, \tag{18}$$
$$V^\top = \begin{pmatrix} 0.40928285 & 0.56345932 & 0.7176358 \\ 0.81597848 & 0.12588456 & -0.56420935 \end{pmatrix}$$

We actually flipped the sign of V compared to the eigenvectors of $M^\top M$ produced by `numpy` in order to produce a correct SVD. But (1) why are we not guaranteed to find a correct SVD directly from such eigenvectors that `numpy` output?; and (2) why does the act of multiplying by -1 happen to give the correct answer?

To answer the first question, note that $M = U\Sigma V^\top$ only guarantees that V and U consist of, respectively, the eigenvectors for $M^\top M$ and $MM^\top$. The opposite is not true in general. The most we can say is that any two eigenvectors corresponding to the two non-zero eigenvalues of $M^\top M$ necessarily differ from the two columns of V by only scalar multiplications; the same goes for $MM^\top$ and U. These scalars reduce to either 1 or -1 if we only consider normalised eigenvectors. The output from `numpy`, accordingly, provides vectors that are either exactly or within a factor of -1 of V and U that constitute an SVD. This fact comes in handy for the second question: by testing with signs ± 1 for each eigenvector of $M^\top M$, we are guaranteed to eventually find an SVD. Multiplying both by -1, we finally find an SVD.

(d) The one dimensional approximation is

$$M' = \begin{pmatrix} 1.509889 & 2.0786628 & 2.64743661 \\ 2.89357443 & 3.98358126 & 5.0735881 \\ 2.64116728 & 3.63609257 & 4.63101787 \\ 1.63609257 & 2.25240715 & 2.86872172 \\ 2.32793529 & 3.20486638 & 4.08179747 \end{pmatrix} \tag{19}$$

(e)

$$\frac{\text{energy of approximation}}{\text{energy of original matrix}} = 0.9086804524257934 \tag{20}$$

Source code

The source code for both exercises is included right below this line:

```
"""
Filename: hw2_2.py
Author: Quang Minh Nguyen

Python source code for homework 2, task 2
```

```
References: EE412 lecture slides
"""

import sys
sys.stdout = open('hw2_2.txt', 'w')
import numpy as np


# EXERCISE 11.1.7
print("Exercise 11.1.7\n")


def get_eigenpairs(A, k, epsilon):
    """
    Returns a list containing the first k eigenpairs of A using
    power iteration until the difference between two consecutive
    vectors is less than epsilon. Also prints the eigenvalues and
    eigenvectors.

    Parameters:
        A: ndarray, a square matrix
        k: int, number of eigenpairs to return
        epsilon: float, threshold for convergence

    Returns:
        eigenpairs: list[(float, ndarray)], a list containing
            the first k eigenpairs of A
    """
    # gets a random vector
    x = np.array([1, 1, 1])
    # sequentially computes all eigenpairs of A using power iteration
    eigenpairs = []
    for i in range(k):
        # computes the eigenvector
        current_x = x
        while True:
            next_x = A @ current_x
            next_x = next_x / np.linalg.norm(next_x)
            if np.linalg.norm(current_x - next_x) < epsilon:
                break
            current_x = next_x
```

```
            eigenvector = next_x
            # computes the eigenvalue
            eigenvalue = eigenvector @ A @ eigenvector
            eigenpairs.append((eigenvalue, eigenvector))
            print(f"Eigenvalue {i+1}: {eigenvalue}")
            print(f"Eigenvector {i+1}: {eigenvector}")
            if i < k - 1:
                # prepares to find the next eigenpair
                A = A - eigenvalue * np.outer(eigenvector, eigenvector)
                print(f'The newly constructed matrix is \n{A}\n')
    return eigenpairs


# constants
EPSILON = 1e-8
K = 3

# defines the given matrix
B = np.array([[1, 1, 1], [1, 2, 3], [1, 3, 6]])
# computes and prints the eigenpairs
eigenpairs = get_eigenpairs(B, K, EPSILON)


print('-----------------------------------------------')

# EXERCISE 11.3.1
print("Exercise 11.3.1\n")

# constants
EPSILON = 1e-8

# defines the given matrix
M = np.array([
    [1, 2, 3],
    [3, 4, 5],
    [5, 4, 3],
    [0, 2, 4],
    [1, 3, 5]
])

# part (a)
```

```
print('(a)')
MTM = M.T @ M
MMT = M @ M.T
print(f'M^T M =\n{MTM},\nM M^T =\n{MMT}\n')

# part (b)
print('(b)')
print('Eigenpairs of M^T M:\n')
eigenvalues, eigenvectors = np.linalg.eig(MTM)
W = []
V = []
for eigenvalue, eigenvector in zip(eigenvalues, eigenvectors.T):
    if eigenvalue > EPSILON:
        W.append(eigenvalue)
        V.append(eigenvector)
        print(f'Eigenvalue: {eigenvalue}')
        print(f'Eigenvector: {eigenvector}\n')
V = np.array(V).T

print('Eigenpairs of M M^T:\n')
eigenvalues, eigenvectors = np.linalg.eig(MMT)
W = []
U = []
for eigenvalue, eigenvector in zip(eigenvalues, eigenvectors.T):
    if eigenvalue > EPSILON:
        W.append(eigenvalue)
        U.append(eigenvector)
        print(f'Eigenvalue: {eigenvalue}')
        print(f'Eigenvector: {eigenvector}\n')
U = np.array(U).T

# part (c)
print('(c)')
print('The (compact) SVD for the original matrix M:\n')
print(f'U =\n{U},\nSigma =\n{np.sqrt(np.diag(W))},\nV^T =\n{-V.T}\n')

# part (d)
print('(d)')
# keeps only the first singular value
U_1d = U[:, 0]
W_1d = np.sqrt(W[0])
```

```
V_1d = -V[:, 0]
# computes the approximation
M_1d = W_1d * np.outer(U_1d, V_1d)
print(f'The one-dimensional approximation is\n{M_1d}\n')

# part (e)
print('(e)')
print('The ratio of retained energy is\n', W[0]/(W[0] + W[1]))
```

Output

The raw output of the code above, i.e., `hw2_2.txt`, is

```
Exercise 11.1.7

Eigenvalue 1: 7.872983346207416
Eigenvector 1: [0.19382266 0.47224729 0.8598926 ]
The newly constructed matrix is
[[ 0.70423389  0.27936833 -0.31216389]
 [ 0.27936833  0.24418694 -0.19707639]
 [-0.31216389 -0.19707639  0.17859583]]

Eigenvalue 2: 1.0
Eigenvector 2: [ 0.81649658  0.40824829 -0.40824829]
The newly constructed matrix is
[[ 0.03756722 -0.053965    0.02116944]
 [-0.053965    0.07752028 -0.03040973]
 [ 0.02116944 -0.03040973  0.01192916]]

Eigenvalue 3: 0.12701665379258315
Eigenvector 3: [ 0.54384383 -0.78122714  0.30646053]
------------------------------------------------
Exercise 11.3.1

(a)
M^T M =
[[36 37 38]
 [37 49 61]
 [38 61 84]],
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]]

(b)
Eigenpairs of M^T M:

Eigenvalue: 153.56699645995903
Eigenvector: [-0.40928285 -0.56345932 -0.7176358 ]

Eigenvalue: 15.433003540040923
Eigenvector: [-0.81597848 -0.12588456  0.56420935]

Eigenpairs of M M^T:

Eigenvalue: 153.56699645995909
Eigenvector: [0.29769568 0.57050856 0.52074297 0.32257847 0.45898491]

Eigenvalue: 15.433003540040925
Eigenvector: [-0.15906393  0.0332003   0.73585663 -0.5103921  -0.41425998]

(c)
The (compact) SVD for the original matrix M:

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]]

(d)
The one-dimensional approximation is
[[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)
The ratio of retained energy is
 0.9086804524257934
```

3 Recommendation Systems

(a)

Exercise 9.3.1

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

Table 3: The utility matrix of Exercise 9.3.1

	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

Table 4: The transformed utility matrix for Exercise 9.3.1(a,b), with non-blanks as 1 (True) and blanks as 0 (False)

(a) This part uses Table 4.

- Between A, B: $|A \cap B|/|A \cup B| = 4/8 = 1/2$
- Between B, C: $|B \cap C|/|B \cup C| = 4/8 = 1/2$
- Between A, C: $|A \cap C|/|C \cup A| = 4/8 = 1/2$

So the Jaccard distance between each pair is $1/2$.

(b) This part uses Table 4.

- Between A, B: $\frac{A \cdot B}{|A||B|} = \frac{4}{\sqrt{6} \times \sqrt{6}} = 2/3$
- Between B, C: $\frac{B \cdot C}{|B||C|} = \frac{4}{\sqrt{6} \times \sqrt{6}} = 2/3$
- Between A, C: $\frac{A \cdot C}{|A||C|} = \frac{4}{\sqrt{6} \times \sqrt{6}} = 2/3$

So the cosine distance between each pair is $\arccos(2/3) = 48.19°$

	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

Table 5: The transformed utility matrix for Exercise 9.3.1(c,d), with $3, 4$, and 5 as 1 and $1, 2$, and blanks as 0

(c) This part uses Table 5.

- Between A, B: $|A \cap B|/|A \cup B| = 2/5$. So the Jaccard distance is $3/5$.
- Between B, C: $|B \cap C|/|B \cup C| = 1/6$. So the Jaccard distance is $5/6$.
- Between A, C: $|A \cap C|/|C \cup A| = 2/6 = 1/3$ So the Jaccard distance is $2/3$.

(d) This part uses Table 5.

- Between A, B: $\frac{A \cdot B}{|A||B|} = \frac{2}{\sqrt{4} \times \sqrt{3}} = \frac{1}{\sqrt{3}}$. So the cosine distance is $\arccos(1/\sqrt{3}) = 54.74°$.
- Between B, C: $\frac{B \cdot C}{|B||C|} = \frac{1}{\sqrt{3} \times \sqrt{4}} = \frac{1}{2\sqrt{3}}$. So the cosine distance is $\arccos(\frac{1}{2\sqrt{3}}) = 73.22°$.
- Between A, C: $\frac{A \cdot C}{|A||C|} = \frac{2}{\sqrt{4} \times \sqrt{4}} = \frac{1}{2}$. So the distance cosine is $\arccos(\frac{1}{2}) = 60°$.

	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

(a) The normalised utility matrix for Exercise 9.3.1(e)

	Average
A	10/3
B	7/3
C	3

(b) The row-wise average of the original utility matrix for Exercise 9.3.1(e)

Table 6: The normalised utility matrix and the row-wise average of the original utility matrix for Exercise 9.3.1(e)

(e) Refer to Table 6a. For convenience, we also included the average rating of each user in Table 6b. These average ratings are computed by

- For A: $\frac{4+5+5+1+3+2}{6} = 10/3$.
- For B: $\frac{3+4+3+1+2+1}{6} = 7/3$.
- For C: $\frac{2+1+3+4+5+3}{6} = 3$.

(f) This part uses Table 6. Note that the cosine similarity between two vectors is invariant under the scaling of any of them with a positive factor; this allows us to write the followings as if the vectors for A and B have been scaled by a factor of 3.

- Between A, B:

$$\frac{A \cdot B}{|A||B|} = \frac{5 \times 2 + 5 \times 2 + 7 \times 4 + 1 \times 4}{\sqrt{2^2 + 5^2 + 5^2 + 7^2 + 1^2 + 4^2}\sqrt{2^2 + 5^2 + 2^2 + 4^2 + 1^1 + 4^2}} = \frac{13\sqrt{55}}{165} \approx 0.584 \tag{21}$$

So the cosine distance is $\arccos(\frac{13\sqrt{55}}{165}) = 54.25°$.

- Between B, C:

$$\frac{B \cdot C}{|B||C|} = \frac{-5 \times 2 - 1 \times 1 - 4 \times 2}{\sqrt{2^2 + 5^2 + 2^2 + 4^2 + 1^2 + 4^2}\sqrt{1^2 + 2^2 + 1^2 + 2^2}} = \frac{-19\sqrt{165}}{330} \approx -0.740 \tag{22}$$

So the cosine distance is $\arccos(\frac{-19\sqrt{165}}{330}) = 137.7°$.

- Between A, C:

$$\frac{A \cdot C}{|A||C|} = \frac{-2 \times 1 - 1 \times 2}{\sqrt{2^2 + 5^2 + 5^2 + 7^2 + 1^2 + 4^2}\sqrt{1^2 + 2^2 + 1^2 + 2^2}} = \frac{-\sqrt{3}}{15} \approx 0.115 \tag{23}$$

So the cosine distance is $\arccos(\frac{-\sqrt{3}}{15}) = 96.63°$.

Exercise 9.3.2

	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

Table 7: The transformed utility matrix for Exercise 9.3.2(a), with $3, 4$, and 5 as 1 and $1, 2$, and blanks as 0

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

Table 8: The Jaccard distance matrix for items in Table 7. We write each pair only once and omit the distance between an item and itself.

(a) The utility matrix after replacing $3, 4$, and 5 with 1 and $1, 2$, and blanks with 0 is included in Table 7. The Jaccard distance matrix is shown in Table 8. We now describe the hierarchical clustering process, which is illustrated in Figure 2. We have eight cluters, with each item in its own cluster at first.

1. {f} and {h} are combined: their distance is 0. The clusters are now {a}, {b}, {c}, {d}, {e}, {f,h}, {g}.

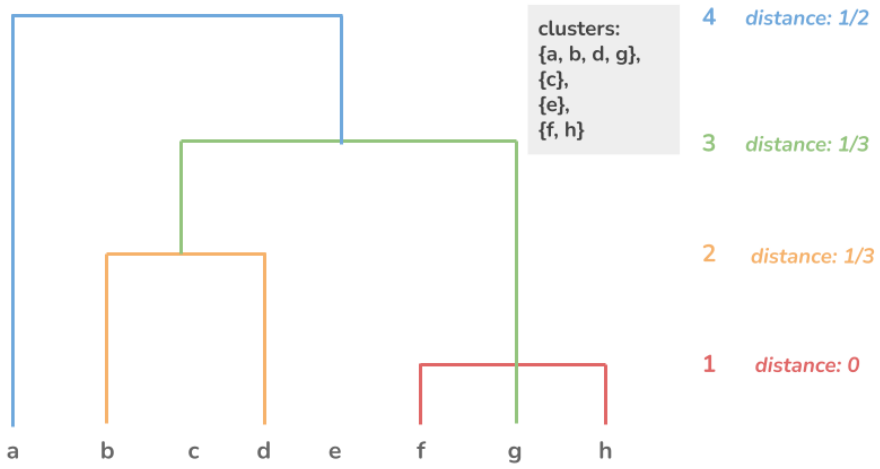


Figure 2: Hierarchical clustering in Exercise 9.3.2(a)

2. {b} and {d} are combined: their distance is $1/3$. The clusters are now {a}, {b,d}, {c}, {e}, {f,h}, {g}.
3. {b,d} and {g} are combined: their distance is $1/3$. The clusters are now {a}, {b,d,g}, {c}, {e}, {f,h}.
4. {a} and {b,d,g} are combined: their distance is $1/2$. The clusters are now {a,b,d,g}, {c}, {e}, {f,h}.

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

Table 9: The clustered utility matrix of Exercise 9.3.2(b)

(b) Refer to Table 9.

(c) Here we treat each blank entry as 0. The cosine distance is unaffected by scaling the vectors corresponding to A, B, and C by factors of $4, 3$, and 6, respectively, and that is what we shall do in the following computations:

- Between A, B:

$$\frac{A \cdot B}{|A||B|} = \frac{17 \times 7 + 4 \times 3 + 8 \times 6}{\sqrt{17^2 + 4^2 + 8^2}\sqrt{7^2 + 12^2 + 3^2 + 6^2}} \approx 0.604 \tag{24}$$

  So the distance is $\arccos(0.604) = 52.84°$.

- Between B, C:

$$\frac{B \cdot C}{|B||C|} = \frac{7 \times 20 + 12 \times 6 + 6 \times 21}{\sqrt{7^2 + 12^2 + 3^2 + 6^2}\sqrt{20^2 + 6^2 + 21^2}} \approx 0.740 \tag{25}$$

So the distance is $\arccos(0.740) = 42.27°$.

- Between A, C:

$$\frac{A \cdot C}{|A||C|} = \frac{17 \times 20 + 8 \times 21}{\sqrt{17^2 + 4^2 + 8^2}\sqrt{20^2 + 6^2 + 21^2}} \approx 0.893 \tag{26}$$

So the distance is $\arccos(0.893) = 26.75°$.

(b) Implementation of collaborative filtering

The output of

```
python hw2_3b.py path/to/ratings.txt
```

is

```
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
```

where the first 5 lines are for the *user-based* method, and the last 5 lines are for the *item-based* method.

(c) Movie recommendation challenge

I used SVD-based collaborative filtering. The k-fold cross validation root mean square error is around 0.962. Apart from the allowed libraries, I used `scipy` for processing sparse matrices.