CMSC 25300/35300, STAT 27700 (Fall 2023)
Homework 5

Submission Instructions:

- Please submit your homework in PDF to Gradescope (which can be accessed from the course's website on Canvas);
- Please paste your code in the submitted PDF. In other words, your submission should be a single PDF that contains both your writing solutions and your code. Be sure to include the output of your code when relevant (e.g. text and figures). Keep in mind that we are not easily able to run your code.
- Note that you do not need to copy the problem statements in your solution, *as long as you clearly indicate the problem numbers* (e.g., 1.a, 2.c, etc).

1. **The SVD and Least Squares**. Let $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ with $\text{rank}(\boldsymbol{X}) = k < \min\{n, p\}$. Each row represents one training sample, and each column represents one feature. Let $\boldsymbol{y} \in \mathbb{R}^n$ be the response vector for each sample. The full SVD of $\boldsymbol{X}$ is given by $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, where

$$\boldsymbol{U} = \begin{bmatrix} \boldsymbol{u}_1, \ldots, \boldsymbol{u}_k, \ldots, \boldsymbol{u}_n \end{bmatrix}, \qquad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\sigma}_1 & & & & \\ & \ddots & & & \\ & & \boldsymbol{\sigma}_k & & \\ & & & 0 & \\ & & & & \ddots \end{bmatrix}, \qquad \boldsymbol{V}^T = \begin{bmatrix} \boldsymbol{v}_1^T \\ \vdots \\ \boldsymbol{v}_k^T \\ \vdots \\ \boldsymbol{v}_p^T \end{bmatrix}.$$

   a. (5 points) Decompose $\boldsymbol{X}$ as the sum of rank 1 matrices. Then, give expressions for the SVDs of $\boldsymbol{X}^\top, \boldsymbol{X}\boldsymbol{X}^\top$ and $\boldsymbol{X}^\top\boldsymbol{X}$.

   b. Now we focus on the least squares problem, and try to find a weight vector $\boldsymbol{w}$ by minimizing $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2$.

      i) (5 points) Consider a simple case where $\boldsymbol{X} = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ and $\boldsymbol{y} = \begin{bmatrix} 6 \\ 2 \\ 1 \\ 2 \end{bmatrix}$. We would like to find $\boldsymbol{w}$ which minimizes the squared error, i.e. $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2$. However, we know that there is no such unique $\boldsymbol{w}$ because $\boldsymbol{X}$ does not have full column rank. Describe the set of $\boldsymbol{w}$ vectors which minimize the squared error. Which of these vectors has the smallest norm?

      ii) (4 points) Now we return to the general case, with $\boldsymbol{X}$ and $\boldsymbol{y}$ unknown, and with the SVD of $\boldsymbol{X}$ given by $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$. Find a basis for all weight vectors that satisfy $\boldsymbol{X}\boldsymbol{w} = 0$.
      (If $\boldsymbol{X}\boldsymbol{w} = 0$, then, for any $\hat{\boldsymbol{w}}$ such that $\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{\boldsymbol{w}}$, we have $\hat{\boldsymbol{y}} = \boldsymbol{X}(\hat{\boldsymbol{w}} + \boldsymbol{w})$.)

iii) (5 points) Let $\tilde{\boldsymbol{y}} = U^T\boldsymbol{y}$ and $\tilde{\boldsymbol{w}} = V^T\boldsymbol{w}$. Prove that $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|_2^2 = \|\tilde{\boldsymbol{y}} - \boldsymbol{\Sigma}\tilde{\boldsymbol{w}}\|_2^2$. Remember your solution from part (i), and give the set of $\tilde{\boldsymbol{w}}$ which minimizes the norm above. Which of these has the smallest norm?

iv) (4 points) Given the $\tilde{\boldsymbol{w}}$ you found above, what is the $\boldsymbol{w}$ with minimum norm which is a solution to the original least squares problem?

c. Least squares estimates can be used in many contexts. Consider a signal, $\boldsymbol{y}$, which is known to depend on a transformation matrix $\boldsymbol{X}$, an input vector $\boldsymbol{w}$, and some noise $\boldsymbol{e}$ as follows:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} + \boldsymbol{e}.$$

Suppose that we know $\boldsymbol{X}$, and observe $\boldsymbol{y}$, but we do not know $\boldsymbol{w}$. Then, we could use least-squares to estimate $\boldsymbol{w}$ from the known values $\boldsymbol{y}$ and $\boldsymbol{X}$. Use the starter code below to load the data, and then write code to answer the following questions.

```
import numpy as np
import matplotlib.pyplot as plt

# Load data
data = np.load("blurring.npz")
X = data['X']
y = data['y']

U, S, Vt = np.linalg.svd(X, full_matrices=False)

# Estimate w using X and y with regular least squares
# Your code here
w_LS

# Estimate w using X and y with truncated SVD
# Your code here
w_15 =
w_75 =
w_200 =

# Compare estimate to true value for LS

plt.plot(data['w'],label='true w')
plt.plot(w_LS, alpha=0.7,c='red',label='LS estimate')
plt.legend()
plt.show()

plt.plot(data['w'],label='true w')
plt.plot(w_15, alpha=0.7,c='red',label='k=15')
plt.plot(w_75, alpha=0.7,c='orange',label='k=75')
plt.plot(w_200, alpha=0.7,c='purple',label='k=200')
plt.legend()
plt.show()
```

i. (6 points) Use standard least squares to obtain an estimate of $\boldsymbol{w}$ and compare the

true $\boldsymbol{w}$ to your approximation using the plotting code above. What is the impact of the added noise to your estimate?

ii. (6 points) Use the truncated SVD to obtain three estimates of $\boldsymbol{w}$ (with $k = 15, 75, 200$) and compare the true $\boldsymbol{w}$ to your approximates. How does regularization using the SVD compare to least squares? How do the different values of k impact the approximation?

2. **PageRank**. Let us consider a web of n pages. Imagine we surf the web by randomly clicking on links at each web page when you arrive at it. If we click randomly in this way long enough, we'll reach a steady state where π_i is the probability that we're at page i at any given time. That is, you will end up landing on "important" pages more frequently, where "importance" reflects the number of other web pages linking to it, weighted by the importance of those pages. We wish to find a vector of probabilities $\boldsymbol{\pi} = \left[\pi_1, \ldots, \pi_n\right]^\top$ reflecting the steady state behavior.

   Define an $n \times n$ adjacency matrix M, corresponding to n pages as

$$M_{ij} = \begin{cases} 1, & \text{if page } j \text{ has a link to page } i \\ 0, & \text{otherwise} \end{cases}$$

   Then we define the "network adjacency matrix" A as the column normalized version of M

   The PageRank vector $\boldsymbol{\pi}$ satisfies

$$\boldsymbol{\pi} = \boldsymbol{A}\boldsymbol{\pi}.$$

   That is, $\boldsymbol{\pi}$ is an eigenvector of $\boldsymbol{A}$.

   Given a matrix A, we say a non-zero vector $\boldsymbol{v}$ is an *eigenvector* of $\boldsymbol{A}$ if $\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}$ for some scalar λ, which is called an *eigenvalue*. Eigenvectors have important connections to singular vectors. Consider a matrix and its singular value decomposition $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$. Then, if $\boldsymbol{A} = \boldsymbol{X}^\top\boldsymbol{X}$, we have $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Sigma}^2\boldsymbol{V}^\top$. Multiplying both size of this equation by $\boldsymbol{V}$ gives $\boldsymbol{A}\boldsymbol{V} = \boldsymbol{V}\boldsymbol{\Sigma}^2$, or $\boldsymbol{A}\boldsymbol{v}_i = \sigma_i^2\boldsymbol{v}_i$. That is, the right singular vectors of $\boldsymbol{X}$ are the eigenvectors of $\boldsymbol{A}$ and the singular values of $\boldsymbol{X}$ can be squared to give the eigenvalues of $\boldsymbol{A}$ corresponding eigenvalues of $\boldsymbol{A}$.

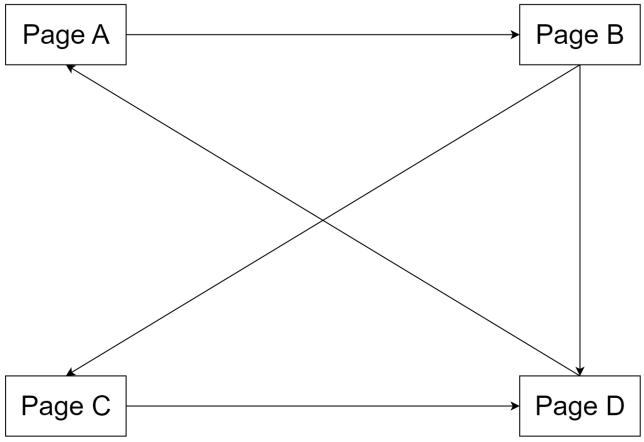
   When each column of $\boldsymbol{A}$ sums to 1, we can compute the PageRank vector $\boldsymbol{\pi} \in \mathbb{R}^n$ by repeatedly multiplying by $\boldsymbol{A}$ until convergence:

$$\boldsymbol{\pi}^{(1)} = A\boldsymbol{\pi}^{(0)},\ \boldsymbol{\pi}^{(2)} = A\boldsymbol{\pi}^{(1)},\ \ldots, \boldsymbol{\pi}^{(t)} = A\boldsymbol{\pi}^{(t-1)}$$

   The elements of the converged vector $\boldsymbol{\pi}$ are the PageRank of corresponding pages, and the matrix A is also called "Google matrix".

   Note that if we can find the converged page rank vector $\boldsymbol{\pi}$, $\boldsymbol{\pi}$ must be an eigenvector of the matrix A with eigenvalue one.

a. (5 points) Consider the following network:



What is $\boldsymbol{M}$ for this network? $\boldsymbol{A}$?

b. (8 points) Using this matrix $\boldsymbol{A}$, what is the PageRank of each of the 4 pages? Write a Python script to compute this.

c. (8 points) In real application, Google found it useful to add a "damping factor" α such that $0 < \alpha < 1$. The modified "Google matrix" G is then defined to be

$$G = \alpha A + (1 - \alpha)\boldsymbol{u}\mathbb{1}^T$$

where $\mathbb{1} \in \mathbb{R}^n$ is all-ones vector and $\boldsymbol{u}$ models the likelihood of users clicking a page (whether or not there is a link to it on current page). Theoretically, we could use any vector $\boldsymbol{u} \in \mathbb{R}^n$, and in practice Google used $\boldsymbol{u} = \mathbb{1}$, so that

$$G = \alpha A + (1 - \alpha)\frac{1}{n}\mathbb{1}\mathbb{1}^T.$$

Intuitively, you can think of the damped PageRank as follows: a user who clicks a random link on the current page with probability α, but with probability $1 - \alpha$ the user would choose a uniformly random page (whether or not there is a link to it).
For $\alpha = 0.8$, what is the PageRank of each of the 4 pages? Write a Python script to compute this. (HINT: If you have a matrix `G` and want to normalize it so each column sums to 1, use the line `G = G/G.sum(axis=0)`.)

d. (9 points) Consider a world of n pages, in which every page (including Youtube and Wikipedia) links to Youtube and Wikipedia, and no page links anywhere else. Youtube is page 1, Wikipedia is page 2, and the other $n-2$ pages are page 3 to n. With damping parameter α, what is the adjacency matrix M in this case? And the Google matrix G? If the PageRank vector $\boldsymbol{\pi}$ is a probability vector in $\mathbb{R}^n$ (elements sum up to 1), what is the PageRank of Youtube and Wikipedia? What about the rank of the rest $n-2$ pages? Express these two ranks in terms of damping parameter α and total number of pages n. (HINT: let $\boldsymbol{\pi} = \begin{bmatrix} x & x & y & y & \cdots & y \end{bmatrix}^\top$ and find an expression for x and y.

3. **SVD for image compression.** In this problem, you will learn how SVD could be used in image compression.

a. (5 pts) Compute the singular values of a pre-downloaded image (via the code provided below) and plot them. Remember to use a logarithmic scale.

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from PIL import Image

# feel free to use your favorite picture
# or work with the provided one
img = Image.open("disaster-girl.jpg", mode="r")
img = np.array(img).astype(np.int32)

plt.imshow(img)
plt.title('Original Picture')
plt.show()

# YOUR CODE BELOW
# note that images usually have 3 channels
# so, you can reshape it
# img_stack = face.reshape((img.shape[0], -1))
# and find its SVD.
# another option is to find the SVD of every color channel
```

b. (15 pts) Complete a function `compress` that performs SVD and truncates it (using k singular values/vectors). See the prototype below. Note that in the colorful case, you must split your image into channels and work with matrices corresponding to different channels separately. Plot a reconstructed image $\hat{\boldsymbol{X}}$ of your original image $\boldsymbol{X}$ such that $\text{rank}(\hat{\boldsymbol{X}}) = 5, 20, 50$ using, for example, `plt.subplots`.

```
def compress(image, k):
    """
    Perform svd decomposition and truncate it (using k singular
       ↪ values/vectors)

    Parameters:
        image (np.array):        input image (probably, colourful)

        k (int):                 approximation rank (number of
           ↪ singular values)

    --------
    Returns:
      reconst_matrix (np.array): reconstructed matrix (tensor in
         ↪ colourful case)

      s (np.array):              array of singular values
    """
#     YOUR CODE IS HERE
    return reconst_matrix, s
```

c. (15 pts) Plot the following figures (feel free to work only with one channel of the image, but working with all three channels shouldn't be more difficult):

1) How does the relative error $\left(\frac{\|\boldsymbol{X}-\hat{\boldsymbol{X}}\|_2}{\|\boldsymbol{X}\|_2}\right)$ of approximation depend on the rank of approximation k?

2) How does compression rate depend on the rank of approximation in terms of storing information? (($\boldsymbol{U}$, $\boldsymbol{V}$ sizes + k singular values to store) /total size of the original image)

Share your observations from both figures. For example, what approximation rank (or number of the singular values k) would you choose to compress this image?