Scientific Computing for Data Analysis
Classification of Handwritten Digits
Mini Project 2

Md Masudul Islam, Hugo Bergendahl Hansson, Mhd Rateb Almajni

February 25, 2024

1 Introduction

Pattern recognition is a crucial technology in numerous applications. For instance, its use in the automated interpretation of postal codes significantly streamlines mail sorting processes. This report focuses on the application of pattern recognition for the identification of handwritten numerical digits, employing a methodology based on Singular Value Decomposition (SVD).

Our objective is to demonstrate the application of SVD in distinguishing and classifying images of handwritten digits ranging from 0 to 9. Utilizing a set of training images, we aim to predict the numeral represented in each image from a separate test dataset. The effectiveness of this method is gauged by the accuracy of these predictions. The report elaborates on the application of SVD in image classification and analyzes the success rate presented in figure 3 of this technique in recognizing handwritten digits.

2 Method

The first step in our approach is the training process. Our training dataset comprises a matrix, where each column is considered to be a gray-scale image of a digit (28x28 pixels), transformed into a 784-element vector. Notably, the number of columns substantially exceeds the number of rows in this matrix. Each column of this matrix is matched with a corresponding label, provided in a separate array. To effectively apply Singular Value Decomposition (SVD) to these digit images, the initial task is to segregate the larger matrix into ten smaller matrices, each representing a different digit from 0 to 9. In an effort to optimize computational efficiency, only a select subset of the images for each digit is utilized in the training phase. Next, we need to perform Singular Value Decomposition (SVD) on each of these ten matrices, each corresponding to a specific digit.

2.1 Singular Value Decomposition

Focusing on a particular matrix, referred to as A, which has dimensions $m \times n$, the process of Singular Value Decomposition is applied. The SVD of A is expressed as $A = U\Sigma V^T$. Here, U and V are orthogonal matrices, and Σ is a diagonal matrix of singular values. The SVD decomposes matrix A into these components, enabling the extraction of significant features from the training images. The matrices U and Σ are of particular interest, as the columns of U (the left singular vectors) form a basis for A's column space, capturing the essential characteristics of the images. A k-rank approximation of A is obtained using the first k singular vectors, providing a simplified representation that retains key features of the original matrix.

2.2 Classification via Residuals of Least Squares Problems

Post-training, we obtain ten distinct k-rank approximations, corresponding to each digit. This results in ten sets of basis vectors, with each set uniquely representing the predominant features of a specific digit.

In the classification phase, we apply these basis sets to the test data, which comprises images similar to the training set. For each test image, we approximate it using all ten basis sets, leading to ten distinct least squares problems. The classification of a test image is determined by the basis set that yields the smallest residual in these least squares problems. In essence, each test image is classified as the digit corresponding to the basis set with the minimal residual.

The residual calculation in this SVD-based classification approach employs the Euclidean norm of the residual vector. The residual for a test image d is computed using the formula: residual = $||I - U_k U_k^T||d$, where I is the identity matrix, U_k represents the first k columns of U, and U_k^T is the transpose of U_k.

2.3 Implementation in Python

The full code implementation detailed in Appendix A starts with importing essential Python libraries: Numpy for numerical computations, and matplotlib for data visualization. The training and test datasets, along with their corresponding labels, are loaded using Numpy's load function.

The training data is then restructured into matrices for each digit, with each matrix comprising columns that represent the gray-scale images of a specific digit, flattened into 784-element vectors. Then, Singular Value Decomposition (SVD) is applied to these matrices, extracting essential features and patterns. This process is implemented as follows:

```
# Create training matrices for each digit and compute SVD
training_matrices = {digit: train_digits[:, (train_labels == digit).flatten()][:, :num_cols]
                     for digit in range(num_digits)}
svd_components = {digit: np.linalg.svd(matrix, full_matrices=False)
                  for digit, matrix in training_matrices.items()}
```

For visualization, the singular values and images of selected digits are plotted using the following code:

```
# Function to plot singular values for selected digits
def plot_singular_values(digit, color, label):
    singular_values = svd_components[digit][1] # Extract singular values
   plt.plot(singular_values, marker='o', linestyle='--', color=color, label=label)
    ...
plt.show()

# Function to plot singular images for a digit
def plot_singular_images(digit, start_subplot):
    ...
    plt.imshow(singular_image, cmap='gray')
    ...
plt.show()
```

In the final stage, the code assesses classification accuracy by computing residuals to measure the discrepancy between test images and their SVD approximations. The classification for each test image is determined based on the smallest residual. This evaluation process is illustrated in the following code snippet:

```
# Compute classification accuracy for each k
accuracy_k = []
```

```
for k in k_range:
    ...
    # Calculate accuracy as the percentage of correct predictions
    accuracy = np.mean(test_labels[0] == predictions) * 100
    accuracy_k.append(accuracy)

# Plot accuracy vs k values
plt.plot(k_range, accuracy_k, marker='o')
...
plt.show()
```

This segment calculates and visualizes the accuracy of digit classification across varying values of k, highlighting the effectiveness of the SVD-based method.

3 Results

3.1 Singular Values

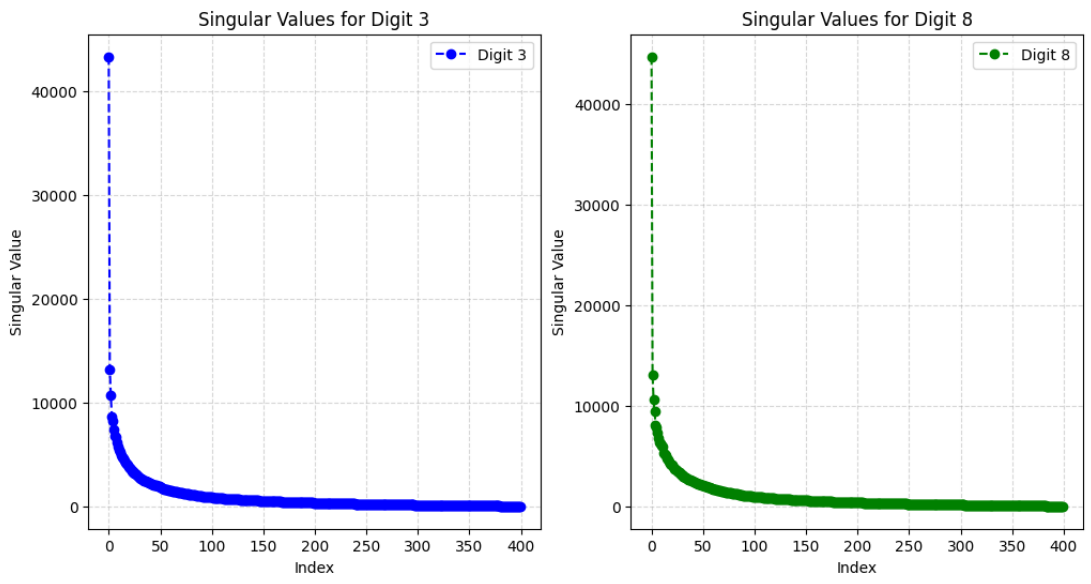

Figure 1: Singular value plots for digits 3 and 8, generated from 400 training images each. The approximation of U is obtained with $k = 15$.

3.2 Singular Images

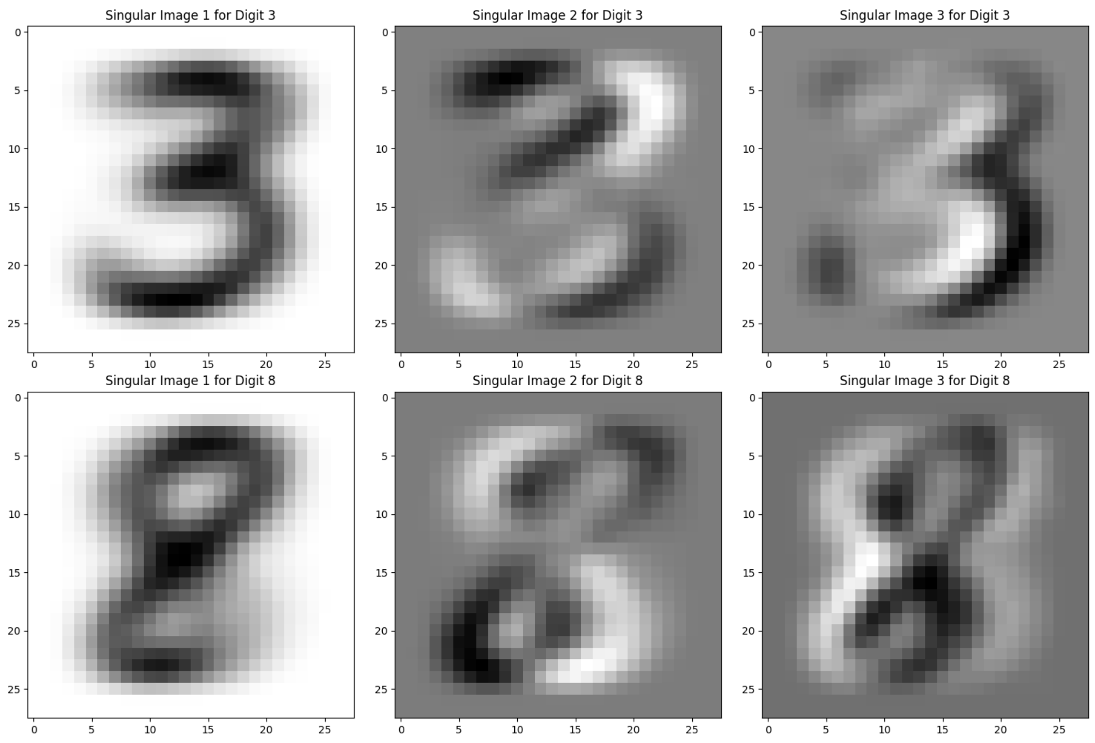

Figure 2: Plots of the first three singular images $\vec{u}_1, \vec{u}_2, \vec{u}_3$ for digits 3 and 8. Using 400 training images for each digit and approximating U with $k = 15$.

3.3 Classification Accuracy

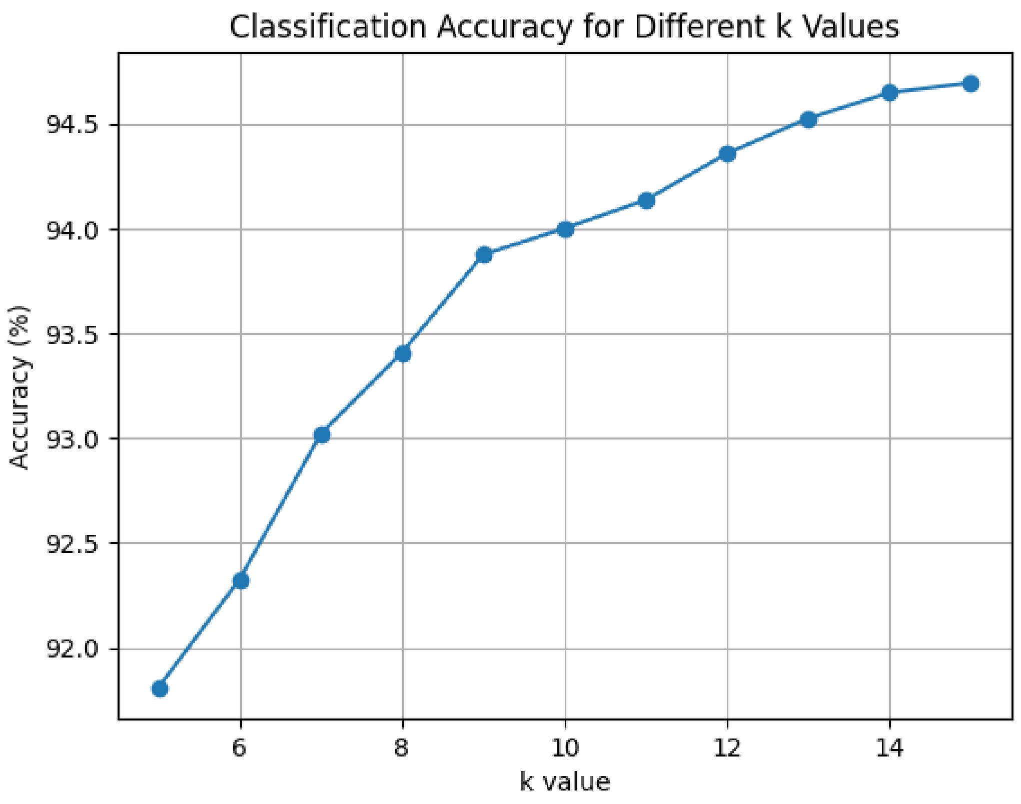

Figure 3: Success of classification accuracy vs. k using 400 training images, $k = 15$ approximation.

4 Discussion

For the first part, we break down a dataset into components that capture important features of our data. Singular values are measures of how important or influential a particular feature is in representing the dataset. A higher singular value means that the feature is more significant. By analyzing our result in figure 1 we see that a lower index would provide the most significant information. For a higher index, the singular value decreases and thus contains less important information. In figure 2 the first three singular images for digits 3 and 8 respectively are shown. From there, it can be observed that the first image for both digits essentially represents an outline for the figure. When increasing the index further, details and contrasts are revealed. However, we see a larger difference between pictures 1 and 2, than between 2 and 3 (see figure 2) and this would be due to a lower singular value for each higher index.

In our classification process, we measure the accuracy of recognizing handwritten digits by comparing test images against our trained model for different values of k, which indicates the number of singular values and vectors used. For each k, we calculate the residuals using the model. The digit with the smallest residual is the predicted classification. By varying k, we observe how the number of singular values used affects the model's accuracy. In figure 3 we observe that a higher value of k would result in a higher accuracy. This is due to the fact that using more singular values in the reconstruction of test digits allows for a more detailed and accurate representation of each image.

A Appendix

```python
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt

# Load data
train_digits = np.load('TrainDigits.npy') # Training images
train_labels = np.load('TrainLabels.npy') # Labels for training images
test_digits = np.load('TestDigits.npy') # Test images
test_labels = np.load('TestLabels.npy') # Labels for test images

# Parameters setting
num_cols = 400 # Number of columns to use from each digit matrix
num_digits = 10 # Total number of digits (0-9)
k_range = range(5, 16) # Range of k values for SVD

# Create training matrices for each digit and compute SVD
training_matrices = {digit: train_digits[:, (train_labels == digit).flatten()][:,
    :num_cols]
                    for digit in range(num_digits)}
svd_components = {digit: np.linalg.svd(matrix, full_matrices=False)
                 for digit, matrix in training_matrices.items()}
U_k_list = [svd_components[digit][0][:, :k_range[-1]]
           for digit in range(num_digits)]

# Function to plot singular values for selected digits
def plot_singular_values(digit, color, label):
    singular_values = svd_components[digit][1] # Extract singular values
    plt.plot(singular_values, marker='o', linestyle='--', color=color, label=label)
    plt.xlabel('Index')
    plt.ylabel('Singular Value')
    plt.title(f'Singular Values for Digit {digit}')
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.5)
```

```
# Plotting singular values for digits 3 and 8
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plot_singular_values(3, 'blue', 'Digit 3')
plt.subplot(1, 2, 2)
plot_singular_values(8, 'green', 'Digit 8')
plt.show()

# Function to plot singular images for a digit
def plot_singular_images(digit, start_subplot):
    for i in range(3): # Display first 3 singular images
        plt.subplot(2, 3, start_subplot + i)
        singular_image = svd_components[digit][0][:, i].reshape(28, 28).T # Reshape
            singular vector into image
        plt.imshow(singular_image, cmap='gray')
        plt.title(f'Singular Image {i+1} for Digit {digit}')

# Plotting singular images for digits 3 and 8
plt.figure(figsize=(15, 10))
plot_singular_images(3, 1)
plot_singular_images(8, 4)
plt.tight_layout()
plt.show()

# Compute classification accuracy for each k
accuracy_k = []
for k in k_range:
    # Initialize matrix to store residuals for each digit
    residuals_matrix = np.zeros((num_digits, test_digits.shape[1]))
    for digit in range(num_digits):
        U_k = U_k_list[digit][:, :k] # Use first k columns of U
        # Compute residuals for each test image
        residuals = np.linalg.norm(test_digits - U_k @ U_k.T @ test_digits, axis=0)
        residuals_matrix[digit] = residuals
    # Find the digit with the smallest residual for each test image
    predictions = np.argmin(residuals_matrix, axis=0)
    # Calculate accuracy as the percentage of correct predictions
    accuracy = np.mean(test_labels[0] == predictions) * 100
    accuracy_k.append(accuracy)

# Plot accuracy vs k values
plt.plot(k_range, accuracy_k, marker='o')
plt.xlabel('k value')
plt.ylabel('Accuracy (%)')
plt.title('Classification Accuracy for Different k Values')
plt.grid(True)
plt.show()
```