ca11

March 2, 2023

0.1 CS/ECE/ME532 Period 11 Activity

0.1.1 Preambles

```
[ ]: %matplotlib inline
     # to enable 3D plot interaction
     import numpy as np # numpy
     from pprint import pprint as pprint # pretty print
     from scipy.io import loadmat # load & save data
     from scipy.io import savemat
     import matplotlib.pyplot as plt # plot
     from mpl_toolkits import mplot3d
     np.set_printoptions(formatter={'float': lambda x: "{0:0.2f}".format(x)})
```

K-means has some 'random' components in it. You will get different results depending on your luck. Even when you run an identical code, you will see some different results from your peers. So… we need the following line of code to start with:

```
[ ]: np.random.seed(2)
```

Indeed, one may be tempted to try so many random seeds until you get a good performance!

Don't do that :-)… Some subfields in ML are suffering from "reproduction crisis" partially due to this: See these for more details https://arxiv.org/abs/1709.06560 https://www.nature.com/articles/d41586-019-03895-5 https://www.wired.com/story/artificial-intelligence-confronts-reproducibility-crisis/

And see the following figure from the attached paper:

0.1.2 1. K-means and SVD for rating prediction

We return to the movies rating problem considered previously. The movies and ratings from your friends on a scale of 1-10 are:

Movie	Jake	Jennifer	Jada	Theo	Ioan	Bo	Juanita
Star Trek	4	7	2	8	7	4	2
Pride and Prejudice	9	3	5	6	10	5	5

Movie	Jake	Jennifer	Jada	Theo	Ioan	Bo	Juanita
The Martian	4	8	3	7	6	4	1
Sense and Sensibility	9	2	6	5	9	5	4
Star Wars: Empire Strikes	4	9	2	8	7	4	1

Run the following code block to create a numpy array X

```
[ ]: X = np.array([
         [4,7,2,8,7,4,2],
         [9,3,5,6,10,5,5],
         [4,8,3,7,6,4,1],
         [9,2,6,5,9,5,4],
         [4,9,2,8,7,4,1],
         ], float)
     print(X)
```

```
[[4.00 7.00 2.00 8.00 7.00 4.00 2.00]
 [9.00 3.00 5.00 6.00 10.00 5.00 5.00]
 [4.00 8.00 3.00 7.00 6.00 4.00 1.00]
 [9.00 2.00 6.00 5.00 9.00 5.00 4.00]
 [4.00 9.00 2.00 8.00 7.00 4.00 1.00]]
```

`float` is necessary as the array will only hold integers otherwise

Also, we load the K-mean algorithm we implemented in the last activity.

```
[ ]: def dist(x, y):
         return (x-y).T@(x-y)

     def kMeans(X, K, maxIters = 20):
         X_transpose = X.transpose()
         centroids = X_transpose[np.random.choice(X.shape[0], K)]
         for i in range(maxIters):
             # Cluster Assignment step
             C = np.array([np.argmin([dist(x_i, y_k) for y_k in centroids]) for x_i␣
      ↪in X_transpose])
             # Update centroids step
             for k in range(K):
                 if (C == k).any():
                     centroids[k] = X_transpose[C == k].mean(axis = 0)
                 else: # if there are no data points assigned to this certain␣
      ↪centroid
                     centroids[k] = X_transpose[np.random.choice(len(X))]
         return centroids.transpose() , C
```

Note that `(x-y).T@(x-y)` is the squared L^2 norm of $x - y$: since x and y are 1-d numpy arrays, the .T does not actually impact the code.

1 a) Use the K-means algorithm to represent the columns of X with two clusters.

[ ]:
```
centroids_2, C_2 =  kMeans(X, 2)
print('centroids = \n', centroids_2)
print('centroid assignment = \n', C_2)
```

```
centroids =
 [[3.00 7.33]
 [6.00 6.33]
 [3.00 7.00]
 [6.00 5.33]
 [2.75 8.00]]
centroid assignment =
 [0 1 0 1 1 0 0]
```

1 b) Express the rank-2 approximation to X based on this cluster as TW^T where the columns of T contains the cluster centers and W is a vector of ones and zeros. Compare the rank-2 clustering approximation to the original matrix.

[ ]:
```
# Construct rank-2 approximation using cluster
centroids_transposed_2 = centroids_2.transpose()
X_hat_2 = centroids_2@centroids_transposed_2
print('Rank-2 Approximation = \n', X_hat_2)
```

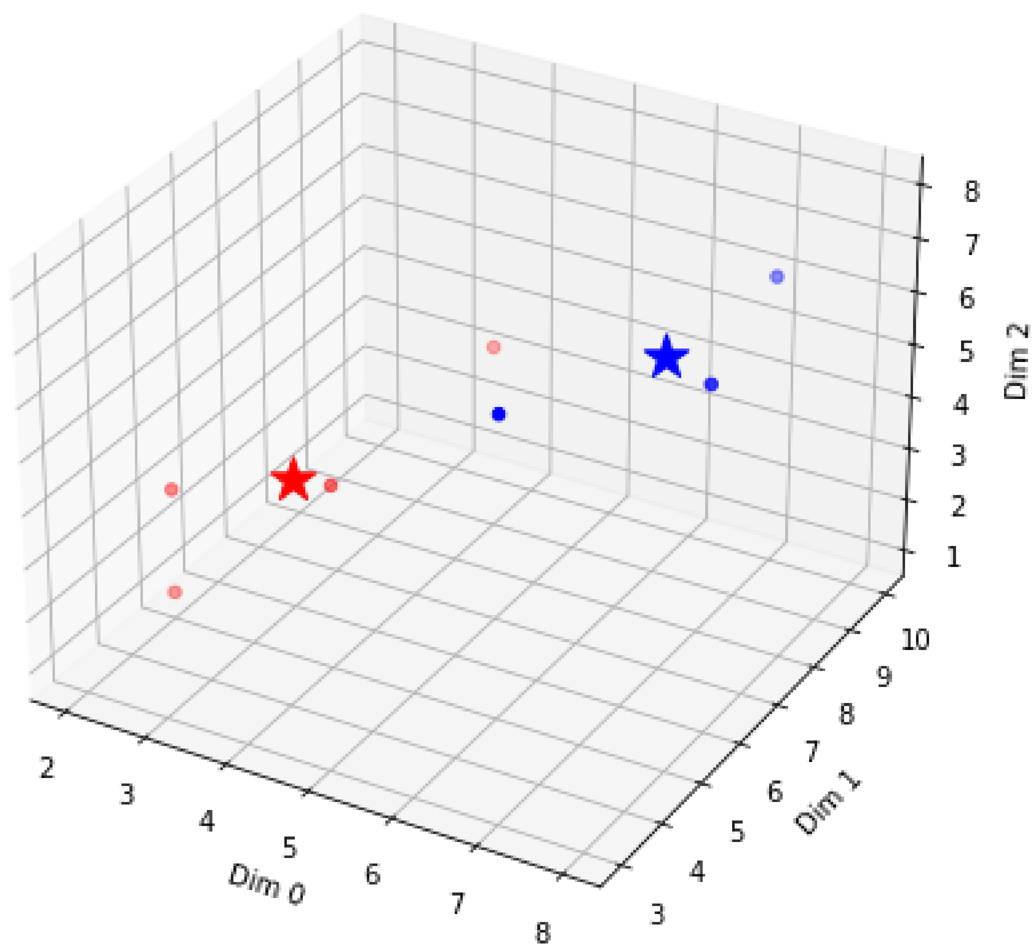
```
Rank-2 Approximation =
 [[62.78 64.44 60.33 57.11 66.92]
 [64.44 76.11 62.33 69.78 67.17]
 [60.33 62.33 58.00 55.33 64.25]
 [57.11 69.78 55.33 64.44 59.17]
 [66.92 67.17 64.25 59.17 71.56]]
```

1 c) Play with the following code! You can pick three dimensions to look at by modifying `coordinates_to_plot`. Just have fun with it.

[ ]:
```
fig = plt.figure(figsize = (10, 7))
ax = plt.axes(projection ="3d")
coordinates_to_plot = [0,1,2]
color_array = np.array(['red', 'blue'])
ax.scatter3D(
            X[coordinates_to_plot[0],:], # x
            X[coordinates_to_plot[1],:], # y
            X[coordinates_to_plot[2],:], # y
            color=color_array[C_2] # color depends on cluster idx
            )
for i in range(2):
    ax.scatter3D(
            centroids_2[coordinates_to_plot[0],i], # x
            centroids_2[coordinates_to_plot[1],i], # y
            centroids_2[coordinates_to_plot[2],i], # y
```

```
                marker='*', # star instead of circle
                s=300, # size
                color=color_array[i] # color
                )

ax.set_xlabel('Dim %d'%coordinates_to_plot[0])
ax.set_ylabel('Dim %d'%coordinates_to_plot[1])
ax.set_zlabel('Dim %d'%coordinates_to_plot[2])
```

[ ]: Text(0.5, 0, 'Dim 2')



1 d) Repeat a)–c) with $K = 3$.

[ ]:
```
centroids_3, C_3 = kMeans(X, 3)
print('centroids = \n', centroids_3)
```

```
print('centroid assignment = \n', C_3)
```

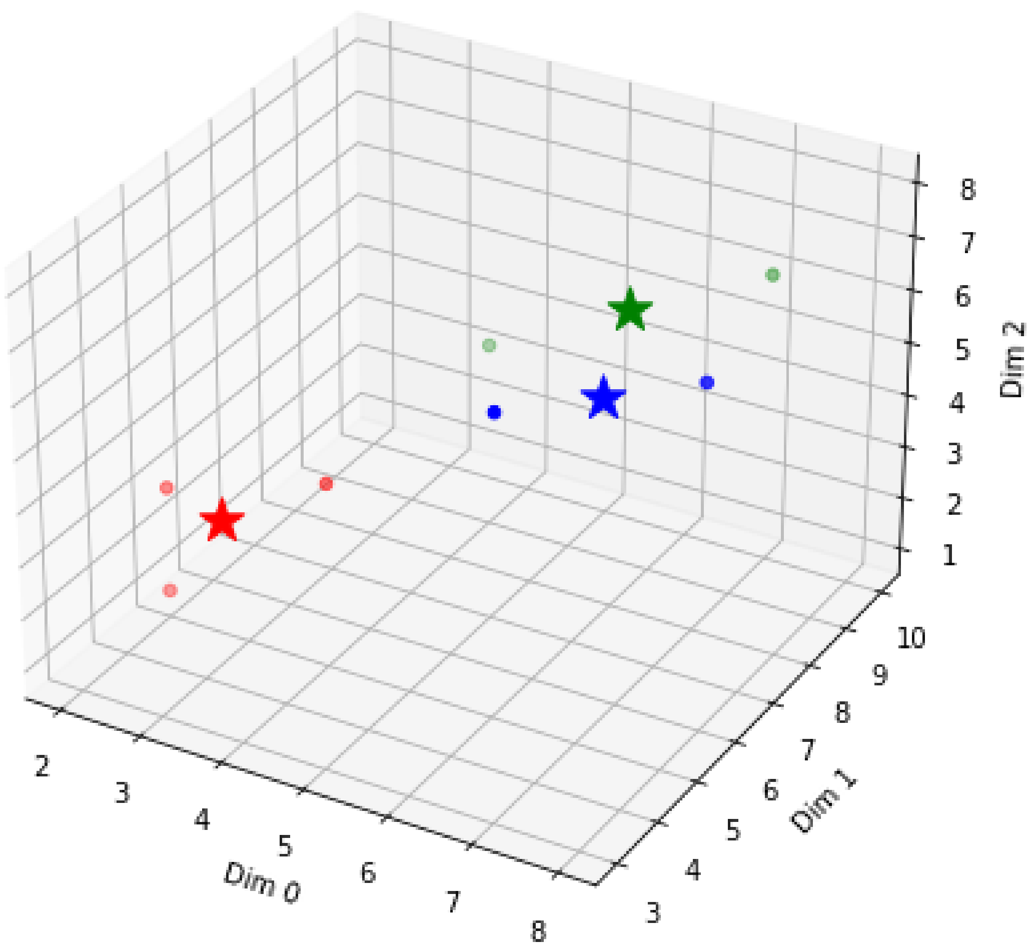
```
centroids =
 [[2.67 7.50 5.50]
 [5.00 4.50 9.50]
 [2.67 7.50 5.00]
 [5.00 3.50 9.00]
 [2.33 8.50 5.50]]
centroid assignment =
 [2 1 0 1 2 0 0]
```

[ ]:
```
# Construct rank-3 approximation using cluster
centroids_transposed_3 = centroids_3.transpose()
X_hat_3 = centroids_3@centroids_transposed_3
print('Rank-3 Approximation = \n', X_hat_3)
```

```
Rank-3 Approximation =
 [[93.61 99.33 90.86 89.08 100.22]
 [99.33 135.50 94.58 126.25 102.17]
 [90.86 94.58 88.36 84.58 97.47]
 [89.08 126.25 84.58 118.25 90.92]
 [100.22 102.17 97.47 90.92 107.94]]
```

[ ]:
```
fig = plt.figure(figsize = (10, 7))
ax = plt.axes(projection ="3d")
coordinates_to_plot = [0,1,2]
color_array = np.array(['red', 'blue', 'green'])
ax.scatter3D(
            X[coordinates_to_plot[0],:], # x
            X[coordinates_to_plot[1],:], # y
            X[coordinates_to_plot[2],:], # y
            color=color_array[C_3] # color depends on cluster idx
            )
for i in range(3):
    ax.scatter3D(
            centroids_3[coordinates_to_plot[0],i], # x
            centroids_3[coordinates_to_plot[1],i], # y
            centroids_3[coordinates_to_plot[2],i], # y
            marker='*', # star instead of circle
            s=300, # size
            color=color_array[i] # color
            )

ax.set_xlabel('Dim %d'%coordinates_to_plot[0])
ax.set_ylabel('Dim %d'%coordinates_to_plot[1])
ax.set_zlabel('Dim %d'%coordinates_to_plot[2])
```

[ ]: Text(0.5, 0, 'Dim 2')



1 e) SVD can be also used to find T and W such that $X \approx TW$. Assume that you are given the SVD of X, i.e., $X = USV^T$. Find SVD-based T and W as a function of U, S, V (In an equation form, not numbers.) Recall that T is a 5-by-r matrix with orthonormal columns. $T = US$

$W = V$

1 f) Find T, W and the rank-r approximation to X for $r = 2$. What aspects of the ratings does the first taste vector capture? What about the second taste vector?

[ ]:
```
U, s, VT = np.linalg.svd(X, full_matrices=True)
S_matrix = np.zeros_like(X)
np.fill_diagonal(S_matrix, s)
```

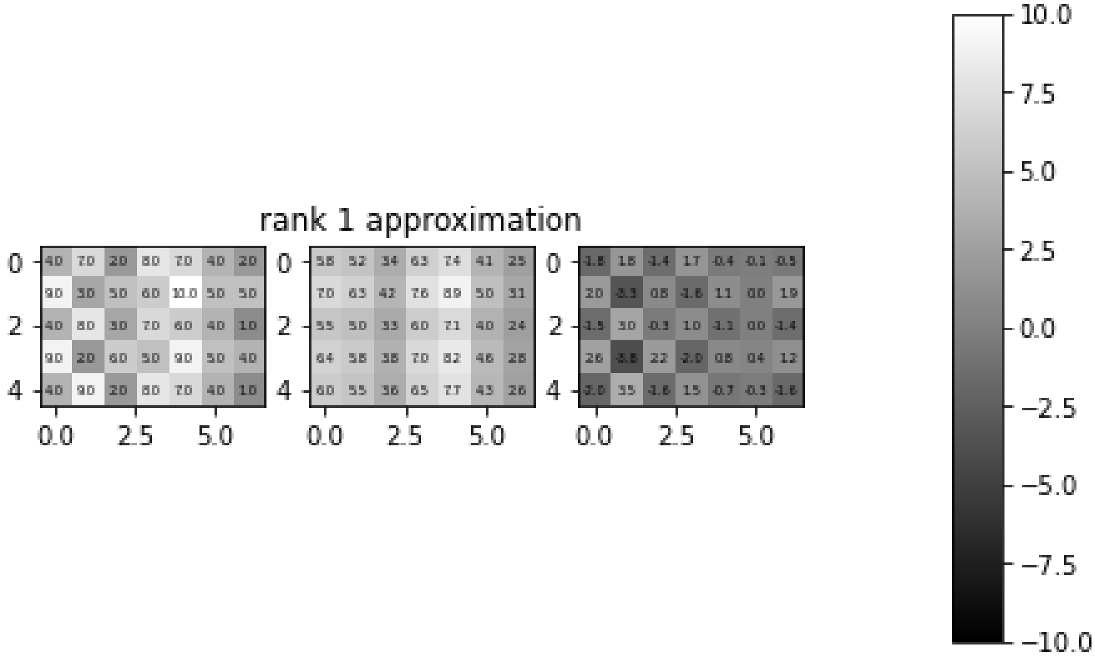
```
## Fill in the blank using U, S_matrix, and VT
T = U@S_matrix
W = VT

for r in range(0,2):
    T_r = T[:,0:r+1] ## Choose the first r columns of T
    W_r = W[0:r+1,:] ## Choose the first r rows of W
    print(T_r)
    print(W_r)
```

```
[[-13.79]
 [-16.66]
 [-13.25]
 [-15.36]
 [-14.37]]
[[-0.42 -0.38 -0.25 -0.46 -0.54 -0.30 -0.18]]
[[-13.79 -3.24]
 [-16.66 4.77]
 [-13.25 -3.78]
 [-15.36 5.61]
 [-14.37 -4.93]]
[[-0.42 -0.38 -0.25 -0.46 -0.54 -0.30 -0.18]
 [0.44 -0.69 0.29 -0.34 0.18 0.04 0.30]]
```

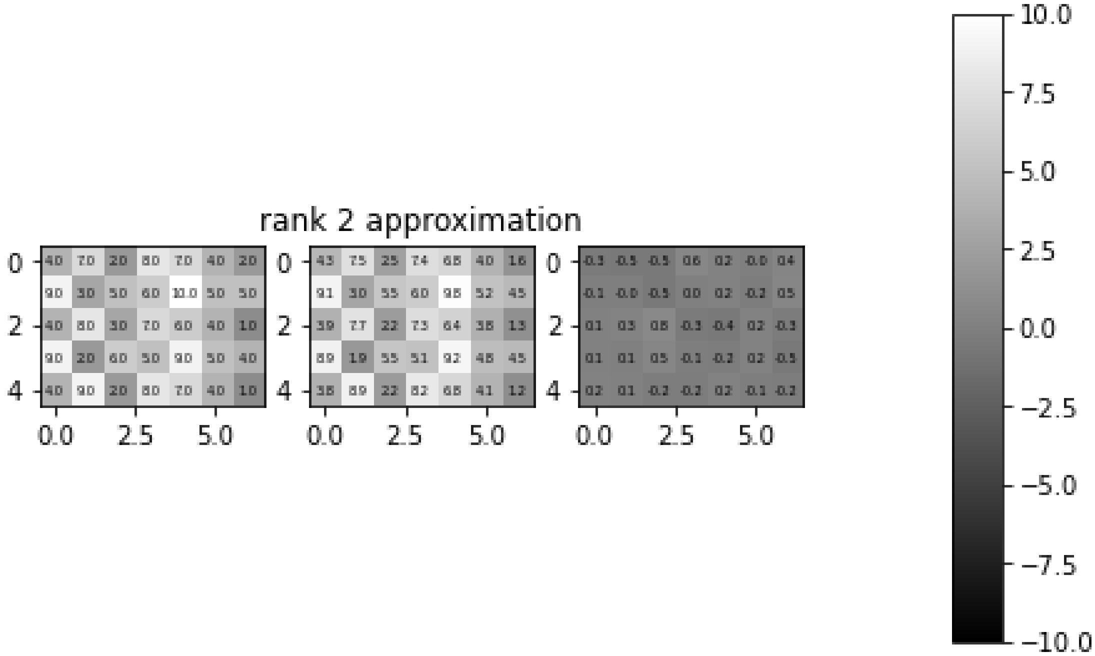
1 g) The following code visualizes the rank-r approximation for an increasing value of r. When does the approximation become exact? Why?

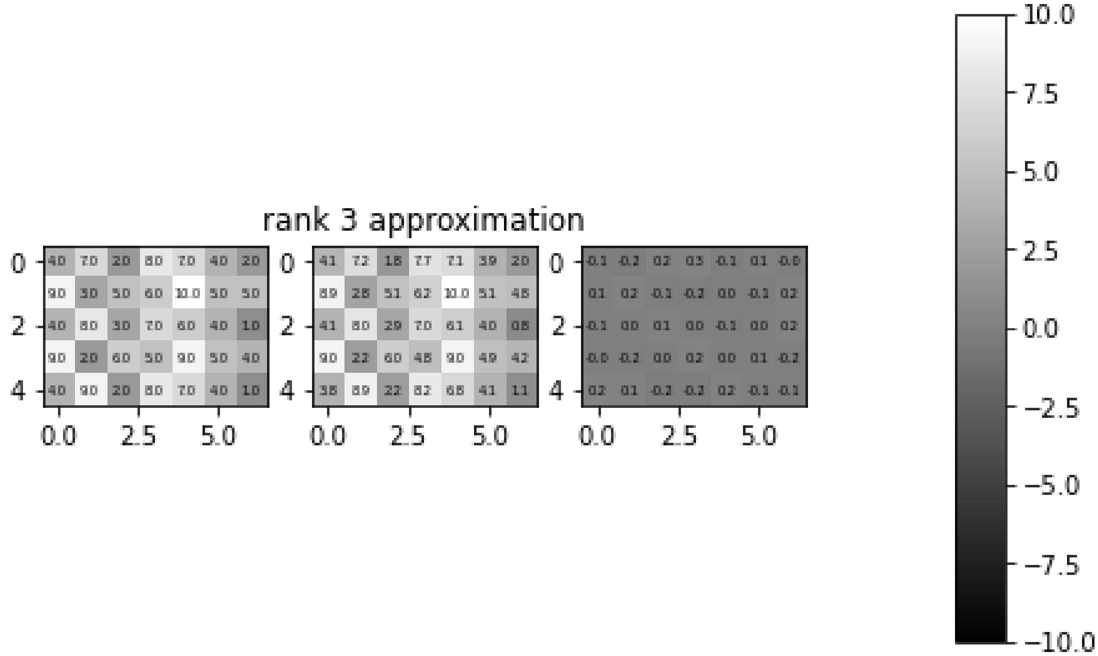
[ ]:
```
for r in range(0,7):
    T_r = T[:,0:r+1] ## Choose the first r columns of T
    W_r = W[0:r+1,:] ## Choose the first r rows of W
    X_rank_r_approx = T_r@W_r
    fig, ax = plt.subplots(1,3,figsize=(5.5, 5))
    for (j,i),label in np.ndenumerate(X):
        ax[0].text(i,j,np.round(label,1),ha='center',va='center', size=5)
    im = ax[0].imshow(X, vmin=-10, vmax=10, interpolation='none', cmap='gray')
    for (j,i),label in np.ndenumerate(X_rank_r_approx):
        ax[1].text(i,j,np.round(label,1),ha='center',va='center', size=5)
    im = ax[1].imshow(X_rank_r_approx, vmin=-10, vmax=10, interpolation='none',
 ↪cmap='gray')
    for (j,i),label in np.ndenumerate(X-X_rank_r_approx):
        ax[2].text(i,j,np.round(label,1),ha='center',va='center', size=5)
    im = ax[2].imshow(X-X_rank_r_approx, vmin=-10, vmax=10,
 ↪interpolation='none', cmap='gray')

    cbar_ax = fig.add_axes([1.05, 0.15, 0.05, 0.7])
    fig.colorbar(im, cax=cbar_ax)
    ax[1].set_title("rank %d approximation" % (r+1))
```

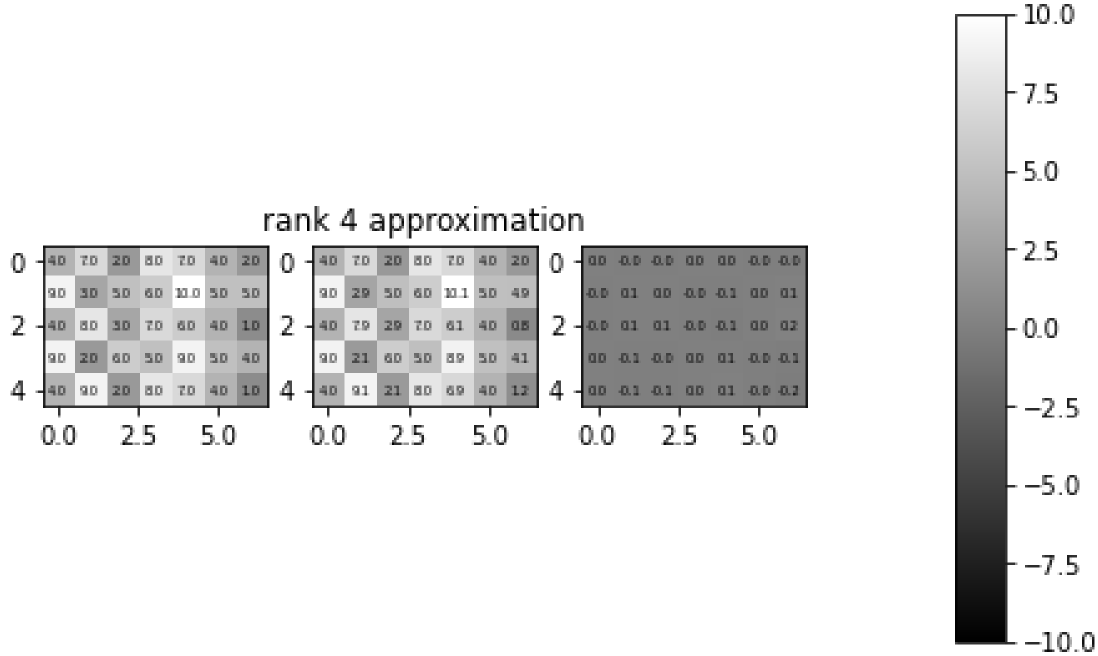
rank 1 approximation

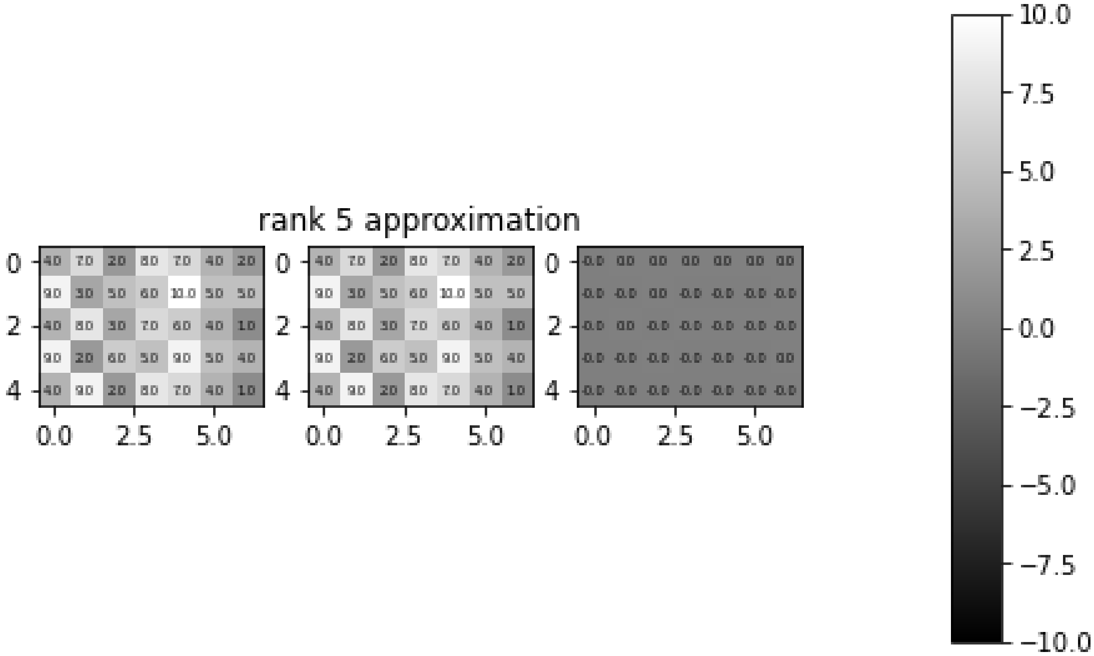


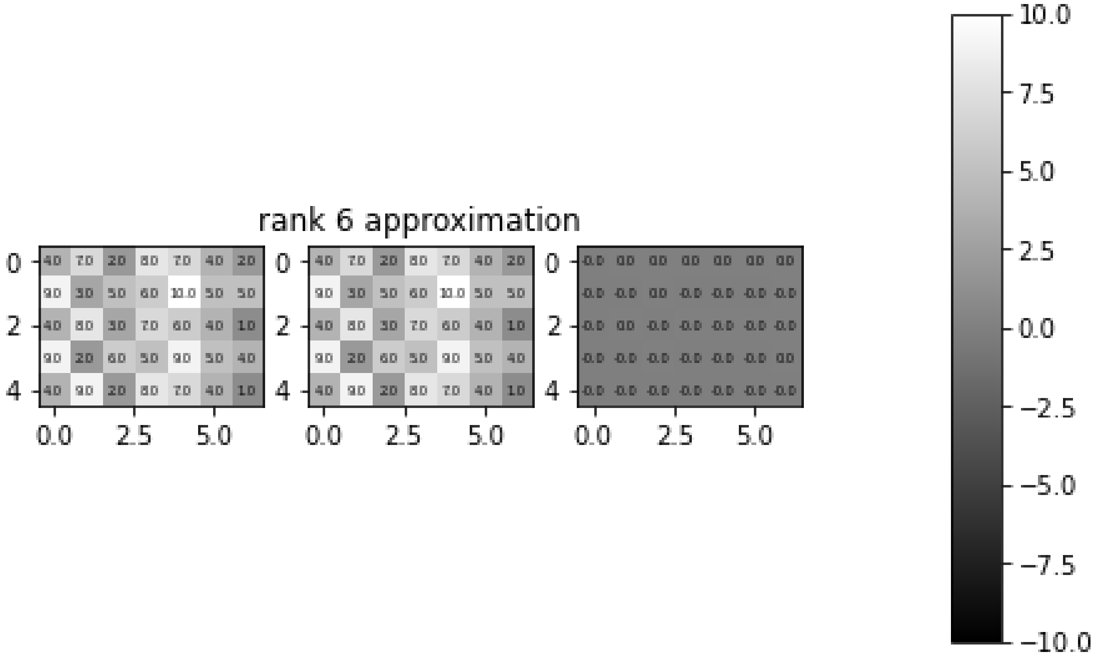
rank 2 approximation

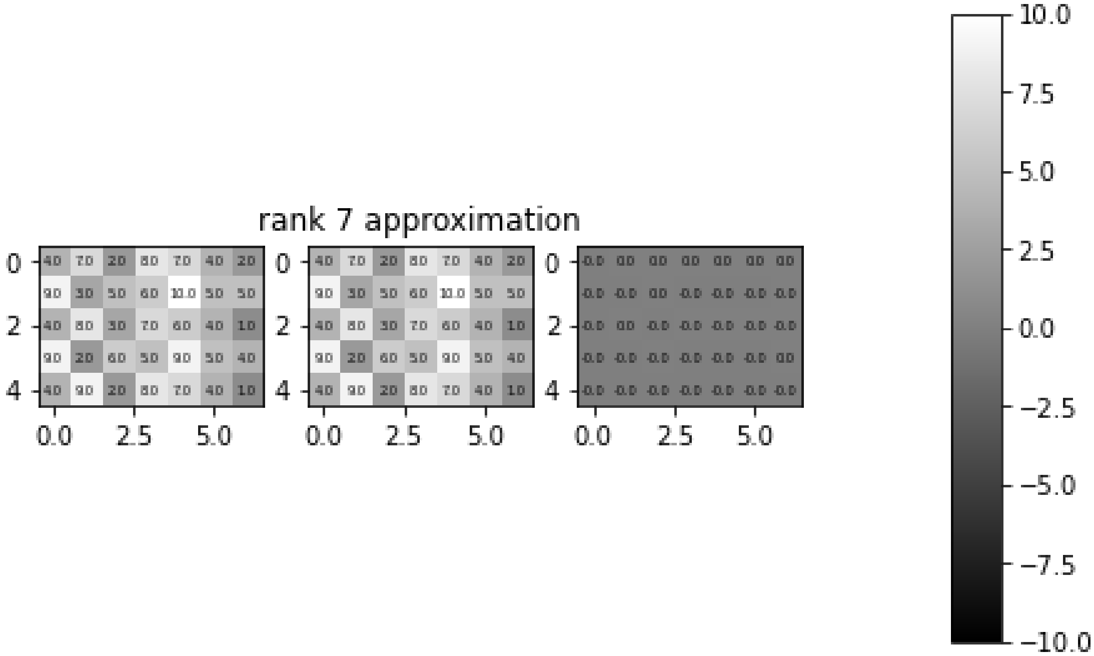



rank 3 approximation


rank 4 approximation


rank 5 approximation


rank 6 approximation



1 g) Your friend Jon rates Star Trek 6 and Pride and Prejudice 4. Assume a two-column taste matrix T. Formulate a system of equations that can find Jon's weight vector. Write down the least square solution. $T = T[:2, :2]$

$y = y[:2, :2]$

$W_{jon} = (T^T T)^{-1} T^T y$

1 h) Using this weight vector, how can we predict Jon's ratings for all five movies, including the remaining three movies? $W_{jon} = T(T^T T)^{-1} T T y$

1 i) Predict Jon's ratings for all the five movies with different choices of the taste matrix.

- Choice 1: T is the two centroids of the K-means result with $K = 2$
- Choice 2: T is the first two centroids of the K-means result with $K = 3$
- Choice 3: T is the first two SVD-based taste vectors

[ ]:
```
y = np.array([[6],[4]])

## Choice 1: K-means (K=2) based taste matrix T
T = centroids_2[:,0:2] # fill in the blank
T_12 = T[0:2,:]
print(T@np.linalg.inv(T_12.T@T_12)@T_12.T@y)

## Choice 2: K-means (K=3) based taste matrix T
```

```
T = centroids_3[:,0:2] # fill in the blank
T_12 = T[0:2,:]
print(T@np.linalg.inv(T_12.T@T_12)@T_12.T@y)

## Choice 3: SVD based taste matrix T
T = T[:,0:2] # fill in the blank
T_12 = T[0:2,:]
print(T@np.linalg.inv(T_12.T@T_12)@T_12.T@y)
```

```
[[6.00]
 [4.00]
 [5.68]
 [3.04]
 [6.73]]
[[6.00]
 [4.00]
 [6.00]
 [3.24]
 [6.72]]
[[6.00]
 [4.00]
 [6.00]
 [3.24]
 [6.72]]
```