CS168 Final Project

June 7, 2023

1 Miniproject 5: Exploring SVD

Eva Batelaan (batelaan@stanford.edu), Jacopo Mascitelli (jmasci@stanford.edu), Jasmine Bilir (jbilir@stanford,edu), Si Yi Ma (siyima00@stanford.edu

This project is meant to serve as a possible alternative to the existing mini-porject on SVD. We have included three questions that are meant to consider conceptual and applied understanding of SVD.

Reflection Paragraph: We feel that the content we produced in this mini-project compliments the content covered in week 4/5 and the existing mini-project 5 because we reinforce the fundamentals of singular value composition. Particularly, in problem 1 we explore the matrix relationships and work to help the student understand the fundamental workings of this approximation. Likewise, in problem 3, we consider the differences in matrix rank and how differences in low-rank approximations relate to approximation quality. Additionally, problems 2 and 3 help underscore the motivations behind low-rank approximations and the decisive utility of SVD. Problem 2 uses recommendation techniques and considers how SVD can be applied to more complex and complicated questions (in some ways, we chose this problem as a way to reveal/expand on conceptual content covered in the "SVD for learning word embeddings" problem of the existing mini-project 5. Problem 3 shows a different application using images and image compression (thereby relating to problem 2 of the existing PSET), but goes further by exploring how this knowledge could inform basic types of image clasification.

For problem 2, use the following datafile: https://drive.google.com/file/d/1cTiqm3373iFAKL-oS1HTeDMXvejQukIU/view?usp=sharing

2 Problem 1: SVD and Pseudoinverse

As a primer into the mini-project we will explore the concepts of SVD and its utility for representing systems of equations. This understanding will help us for the application basis of the subsequent problems.

When the coefficient matrix A is square and nonsingular, the solution to the linear system of equations $A\vec{x} = \vec{b}$ can be expressed as:

$$\vec{x} = V\Sigma^{-1}U^T\vec{b}$$

where $A = U\Sigma V^T$ si the SVD of A, U and V are orthogonal matrices, Σ is a diagonal matrix with singular values of A.

2.1 Part A

Solve the linear system of equations using SVD

$$4x + 4y = 8$$

$$2x - y = 1$$

Example Solution:

[ ]:
```
import numpy as np

def solve_linear_system_svd(A, b):
    U, s, VT = np.linalg.svd(A)  # Perform SVD on coefficient matrix A
    x = VT.T @ np.linalg.inv(np.diag(s)) @ U.T @ b

    return x
```

[ ]:
```
A1 = np.array([[4, 4], [2, -1]])
b1 = np.array([8, 1])
solution1 = solve_linear_system_svd(A1, b1)
print(solution1)
```

```
[1. 1.]
```

2.2 Part B

Now, use the function implemented in part(a), solve the linear system of equation using SVD

$$2x + 3y + z = 5$$

$$4x + 5y + 2z = 7$$

What is the output of the function? Explain why you get the answer and the condition of the rank of the matrix.

Example Solution:

[ ]:
```
A = np.array([[3, 4], [6, 8], [9, 12]])
b = np.array([5, 10, 15])

solution2 = solve_linear_system_svd(A, b)
print(solution2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[15], line 4
      1 A = np.array([[3, 4], [6, 8], [9, 12]])
      2 b = np.array([5, 10, 15])
----> 4 solution2 = solve_linear_system_svd(A, b)
      5 print(solution2)
```

```
Cell In[13], line 5, in solve_linear_system_svd(A, b)
      3 def solve_linear_system_svd(A, b):
      4     U, s, VT = np.linalg.svd(A)  # Perform SVD on coefficient matrix A
----> 5     x = VT.T @ np.linalg.inv(np.diag(s)) @ U.T @ b
      7     return x

ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0, with gufunc signature (n?,k),(k,m?)->(n?,m?) (size 3 is different from 2)
```

I got an error message due to the dimension of s. The normal inverse is only defined for square matrices.If the coefficient matrix A is not square (having more columns than rows or more rows than columns), it does not have an inverse. The matrix inverse is closely tied to the concept of linear independence and full rank. In order for a square matrix to have an inverse, it must be full rank, meaning that its columns (or rows) are linearly independent. This ensures that the system of linear equations has a unique solution.

2.3 Part C

When A is nonsquare or singular, the linear system $A\vec{x} = \vec{b}$ may not have a unique solution. In this case, it is common to use the pseudoinverse, denoted as A^+, to obtain a solution. The pseudoinverse is a generalization of the inverse that can be applied to any matrixc, regardless of its shape and singularity. The solution to the linear system is then given by $\vec{x} = V\Sigma^+U^T\vec{b}$ where

$$\Sigma^+_{ij} = \begin{cases} \frac{1}{\Sigma^+_{ij}} & \text{if } \Sigma^+_{ij} \neq 0 \\ 0 & \text{if } \Sigma^+_{ij} = 0 \end{cases} \tag{1}$$

The matrix $A^+ = V\Sigma^+U^T$ is defined as the Moore-Penrose inverse of a matrix. Given the linear system equations from part(b), find the solution using pseudoinverse without using numpy.

Example solution:

[ ]:
```
import numpy as np

def pseudoinverse(A):
    U, s, Vt = np.linalg.svd(A)
    m, n = A.shape

    # Calculate the reciprocal of nonzero singular values
    s_inv = np.zeros((n, m))
    s_inv[:n, :n] = np.diag([1/s_val if s_val != 0 else 0 for s_val in s])

    # Calculate the pseudoinverse
    A_plus = Vt.T @ s_inv @ U.T

    return A_plus

def solve_linear_system_pseudoinverse(A, b):
```

```
    A_plus = pseudoinverse(A)
    x = A_plus @ b
    return x

A = np.array([[3, 4], [6, 8], [9, 12]])
b = np.array([5, 10, 15])
solution = solve_linear_system_pseudoinverse(A, b)
print(solution)
```

```
[0.71875 1.25   ]
```

3 Problem 2: SVD Based Movie Recommendations

In this problem, we will use the concept of SVD to consider the application of movie recommendations. We can therefore learn more about how to use SVD as a tool for investigating the likeness of data.

3.1 Part A

Imagine you have a small online movie streaming service. Your company has 10 movies available, and you have 5 registered users. The users have rated some of the movies on a scale of 1-5. You have the ratings data organized in a 5x10 matrix R, where each row corresponds to a user, each column corresponds to a movie, and each entry represents a user's rating for a movie. An entry of 0 means the user has not yet rated the movie.

Write down a random 5x10 matrix R, where each entry is either an integer from 1 to 5 or 0

Example Answer R = [[4, 0, 3, 5, 0, 1, 0, 2, 0, 4], [0, 5, 0, 3, 4, 0, 2, 0, 1, 3], [5, 3, 0, 0, 4, 0, 1, 3, 2, 0], [0, 4, 2, 0, 0, 3, 0, 1, 5, 3], [3, 0, 1, 4, 2, 0, 0, 5, 0, 0]]

3.2 Part B

Our current approach for recommending movies consists in iterating through each user's unrated movies and suggesting they watch the one with the highest average rating (as computed from the scores of all other users). Discuss the limitations of this recommendation approach. How might these limitations affect the user experience on the platform?

Example Answer This approach is very simplistic. It only considers the average rating for all users and doesn't consider the specific preferences of our interested user. Even though movies with a globaly wide appeal are trivially likely to also appeal most of the users on our platform, this approach may still recommend movies that certain users do not like, leading to a poor user experience, and would risk alientating more nieche video productions from our streaming service as they would never get reccomended.

3.3 Part C

Knowing this, you start brainstorming different approaches that could work better for the problem at hand. Briefly describe how an SVD-based recommendation system could improve upon your current approach and would it handle unrated movies?

You also want to consider potential downsides for the system. Despite the advantages of SVD, under what conditions might your original recommendation approach be more suitable? Consider factors like the number of users, number of movies, and sparsity of the ratings matrix.

Example Answer An SVD-based recommendation system would create a latent factor model from the user-movie ratings matrix. It would predict the missing ratings by filling in the 0s in the matrix with these predictions, providing a better estimate of how much the user might like unrated movies. For instance, if two users have similar patterns of ratings across a set of movies (say they both like action movies and dislike romantic movies), they are considered to have similar tastes. So, if one user liked a particular action movie, that movie would likely be recommended to the other user. Therefore, SVD implicitly gives more weight to users with similar preferences when generating recommendations.

The original recommendation approach might be more suitable when the ratings matrix is extremely sparse (i.e very few movies have been rated by users) or when the number of users or movies is very small. The SVD approach is more computationally intensive and may not provide meaningful results when there is very little ratings data to work with. Because of this you might want to wait until your service scales up before performing the upgrade.

3.4 Part D

Seeing the potential benefits of this approach you start thinking about how to implement the new system. Explain in simple terms how SVD would work in the context of your movie recommendation system?

Example Answer SVD works by decomposing the user-movie ratings matrix into three separate matrices: a matrix of user preferences, a diagonal matrix of singular values (which can be thought of as weights), and a matrix of movie attributes. The matrix of user preferences and the matrix of movie attributes provide a compressed representation of the users' tastes and the movies' characteristics, respectively.

3.5 Part E

Let's now implement a simplified version of this system: Fill in the missing movie ratings with the mean rating of each movie. Let's call this matrix then perform Singular Value Decomposition on it and reduce its dimensionality. Use a rank of k = 2. Then reconstruct the matrix from the decomposed matrices and, for each user, recommend the movie with the highest estimated rating in R_hat that the user has not rated yet.

You may use external libraries.

[ ]:
```python
# Sample python pseudo code
import numpy as np
from scipy.sparse.linalg import svds

# a. Fill missing ratings with mean movie rating
R_filled = np.where(R == 0, np.mean(R, axis=0), R)

# b. Compute SVD
```

```
U, sigma, Vt = svds(R_filled, k=2)

# c. Reconstruct matrix
R_hat = np.dot(U, np.dot(np.diag(sigma), Vt))

# d. Recommend movie with highest estimated rating not yet rated
recommendations = np.where(R == 0, R_hat, 0)
recommendation_indices = np.argmax(recommendations, axis=1)
```

4 Problem 3: Image Classification with SVD

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.linalg import svd, norm
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
import h5py
import os
from PIL import Image
```

Note to TAs *the general flow of this problem follows that of* How to Use Singular Value Decomposition (SVD) for Image Classification in Python

We used this blogpost for inspiration but made sure that our problems were unqiue, and incorporated class material.

We see how we can use SVD in order to capture prominent characteristics in images. Our images will be represented by matrices of 1s and 0s, representing the black and white in the images. We will use the Handwritten Digits USPS dataset, a commonly used datset for image recognition tasks in CV (https://www.kaggle.com/datasets/bistaumanga/usps-dataset?select=usps.h5). Using compression, we will see which characteristics are identifiably associated with specific digits and use our understanding of approximations.

4.1 Part A:

Download and successfully upload the USPS Handwritten Digits dataset. Load its contents into a Pandas dataframe. (Hint: The dataset is given in hdf5 file format. Use the code example provided on the kaggle webpage to understand how to read this dataset. https://www.kaggle.com/datasets/bistaumanga/usps-dataset?select=usps.h5)

[ ]:
```
# Based off code in the article
with h5py.File(os.path.join(os.getcwd(), 'usps.h5'), 'r') as hf:
        train = hf.get('train')
        test = hf.get('test')
        x_train = pd.DataFrame(train.get('data')[:]).T
        y_train = pd.DataFrame(train.get('target')[:]).T
        x_test = pd.DataFrame(test.get('data')[:]).T
```

```
        y_test = pd.DataFrame(test.get('target')[:]).T
```

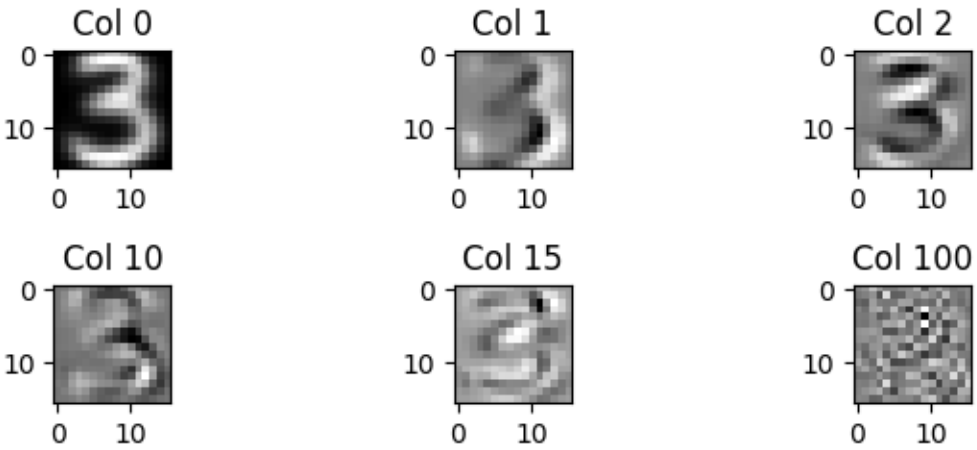
4.2 Part B:

Pick any single number from the USPS dataset and compute the SVD composition. Using plt.imageshow, plot the image representation of the following columns in the left-singular matrix: {0, 1, 2, 10, 15, 100}. Explain why the plots look as they do. Can you think about a justification for why the images change as we pick larger indices?

Example Answer

[ ]:
```
# Based off code in the article
threes = x_train.loc[:,list(y_train.loc[0,:]==3)]
u, s, v_t = svd(threes, full_matrices=True)
cols = [0, 1, 2, 10, 15, 100]

plt.figure(layout='tight')
for i in range(len(cols)):
  plt.subplot(10//3 + 1, 3, i + 1)
  plt.imshow(u[:,cols[i]].reshape(16,16),cmap='binary')
  plt.title("Col " +str(cols[i]))
```



Comprehensive Justification (Note that student response may not need to be as thourough):

The matrix U in SVD represents the left singular vectors of the original matrix A. It consists of a set of orthonormal column vectors that form a basis for the column space of A. These vectors span the range of A and provide information about the directions of maximum variance in the input data. The elements within each column of U represent the coefficients or weights that define how the corresponding left singular vector contributes to the original matrix A when it is reconstructed using the SVD components. The significance of the columns of U is determined by the corresponding singular values in the diagonal matrix Σ. The singular values represent the scaling factors applied to the left and right singular vectors during the matrix transformation. In Σ, the singular values are

arranged in descending order along the diagonal, meaning that the left-most columns will correspond to the dominant modes of variation that explain the variability in the data.

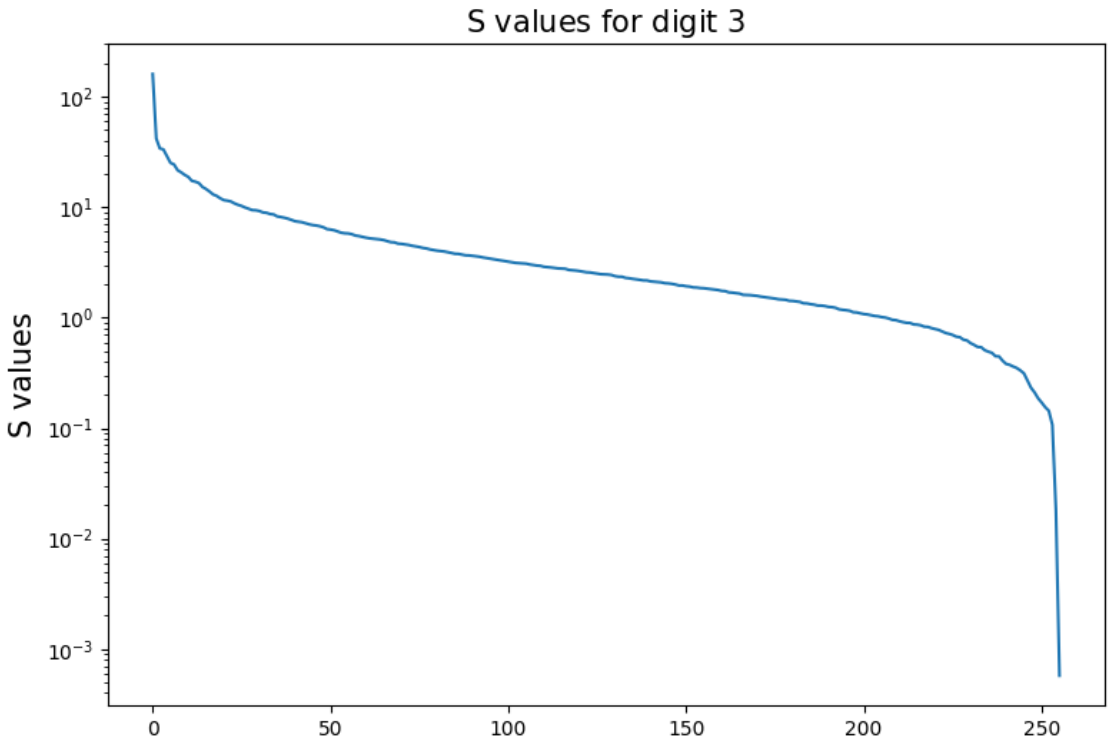
In the plots, we can see that column zero contains the most dominant information, as it is the most resembling the digit 3. As we scale and select columns to the right, we see that the images have less and less ressemblance to the digit 3 as they represent less dominant characteristics.

4.3 Part C:

Again for the digit 3, plot the values in the singular matrix S and explain how these values correspond to and explain the relationship seen in part (b). Explain why this information is useful when we make rank k approximations.

Example Answer

[ ]:
```
# Based off code in the article
plt.figure(figsize = (9, 6))
plt.plot(s)
plt.title('S values for digit $3$',fontsize=15)
plt.ylabel('S values' ,fontsize=15)
plt.yscale("log")
plt.show()
```



Comprehensive Justification (Note that student response may not need to be as thourough):

As shown in the graph, values along the diagonal in the left-most columns have greater values than the right-most columns. Particularly, their weights plateu before decreasing in magnitude for the right-most columns of the graph. Therefore, when we do a rank K approximations, we use the first K values, representing the columns of U with the greatest variance which are weighted as significant information by S. Therefore, when we clip any non-K values, we still maintain a significant portion of the data, and smaller amounts of information is lost relative to what came before.

4.4 Part D:

Using informations from the plots generated above, provide a general description of how well/what would be captured for the following rank-k approximations of the digit 3 images. How different would each image be to each other? k = {50, 200, 250}

Example Answer The rank 50 approximation would be the poorest quality and may maintain a hazy white outline of the three, but the background would likely be all black. The image would generally lack detail.

The rank 200 approximation would contain the vast majority of the information needed for a strong approximation. It would contain more shades of gray and contain more detail for the three and its writing. As seen from the value graph, this is the end of the plateau and indicates that it contains the majority of the heavily weighted information.

The rank 250 approximation would contain slightly more detail than the 200 approximation, but for the most part the two images will look similar. The information included in the 250 approximation is less heavily weighted by S and is therefore less significant.

4.5 Part E:

Now, we're going to test how well lower rank approximations of the left singular matrix can approximate the basis. For $k = [10, 15, 50, 200, 256]$, use the rank-k approximation of the left singular matrix to predict the labels of the images in the test set. Report the accuracy for each k.

Specifically, compute how well a digit from the test set can be represented in the 10 different orthonormal bases and then choose the basis with the smallest residual (according to least squares error). You can use that $\|(I - U_k U_k^\top)z|$ is the residual norm vector for a vector z and a rank-k approximation U_k. To do so, you will first need to compute SVD for 0-9 as you did for 3 in part (b). Explain how your results from part (c) and (d) and your knowledge of SVD compare with your results.

Example Answer

[ ]:
```
num_to_u = {}
for i in range(10):
  num = x_train.loc[:,list(y_train.loc[0,:]==i)]
  u, s, v_t = svd(num, full_matrices=True)
  num_to_u[i] = u
```

[ ]:
```
# Based off code in the article
x_d, x_n = x_test.shape
y_d, y_n = y_test.shape
```

```
I = np.identity(x_d)
ks = [10, 15, 50, 200, 256]

predictions=np.empty((y_n,0), dtype = int)

for k in ks:
    prediction = []
    for i in range(x_n):
        residuals = []
        for j in range(10):
            u_k = num_to_u[j][:,0:k]
            res = norm(np.dot(I - np.dot(u_k,u_k.T), x_test[i]))
            residuals.append(res)
        index_min = np.argmin(residuals)
        prediction.append(index_min)
    prediction = np.array(prediction)
    predictions = np.hstack((predictions, prediction.reshape(-1,1)))
```

[ ]:
```
# Based off code in the article
scores=[]
for i in range(len(ks)):
    score = np.count_nonzero(y_test.loc[0,:] == predictions[:,i]) / y_n
    print("Number of basis vectors:", ks[i], ", Accuracy:", score)
    scores.append(score)
```

```
Number of basis vectors: 10 , Accuracy: 0.9347284504235177
Number of basis vectors: 15 , Accuracy: 0.9441953163926258
Number of basis vectors: 50 , Accuracy: 0.9277528649725959
Number of basis vectors: 200 , Accuracy: 0.7483806676631789
Number of basis vectors: 256 , Accuracy: 0.5999003487792726
```

Example Justification:

As k increases, the accuraccy first increases and then decreases.

This aligns with our results in part (b), which show that the the majority of the image structure is captured in the first few columns of the singular matrix. Similarly, our results from part (c) showed that the s values for the lower indexed columns are much larger than those in the higher vauled columns, indicating they're more important in approximating the key features of the images.

Thus, as k increases, we're capturing more of the noise that was specific to the training set, making our results less generalizable.

4.6 Part F:

Select the k with the best performance, and show 12 examples of predictions, 6 correctly classified and 6 incorrrectly classified. Comment on your results.

[ ]:
```
# Based off code in the article
incorrect = np.where(y_test.loc[0,:] != predictions[:,1])
```

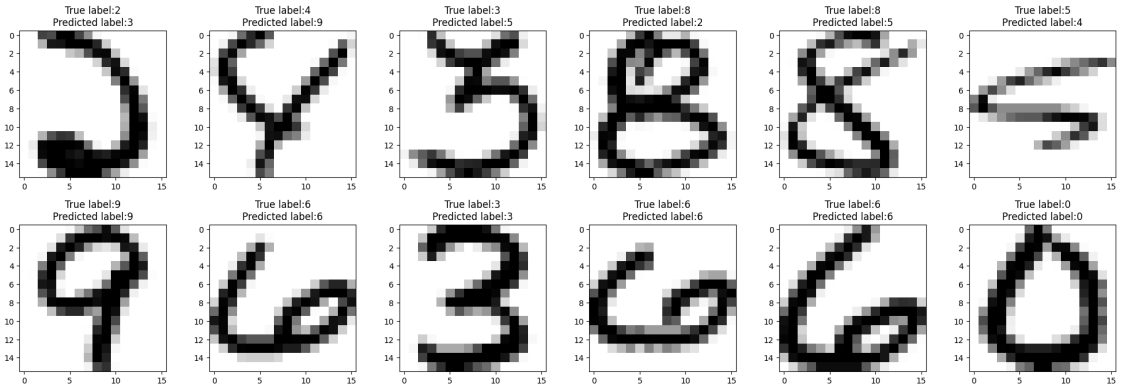
```
correct = np.where(y_test.loc[0,:] == predictions[:,1])

plt.figure(figsize=(20,10), layout='tight')
columns = 6
for i in range(0,6):
    incorrect_id = incorrect[0][i]
    incorrect_image = x_test[incorrect_id].to_numpy().reshape(16,16)

    plt.subplot(int(12 / columns + 1), columns, i + 1)
    plt.imshow(incorrect_image, cmap='binary')
    plt.title("True label:"+str(y_test.loc[0,incorrect_id]) + '\n'+ "Predicted␣
 ↪label:"+str(predictions[incorrect_id,1]))

for i in range(0,6):
    correct_id = correct[0][i]
    correct_image = x_test[correct_id].to_numpy().reshape(16,16)

    plt.subplot(int(12 / columns + 1), columns, 6 + i + 1)
    plt.imshow(correct_image, cmap='binary')
    plt.title("True label:"+str(y_test.loc[0,correct_id]) + '\n'+ "Predicted␣
 ↪label:"+str(predictions[correct_id,1]))
```



Example Commentary:

The SVD approximation seems to correctly classify well-written images. Meanwhile, images with sloppier handwriting seem to be more difficult. From this subsample, we can see that the SVD approximation particularly struggles with 8s.