532

Activity 11 DEVIN BRESSER

CS/ECE/ME532 Activity 11

Estimated time: 45 mins for Q1 and 20 mins for Q2.

1. See `period_11.ipynb`.

2. Let a 4-by-2 matrix $\boldsymbol{X}$ have SVD $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$ where $\boldsymbol{U} = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}$, $\boldsymbol{S} = \begin{bmatrix} 1 & 0 \\ 0 & \gamma \end{bmatrix}$, and $\boldsymbol{V} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$.

a) Express the solution to the least-squares problem $\arg\min_{\boldsymbol{w}} ||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2$ as a function of $\boldsymbol{U}$, $\boldsymbol{S}, \boldsymbol{V}$, and $\boldsymbol{y}$.

b) Let $\boldsymbol{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$. Find the weights $\boldsymbol{w}$ that minimize $||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2$ as a function of γ. Calculate $||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2$ and $||\boldsymbol{w}||_2^2$ as a function of γ for this value of $\boldsymbol{w}$. What happens to $||\boldsymbol{w}||_2^2$ as $\gamma \to 0$?

c) Now consider a "low-rank" inverse. Instead of writing

$$(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T = \sum_{i=1}^{p} \frac{1}{\sigma_i}\boldsymbol{v}_i\boldsymbol{u}_i^T$$

where p is the number of columns of $\boldsymbol{X}$ (assumed less than the number of rows), we approximate

$$(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T \approx \sum_{i=1}^{r} \frac{1}{\sigma_i}\boldsymbol{v}_i\boldsymbol{u}_i^T$$

In this approximation we only invert the largest r singular values, and ignore all of them smaller than σ_r. If $r = 1$, use the low-rank inverse to find $\boldsymbol{w}$, $||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}||_2^2$, and $||\boldsymbol{w}||_2^2$ when $\boldsymbol{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ as in part b). Compare $||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}||_2^2$, and $||\boldsymbol{w}||_2^2$ to the results for part b).

2. Let a 4-by-2 matrix $\boldsymbol{X}$ have SVD $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$ where $\boldsymbol{U} = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}$, $\boldsymbol{S} = \begin{bmatrix} 1 & 0 \\ 0 & \gamma \end{bmatrix}$, and $\boldsymbol{V} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$.

a) Express the solution to the least-squares problem $\arg\min_{\boldsymbol{w}} ||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2$ as a function of $\boldsymbol{U}$, $\boldsymbol{S}, \boldsymbol{V}$, and $\boldsymbol{y}$.

b) Let $\boldsymbol{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$. Find the weights $\boldsymbol{w}$ that minimize $||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2$ as a function of γ. Calculate $||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2$ and $||\boldsymbol{w}||_2^2$ as a function of γ for this value of $\boldsymbol{w}$. What happens to $||\boldsymbol{w}||_2^2$ as $\gamma \to 0$?

a.) $\underline{X}$ has SVD $\underline{U}\,\underline{\Sigma}\,\underline{V}^T$

$\underset{4\times 2}{}$ 4x2   4x2 2x2 2x2

$$\underset{w}{\text{argmin}} \; ||\underline{X}\,\underline{w} - \underline{y}||_2^2$$

Solution (traditional form): $\underline{w}_{min} = (X^T \underline{X})^{-1} \underline{X}^T \underline{y}$

Solution (SVD form) : $\underline{w}_{min} = (\underline{V}\,\underline{\Sigma}^T \cancel{\underline{U}^T} \overset{I}{\cancel{\underline{U}}}\,\underline{\Sigma}\,\underline{V}^T)^{-1} \cdot \underline{V}\,\underline{\Sigma}^{\cancel{T}}\,\underline{U}^T \underline{y}$

econ.

$\rightarrow \underline{w}_{min} = (\underline{V}\,\underline{\Sigma}^T \underline{\Sigma}\,\underline{V}^T)^{-1} \underline{V}\,\underline{\Sigma}\,\underline{U}^T \underline{y}$

$= (\underline{V}\,\underline{\Sigma}^2 \underline{V}^T)^{-1} \underline{V}\,\underline{\Sigma}\,\underline{U}^T \underline{y}$

goes to $\underline{\Sigma}^{-1}$ if diagonal matrix

$= \underline{V}\,\underline{\Sigma}^{-2} \overset{I}{\cancel{\underline{V}^T \underline{V}}}\,\underline{\Sigma}\,\underline{U}^T \underline{y}$

$= \underline{V}\,\underline{\Sigma}^{-2}\,\underline{\Sigma}\,\underline{U}^T \underline{y}$

$$\boxed{\underline{w}_{min} = \underline{V}\,\underline{\Sigma}^{-1}\underline{U}^T \underline{y}}$$

b) Let $\boldsymbol{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$. Find the weights $\boldsymbol{w}$ that minimize $||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2$ as a function of γ. Calculate $||\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}||_2^2$ and $||\boldsymbol{w}||_2^2$ as a function of γ for this value of $\boldsymbol{w}$. What happens to $||\boldsymbol{w}||_2^2$ as $\gamma \to 0$?

a.)

$$\underline{w}_{min} = \underline{V}\, \underline{\Sigma}^{-1}\, \underline{U}^T\, \underline{y}$$

$$\underset{2\times 1}{\underline{w}_{min}} = \underset{2\times 2}{\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}} \underset{2\times 2}{\begin{bmatrix} 1 & 0 \\ 0 & 1/\gamma \end{bmatrix}} \underset{2\times 4}{\frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix}} \underset{4\times 1}{\begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}}$$

$$= \frac{1}{2\sqrt{2}} \begin{bmatrix} 1 & 1/\gamma \\ 1 & -1/\gamma \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \underline{y}$$

$$\underline{w}_{min} = \frac{1}{2\sqrt{2}} \begin{bmatrix} 1+1/\gamma & 1-1/\gamma & 1-1/\gamma & 1+1/\gamma \\ 1-1/\gamma & 1+1/\gamma & 1+1/\gamma & 1-1/\gamma \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\underline{w}_{min} = \frac{1}{2\sqrt{2}} \begin{bmatrix} 1+1/\gamma+1+1/\gamma \\ 1-1/\gamma+1-1/\gamma \end{bmatrix} = \frac{1}{2\sqrt{2}} \begin{bmatrix} 2+2/\gamma \\ 2-2/\gamma \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1+1/\gamma \\ 1-1/\gamma \end{bmatrix}$$

b.)

$$||\underline{X}\,\underline{w} - \underline{y}||_2^2 = ||\underbrace{\underset{4\times 2}{\underline{U}\,\underline{\Sigma}\,\underline{V}^T}\cdot \underset{2\times 1}{\underline{w}_{min}}}_{4\times 1\,\checkmark} - \underset{4\times 1\,\checkmark}{\underline{y}}||_2^2 = \left|\left| \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right|\right|_2^2 = 0$$

Because $\underline{X}\,\underline{w}_{min} = \underline{y}$

c.)

$$||\underline{w}_{min}||_2^2 = \sqrt{\left[\frac{1}{\sqrt{2}}(1+1/\gamma)\right]^2 + \left[\frac{1}{\sqrt{2}}(1-1/\gamma)\right]^2}^{\,2}$$

$$= \sqrt{\tfrac{1}{2}(1+1/\gamma)^2 + \tfrac{1}{2}(1-1/\gamma)^2}^{\,2}$$

$$= \sqrt{\tfrac{1}{2}(1+2/\gamma+1/\gamma^2) + \tfrac{1}{2}(1-2/\gamma+1/\gamma^2)}^{\,2}$$

$$= \sqrt{\tfrac{1}{2}\left(1+\cancel{2/r}+1/r^2+1-\cancel{2/r}+1/r^2\right)}^{\,2}$$

$$= \sqrt{\tfrac{1}{2}\left(2+2/r^2\right)}^{\,2} = \sqrt{1+1/r^2}^{\,2} = 1+1/r^2 = \|w_{min}\|_2^2$$

1e.) $\underline{X} \approx \underline{T}\,\underline{W}$, $\underline{X} = \underline{U}\,\underline{\Sigma}\,\underline{V}^T$

To represent $\underline{T}\,\underline{W}$ in terms of SVD, need to write:

$\underline{T} = \underline{U}_r \underline{\Sigma}_r^{1/2}$ where r is the desired rank of approximation.

$\underline{W} = \underline{\Sigma}_r^{1/2} \underline{V}_r^T$, and $\underline{A}_r$ denotes the submatrix consisting of the first r rows or columns of $\underline{A}$.

Then, $\underset{5\times 7}{\underline{T}\,\underline{W}} = \underset{5\times r}{\underline{U}_r}\ \underset{r\times r}{\underline{\Sigma}_r}\ \underset{r\times 7}{\underline{V}_r^T}$

which is the definition of the rank $-r$ SVD of $\underline{X}$.

1g.) Jon

$$\underset{2\times 1}{\begin{array}{c} \text{ST} \\ \text{P\&P} \end{array}\begin{bmatrix} 6 \\ 4 \end{bmatrix}} = \underset{2\times 2}{\underline{T}}\ \underset{2\times 1}{\underline{W}} = \begin{bmatrix} | & | \\ \underline{t}_1 & \underline{t}_2 \\ | & | \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = w_1 \underline{t}_1 + w_2 \underline{t}_2$$

$$\rightarrow \begin{bmatrix} 6 \\ 4 \end{bmatrix} = w_1 \begin{bmatrix} t_{11} \\ t_{12} \end{bmatrix} + w_2 \begin{bmatrix} t_{21} \\ t_{22} \end{bmatrix}$$

LS Solution

$$\underline{w}_{min} = (\underline{T}^T \underline{T})^{-1} \underline{T}^T \underline{r}$$

$$\rightarrow\ 6 = w_1 \cdot t_{11} + w_2 \cdot t_{21}$$

$$4 = w_1 \cdot t_{12} + w_2 \cdot t_{22}$$

1h.)

$$\underset{5\times 1}{\begin{bmatrix} 6 \\ 4 \\ ? \\ ? \\ ? \end{bmatrix}} = \underset{5\times 2}{\underline{T}}\ \underset{2\times 1}{\underline{w}} = \begin{bmatrix} | & | \\ \underline{t}_1 & \underline{t}_2 \\ | & | \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 6 \\ 4 \\ ? \\ ? \\ ? \end{bmatrix} = \begin{bmatrix} | & | \\ \underline{t}_1 & \underline{t}_2 \\ | & | \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} t_{11} & t_{21} \\ t_{12} & t_{22} \\ t_{13} & t_{23} \\ t_{14} & t_{24} \\ t_{15} & t_{25} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

↓ See code pdf below ↓

CS/ECE/ME532 Period 11 Activity

Preambles

In [1]:
```
%matplotlib notebook
# 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:

In [8]:
```
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:

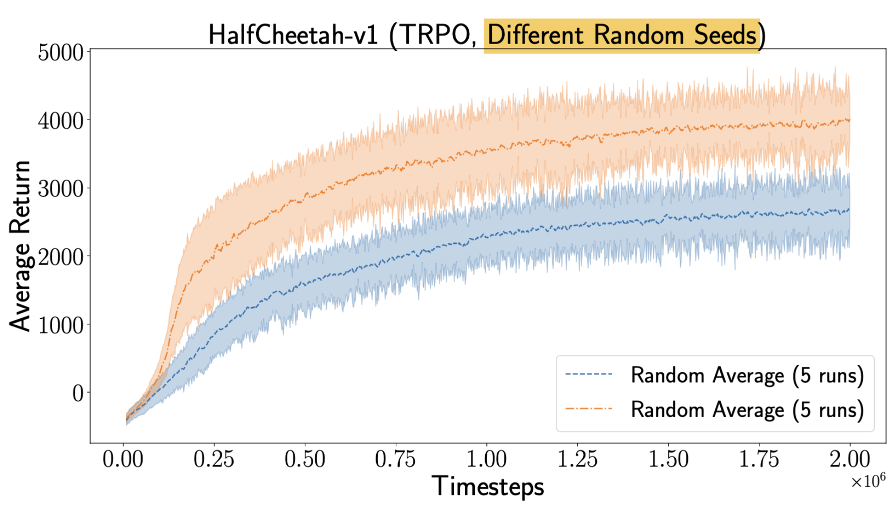


Figure 5: TRPO on HalfCheetah-v1 using the same hyperparameter configurations averaged over two sets of 5 different random seeds each. The average 2-sample t-test across entire training distribution resulted in $t = -9.0916$, $p = 0.0016$.

---

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
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

In [9]:
```
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.

In [11]:
```
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_tran
        # 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.

In [12]:
```
centroids_2, C_2 = kMeans(X, 2) ## Fill in the blank: call the "kMeans" algorithm with p
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 $\boldsymbol{X}$ based on this cluster as $\boldsymbol{T}\boldsymbol{W}^T$ where the columns of $\boldsymbol{T}$ contains the cluster centers and $\boldsymbol{W}$ is a vector of ones and zeros. Compare the rank-2 clustering approximation to the original matrix.

In [33]:

```
# Construct rank-2 approximation using cluster
W_transpose_2 = np.vstack([1-C_2,C_2])
centroids_transposed_2 = centroids_2.transpose()
X_hat_2 = centroids_2 @ W_transpose_2 ## Fill in the blank

print('Rank-2 Approximation = \n', X_hat_2)
np.linalg.matrix_rank(X_hat_2)
```

```
Rank-2 Approximation =
 [[3.00 7.33 3.00 7.33 7.33 3.00 3.00]
 [6.00 6.33 6.00 6.33 6.33 6.00 6.00]
 [3.00 7.00 3.00 7.00 7.00 3.00 3.00]
 [6.00 5.33 6.00 5.33 5.33 6.00 6.00]
 [2.75 8.00 2.75 8.00 8.00 2.75 2.75]]
```

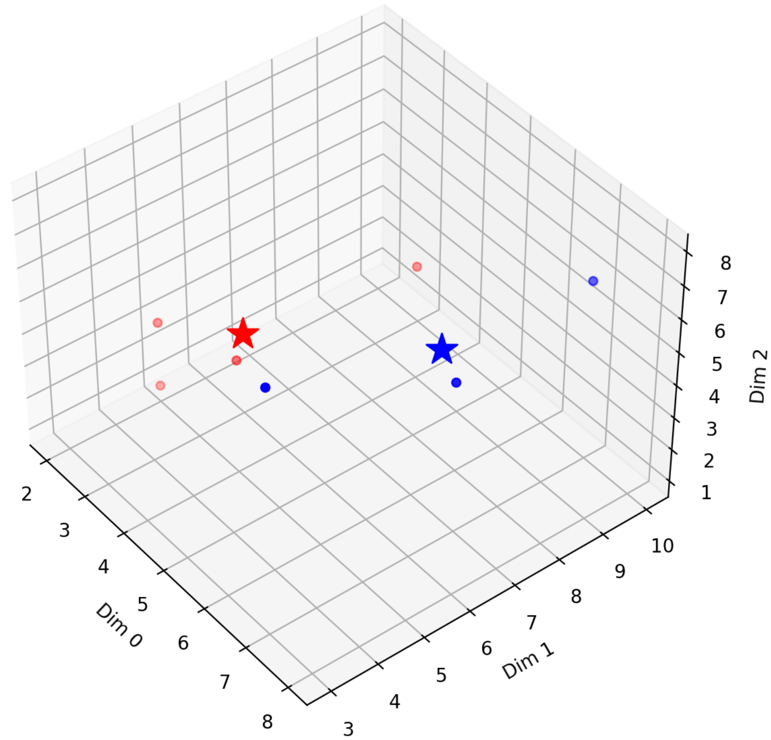
Out[33]:

```
2
```

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.

In [34]:

```
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])
```



Out[34]: Text(0.5, 0, 'Dim 2')

1 d) Repeat a)--c) with $K = 3$.

In [38]:
```
centroids_3, C_3 = kMeans(X,3) ## Fill in the blank

print('centroids = \n', centroids_3)
print('centroid assignment = \n', C_3)
```

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

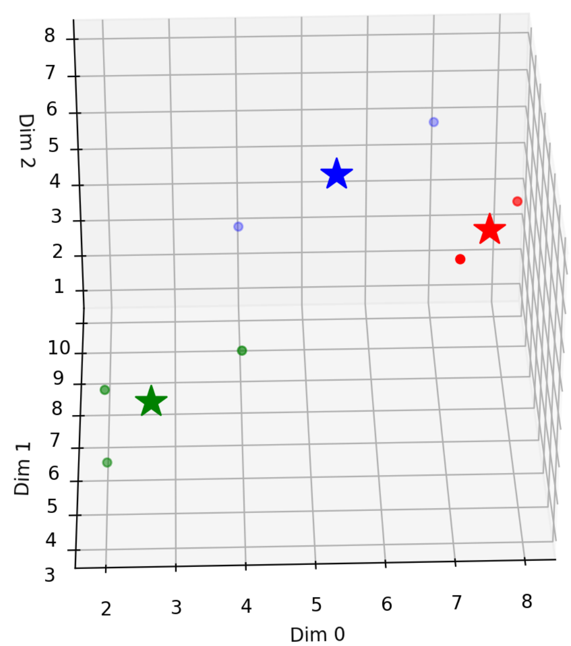
In [41]:
```
# Construct rank-3 approximation using cluster
centroids_transposed_3 = centroids_3.transpose()
W_transpose_3 = np.vstack([np.eye(max(C_3)+1)[:,i] for i in C_3]).T
X_hat_3 = centroids_3 @ W_transpose_3 ## Fill in the blank
print('Rank-3 Approximation = \n', X_hat_3)
np.linalg.matrix_rank(X_hat_3)
```

```
Rank-3 Approximation =
 [[5.50 7.50 2.67 7.50 5.50 2.67 2.67]
 [9.50 4.50 5.00 4.50 9.50 5.00 5.00]
 [5.00 7.50 2.67 7.50 5.00 2.67 2.67]
 [9.00 3.50 5.00 3.50 9.00 5.00 5.00]
 [5.50 8.50 2.33 8.50 5.50 2.33 2.33]]
```

Out[41]: 3

In [42]:
```
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])
```



Out[42]:
```
Text(0.5, 0, 'Dim 2')
```

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

Your answer goes here:

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

In [49]:
```
U, Sigma, VT = np.linalg.svd(X, full_matrices=True)
Sigma_matrix = np.zeros_like(X)
np.fill_diagonal(Sigma_matrix, Sigma)
```

In [70]:
```
# Problem 1f solution #

## Fill in the blank using U, S_matrix, and VT
T = U @ np.sqrt(Sigma_matrix)
W = np.sqrt(Sigma_matrix) @ 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)

print(f"T_2: \n{T_r}\n")
print(f"W_2: \n{W_r}\n")

X_2 = T_r @ W_r
print(f"Rank-2 approximation to X via SVD: \n{X_2}")
np.linalg.matrix_rank(X_2)

# Comment: The first taste vector captures a
# baseline taste profile. The corresponding affinity vector captures how
# high or low that user's baseline ratings are (e.g. if they tend to rate movies
# high or low, in general)

# The second taste vector captures a preference for sci-fi versus RomCom
# movies. The second corresponding affinity vector indicates how strong each user's
# tastes adhere to that taste profile.
```

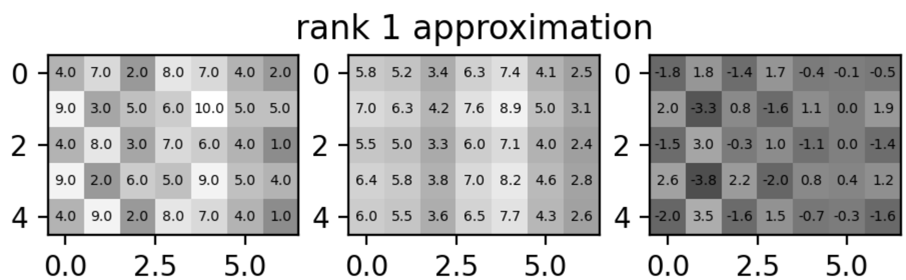
```
T_2:
[[2.40 1.02]
 [2.90 -1.50]
 [2.31 1.19]
 [2.68 -1.76]
 [2.50 1.55]]

W_2:
[[2.40 2.18 1.44 2.62 3.07 1.72 1.06]
 [-1.41 2.21 -0.92 1.08 -0.57 -0.13 -0.96]]

Rank-2 approximation to X via SVD:
[[4.34 7.49 2.51 7.39 6.81 4.01 1.57]
 [9.07 3.02 5.54 5.97 9.77 5.19 4.50]
 [3.87 7.66 2.22 7.32 6.42 3.82 1.30]
 [8.90 1.94 5.46 5.09 9.22 4.83 4.52]
 [3.83 8.88 2.17 8.23 6.82 4.12 1.16]]
```

Out[70]:
```
2
```

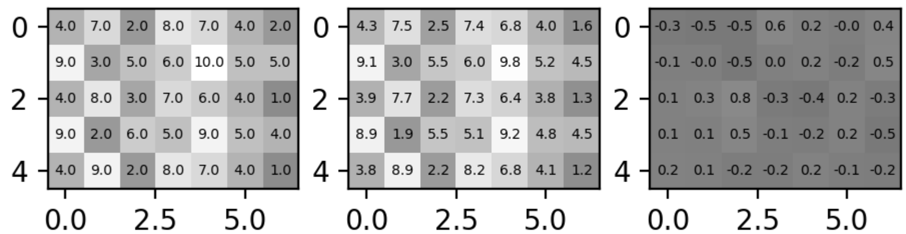
1 g) The following code visualizes the rank-r approximation for an increasing value of r. When does the approximation become exact? Why?

In [72]:
```
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):
```

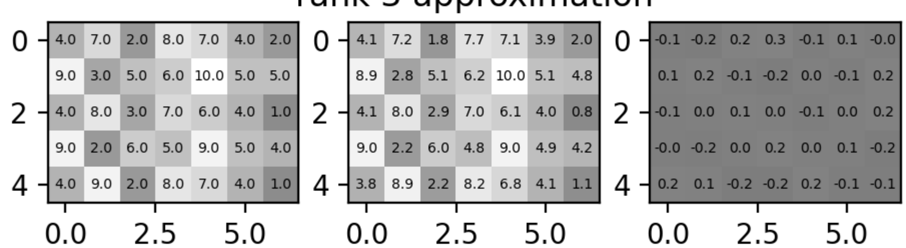
```
        ax[0].text(i,j,np.round(label,1),ha='center',va='center', size=5)
    im = ax[0].imshow(X, vmin=-10, vmax=10, interpolation='none', cmap='gray')
    for (j,i),label in np.ndenumerate(X_rank_r_approx):
        ax[1].text(i,j,np.round(label,1),ha='center',va='center', size=5)
    im = ax[1].imshow(X_rank_r_approx, vmin=-10, vmax=10, interpolation='none', cmap='gr
    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='

    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))

# Comment: The approxmation becomes exact at rank-5 because the original matrix and the
# appromxation are the same (also, 0 errors)
```



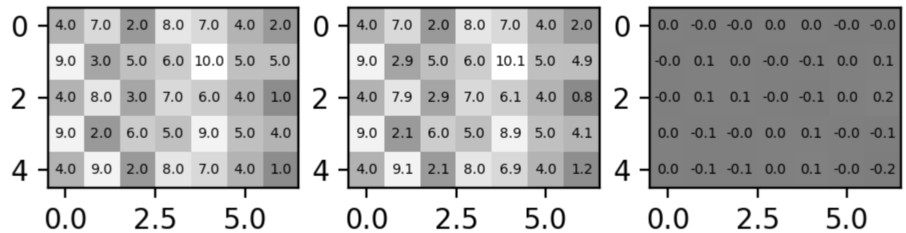
rank 2 approximation



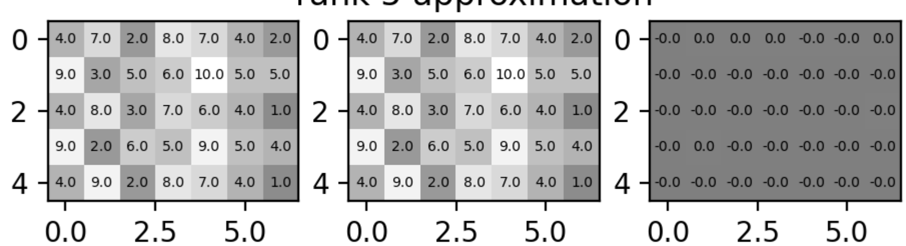
rank 3 approximation



rank 4 approximation



rank 5 approximation



```
ValueError                                Traceback (most recent call last)
Cell In[72], line 4
      2 T_r = T[:,0:r+1] ## Choose the first r columns of T
      3 W_r = W[0:r+1,:] ## Choose the first r rows of W
----> 4 X_rank_r_approx = T_r@W_r
      5 fig, ax = plt.subplots(1,3,figsize=(5.5, 5))
      6 for (j,i),label in np.ndenumerate(X):

ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0, with gufunc
 signature (n?,k),(k,m?)->(n?,m?) (size 5 is different from 6)
```

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.

Your answer goes here:

In [ ]:
```
# See submission pdf for handwritten answer
```

1 h) Using this weight vector, how can we predict Jon's ratings for all five movies, including the remaining three movies?

Your answer goes here:

In [ ]:
```
# See submission pdf for handwritten answer
```

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

In [127…
```
y = np.array([[6],[4]])

## Choice 1: K-means (K=2) based taste matrix T
T_kmeans_2, _ = kMeans(X,2)
T = T_kmeans_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)
```

```
[[6.00]
 [4.00]
 [5.98]
 [3.23]
 [6.74]]
```

In [186…
```
## Choice 2: K-means (K=3) based taste matrix T
T_kmeans_3, _ = kMeans(X,3)
T = T_kmeans_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)
```

```
[[6.00]
 [4.00]
 [6.00]
 [3.24]
 [6.74]]
```

In [210…
```
## Choice 3: SVD based taste matrix T
U, Sigma, _ = np.linalg.svd(X, full_matrices=False)
```

```python
Sigma_matrix = np.zeros_like(X)
np.fill_diagonal(Sigma_matrix, Sigma)
T = U[:,0:2] @ np.sqrt(Sigma_matrix[0:2]) # fill in the blank
T = T[:,0:2]
T_12 = T[0:2,0:2]
print(T@np.linalg.inv(T_12.T@T_12)@T_12.T@y)
```

```
[[6.00]
 [4.00]
 [6.01]
 [3.23]
 [6.83]]
```