CS550: Massive Data Mining and Learning
Homework 2

Due 11:59pm Saturday, March 23, 2019
Only one late period is allowed for this homework (11:59pm Sunday 3/24)

Submission Instructions

Assignment Submission Include a signed agreement to the Honor Code with this assignment. Assignments are due at 11:59pm. All students must submit their homework via Sakai. Students can typeset or scan their homework. Students also need to include their code in the final submission zip file. Put all the code for a single question into a single file.

Late Day Policy Each student will have a total of *two* free late days, and for each homework only one late day can be used. If a late day is used, the due date is 11:59pm on the next day.

Honor Code Students may discuss and work on homework problems in groups. This is encouraged. However, each student must write down their solutions independently to show they understand the solution well enough in order to reconstruct it by themselves. Students should clearly mention the names of all the other students who were part of their discussion group. Using code or solutions obtained from the web is considered an honor code violation. We check all the submissions for plagiarism. We take the honor code seriously and expect students to do the same.

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

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

I acknowledge and accept the Honor Code.

*(Signed)*Xinyang Wang_____

If you are not printing this document out, please type your initials above.

Answer to Question 1(a)

$(\mathbf{M}\mathbf{M}^{\mathrm{T}})^{\mathrm{T}} = (\mathbf{M}^{\mathrm{T}})^{\mathrm{T}}\mathbf{M}^{\mathrm{T}} = \mathbf{M}\mathbf{M}^{\mathrm{T}}$
$(\mathbf{M}^{\mathrm{T}}\mathbf{M})^{\mathrm{T}} = \mathbf{M}^{\mathrm{T}}\mathbf{M}$
So both are symmetric.

Because matrix M is of size $p \times q$, so $\mathbf{M}^{\mathrm{T}}$ is of size $q \times p$.
So $\mathbf{M}\mathbf{M}^{\mathrm{T}}$ is of size $p \times p$ and
$\mathbf{M}^{\mathrm{T}}\mathbf{M}$ is of size $q \times q$
so both are square

Because M is real, $\mathbf{M}^{\mathrm{T}}$ is real. So the products of $\mathbf{M}^{\mathrm{T}}\mathbf{M}$ and $\mathbf{M}\mathbf{M}^{\mathrm{T}}$ are real.

Answer to Question 1(b)

Suppose the eigenvalue of $\mathbf{M}\mathbf{M}^{\mathrm{T}}$ is λ and corresponding eigenvector is e.
$\mathbf{M}\mathbf{M}^{\mathrm{T}}e = \lambda e$

we multiply $\mathbf{M}^{\mathrm{T}}$ to the equation above from the left:
$\mathbf{M}^{\mathrm{T}}\mathbf{M}\mathbf{M}^{\mathrm{T}}e = \mathbf{M}^{\mathrm{T}}\lambda e$

we can rewrite it as below:
$(\mathbf{M}^{\mathrm{T}}\mathbf{M})(\mathbf{M}^{\mathrm{T}}e) = \lambda(\mathbf{M}^{\mathrm{T}}e)$

As we can see above, the eigenvalues the same and the eigenvectors are not.

Answer to Question 1(c)

$\mathbf{M}^{\mathrm{T}}\mathbf{M} = \mathbf{Q}\lambda\mathbf{Q}^{\mathrm{T}}$

Answer to Question 1(d)

$\mathbf{M}^{\mathrm{T}}\mathbf{M} = (\mathbf{U}\sum\mathbf{V}^{\mathrm{T}})^{\mathrm{T}} * (\mathbf{U}\sum\mathbf{V}^{\mathrm{T}}) = (\mathbf{V}\sum\mathbf{U}^{\mathrm{T}})(\mathbf{U}\sum\mathbf{V}^{\mathrm{T}}) = \mathbf{V}\sum(\mathbf{U}^{\mathrm{T}}\mathbf{U})\sum\mathbf{V}^{\mathrm{T}} = \mathbf{V}\sum^{2}\mathbf{V}^{\mathrm{T}}$

Answer to Question 1(e)(a)

The returned values are as follows

```
U is
 [[-0.27854301  0.5       ]
 [-0.27854301 -0.5       ]
 [-0.64993368  0.5       ]
 [-0.64993368 -0.5       ]]
s is
 [ 7.61577311  1.41421356]
VT is
 [[-0.70710678 -0.70710678]
 [-0.70710678  0.70710678]]
```

Answer to Question 1(e)(b)

The values of Evals and Evecs are as follows(after sorting and re-arranging):

```
Evals is
 [58.0, 2.0]
Evecs is
 [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
```

Answer to Question 1(e)(c)

As we can see from the results of our experiments, the V produced by SVD and the matrix of eigenvectors Evecs are equal in value.

Answer to Question 1(e)(d)

As we can see from the results of our experiments, the eigenvalues of $\mathbf{M}^{\mathrm{T}}\mathbf{M}$ are square of the singular values of M.

Answer to Question 2(a)

Target is to prove that: for $r' = Mr$, we have $w(r') = w(r)$
let's consider the first row of r' (r'_1) and the first row of r (r_1). We have $r'_1 = \sum_{i=1}^{n}(W_{i1}r_1) = (\sum_{i=1}^{n} W_{i1})r_1 = r_1$. And it is easy to see this relation holds for every row of r. So we can draw the conclusion that $r' = r$

Answer to Question 2(b)

Under the circumstance of $w(r) = 1$ will $w(r') = w(r)$.
Proof is shown below:
Assume $w(r') = w(r)$, and we have $\beta w(r) = \beta w(r')$. So $w(r') = \beta w(r') + (1 - \beta)$, which leads to $w(r') = 1$

Answer to Question 2(c)(a)

$r'_i = \beta \sum_{livej=1}^{n} M_{ij}r_j + \beta \sum_{deadj=1}^{n} M_{ij}r_j + (1 - \beta)/n$

Answer to Question 2(c)(b)

As we can see in the equation above, we can assume that 'dead' ends are in some sense 'live'.(we can think of it in the following way: each 'dead' end actually has n-1 arc to all the left nodes) Thus we can combine the first two parts in the above equation together, and rewrite it as follows:
$r'_i = \beta \sum_{allj=1}^{n} M_{ij}r_j + (1 - \beta)/n$
Thus if w(r) = 1, according to the conclusion of question 2(b), we have w(r') = 1 as well.

Answer to Question 3(a)

Before giving out the results, let's first look at the structure of our code: (detailed code is in the question3 file)

```
class PageRank():

    def __init__(self,n:int):
        '''Initialization '''

    def load_file(self,file):
        '''load file '''

    def generate_matrix(self,file):
    '''generate matrix M according to file \
    (during which remove the duplicates) '''

    def Iteration(self, times:int, beta):
        '''iterate to compute final scores '''
```

And the top 5 ids with scores are shown as follows:

```
the top 5 ids with the highest PageRank score
id:53, score:[ 0.0357312]
id:14, score:[ 0.03417091]
id:40, score:[ 0.03363009]
id:1, score:[ 0.03000598]
id:27, score:[ 0.02972014]
```

Answer to Question 3(b)

The bottom 5 ids with scores are shown as follows:

```
the bottom 5 ids with the lowest PageRank score
id:85, score:[ 0.00340969]
id:59, score:[ 0.00366986]
id:81, score:[ 0.00369535]
id:37, score:[ 0.0038082]
id:89, score:[ 0.00392247]
```

Answer to Question 4(a)

Before giving out the results, let's first look at the structure of our code: (detailed code is in the question4 file)

```
def mapper(line):
    '''spliting the line and then return \
    in the form of np.array'''

def closest_center(row, centers):
    '''for each data, find the closest center \
    and returned in the form (center,(data,1)) '''

def cost_computing(row, centers):
    '''compute the distance between data and \
    corresponding center '''

def plot_cost(c1_cost, c2_cost):
    '''plot cost as a function of loop '''
```

some attention 1: if you want to test the code, you should configure the pyspark environment first. Otherwise, you can use the jupyter offered by school server (`https://jupyter.cs.rutgers.edu/hub/login?next=%2Fhub%2Fuser%2Fhz333`).
some attention 2: before running the script, be sure to put the data.txt, c1.txt, c2.txt into the school server HDFS system by using this link: `https://data-services1.cs.rutgers.edu/#/login`. Because pyspark read the file from HDFS system on default.

Below we plot the cost function as a function of iterations.

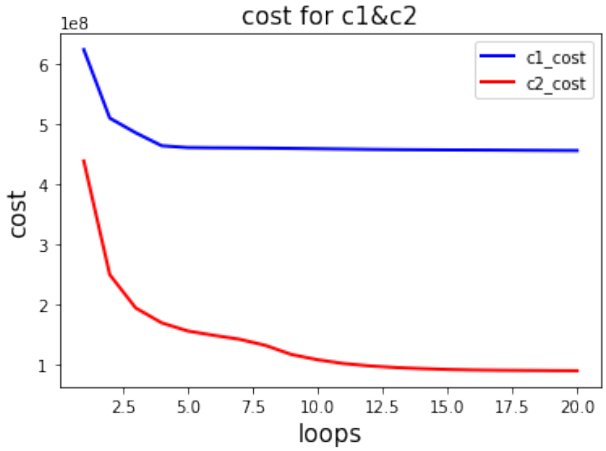

Figure 1: cost

Answer to Question 4(b)

In this part, we compute how much the cost decreases when using c1 and c2. And we also output the final centers in c1 and c2.

```
when using c1 as centroids, the percentage change in cost\
after 10 iterations is
0.2648391714456056

when using c2 as centroids, the percentage change in cost\
after 10 iterations is
0.7669795594605946
```

centers in c1
0.170.260.480.420.310.160.080.220.340.630.120.750.150.120.20.250.160.351.720.21.370.310.210.190.410.130.020.020.010.0
20.010.00.090.00.070.030.110.010.020.030.010.040.010.010.040.030.00.020.070.130.030.690.210.0710.62187.83868.940.75

0.190.180.590.00.50.310.160.140.450.440.180.570.210.270.540.290.180.461.690.480.910.180.430.210.180.040.010.00.00.010
.00.00.070.00.00.030.080.00.00.040.010.00.050.010.050.060.00.010.020.090.010.370.230.0663.93555.61770.690.85

0.130.120.30.160.30.110.090.130.390.380.10.510.130.110.10.280.120.311.530.240.940.970.150.180.310.150.020.010.00.020.
030.00.070.00.090.040.150.00.020.030.010.030.010.020.040.030.010.030.040.130.040.280.170.115.86110.281273.630.65

0.060.410.20.00.230.040.050.040.020.140.010.470.060.050.00.190.030.111.670.020.470.00.020.060.580.282.150.120.190.120
.10.070.090.070.120.110.090.030.10.080.070.290.030.140.570.240.00.040.020.120.020.20.020.011.695.3420.580.12

0.290.130.270.080.070.110.020.130.210.380.140.580.320.590.030.150.260.121.750.040.810.210.420.270.090.050.00.00.00.00
.00.00.280.00.00.030.160.00.010.020.020.010.00.010.040.050.010.050.070.30.010.260.330.1210.45388.564058.560.77

0.090.140.350.170.450.130.180.130.070.270.080.550.140.040.040.30.310.251.730.140.990.160.160.10.480.210.090.090.050.0
80.040.030.10.030.070.090.170.00.040.060.010.030.050.040.140.110.00.020.030.150.020.310.110.044.1252.17236.430.61

0.120.090.240.00.250.070.140.130.030.230.070.570.080.030.040.240.110.181.660.040.860.010.060.070.660.360.210.230.090.
160.070.060.130.060.160.130.190.010.110.060.030.10.080.10.270.240.010.030.020.150.010.250.050.052.6315.9773.250.36

0.110.110.310.00.430.150.140.110.040.210.050.520.090.010.030.310.170.21.660.090.990.030.140.090.520.330.20.160.080.12
0.050.050.090.050.110.110.150.010.10.070.050.040.080.040.240.150.00.030.030.140.020.250.080.053.0525.79140.450.49

0.160.140.340.050.330.120.090.140.250.350.090.750.120.10.080.270.240.141.470.121.030.320.140.180.740.170.020.020.010.
020.010.00.090.00.020.070.140.010.020.020.050.080.010.010.060.150.010.020.080.160.010.340.110.076.1776.09582.320.58

0.120.140.330.370.40.110.250.160.170.290.10.530.090.030.040.310.340.211.740.161.00.520.160.10.450.140.090.050.040.030
.050.020.040.020.050.060.140.020.040.10.020.030.030.030.110.110.010.030.180.140.010.290.120.18.3177.24376.580.59

Figure 2: final-center-in-c1

```
centers in c2
0.090.230.250.020.30.080.110.090.030.190.040.520.080.040.020.240.110.171.670.050.740.020.070.070.570.311.00.160.120.1
30.080.060.10.060.120.110.140.010.10.070.050.160.050.10.370.20.00.030.020.140.020.240.050.032.4817.2180.880.31

0.130.140.350.250.360.130.160.150.20.320.10.630.110.060.060.280.270.191.650.161.030.380.150.140.580.170.050.030.030.0
30.030.010.070.010.040.060.150.010.030.060.030.060.020.020.090.140.010.020.120.150.020.370.130.086.9380.76478.70.59

0.270.171.080.010.970.060.130.120.461.010.050.240.050.00.20.180.050.160.890.570.990.00.150.080.210.00.090.030.00.040.
00.00.00.020.00.00.050.00.00.070.00.00.00.00.00.00.00.00.00.020.00.660.180.0426.511302.691449.060.94

0.10.010.20.00.150.130.00.10.10.310.060.070.120.00.060.010.130.060.630.010.283.180.230.180.00.00.00.00.00.00.00.00.00
.050.00.010.00.00.030.00.00.00.00.00.060.00.00.00.370.10.010.140.030.015.19493.09113.671.0

0.00.00.040.00.020.00.00.00.020.060.020.00.00.020.00.00.00.040.00.00.00.00.00.00.00.00.00.00.00.00.00.00.040.00.00.06
0.00.00.00.00.230.00.00.020.00.40.00.040.040.160.080.010.060.013.0343.55648.50.0

0.150.180.440.150.360.220.120.170.410.50.130.640.150.110.320.30.140.441.570.341.130.660.260.180.290.110.020.010.00.01
0.020.00.070.00.080.030.110.00.020.030.010.020.030.010.040.040.010.020.040.120.030.370.170.099.14234.291160.10.75

0.240.210.310.010.250.090.030.150.320.410.160.550.270.480.110.260.270.161.890.050.840.110.320.280.360.090.00.00.00.01
0.010.00.140.00.010.060.160.00.010.040.020.00.010.020.050.080.00.030.040.110.020.280.30.086.84287.532168.890.67

0.320.160.290.090.080.110.020.150.240.420.150.660.360.690.030.160.290.121.970.050.910.030.470.310.10.060.010.00.00.00
.00.00.310.00.00.030.180.00.010.020.020.010.00.020.040.040.010.050.030.110.00.260.350.095.57191.043245.110.78

0.190.00.380.00.00.190.00.00.00.00.190.190.00.00.00.380.00.191.140.00.380.00.00.00.00.00.00.00.00.00.00.00.00.00.00.0
0.00.00.00.00.00.00.00.00.00.00.00.00.00.010.010.00.00.00.02.3821.015841.01.0

0.00.00.00.040.00.00.00.00.00.00.00.00.00.00.00.00.020.00.00.020.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.0
0.00.00.00.00.00.00.00.00.00.00.01.029.750.01.541.782.0239.579989.010062.01.0
```

Figure 3: final-center-in-c1

As we can see from above, initialization using c2 has a better effect than initialization using c1. This is because we are likely to choose points that are close to each other as the initial centroids, which leads to bigger cost.