Uppsala University

Scientific Computing for Data Analysis

Mini-Project Group Kand.02

Digit Classification Using the Singular Value Decomposition Approach

Authors:
Hamit Efe Çınar, Mehmet Fırat Dündar

July 3, 2024

1 Introduction

The digit classification task involves identifying handwritten digits from images. In this mini-project, we explore a method based on using basis vectors to represent and classify digits. The primary goal is to develop an algorithm that accurately classifies digits using a subset of basis vectors extracted from the labeled training data.

2 Approach

Our approach to the problem involves several key steps:

2.1 Data Loading and Preprocessing

We begin by loading the training and testing datasets, which consist of images of handwritten digits along with their corresponding labels. These datasets are then preprocessed (dimension flattening) to ensure compatibility with the subsequent steps.

2.2 Basis Vector Computation

For each digit, we compute a set of basis vectors using Singular Value Decomposition (SVD). These basis vectors capture the essential features of each digit and serve as a reduced representation of the original data. The main idea behind this SVD approach is that it is sufficient to do a low rank computation expecting that the basis matrices have low rank, thus considering certain columns of the decomposed elements of the matrix is enough for the computation so that we have a faster calculation and the principal components of the images are taken into consideration.

2.3 Residual Calculation

We compute the residuals for the testing data using the given formula $\text{residual} = \left\| \left(I - U_k U_k^T\right) d \right\|_2$, where U_k represents the matrix of basis vectors and d is the input digit.

2.4 Classification

Based on the computed residuals, we classify each digit by selecting the digit corresponding to the minimum residual.

2.5 Performance Evaluation

We evaluate the performance of the classification algorithm by comparing the predicted labels with the true labels of the testing data. The percentage of correct classifications is calculated for various values of k, representing the number of basis functions used.

3 Results

The results of our experiments are summarized as follows:

3.1 Task 1

We have addressed the task of computing the solution of the least squares problem and the residual using the provided mathematical formulations. The solution $x = U_k^T d$ and the residual $= \left\| \left(I - U_k U_k^T \right) d \right\|_2$ have been derived from orthogonality and implemented in the below function:

```
def compute_residual(k, TestMat, TestLabel, TrainLabel, TrainMat):
    # Initialize counters
    correct_predictions = 0
    digit_counts = {i: 0 for i in range(10)}
    correct_digit_counts = {i: 0 for i in range(10)}

    # Compute the projection matrices for each digit
    Uk_Ukt = Compute_UK(k, TrainLabel, TrainMat)

    # Loop through each sample in the test set
    for x in range(40000):
        # Calculate the residuals for the current sample
        residuals = [np.linalg.norm(Uk_Ukt[key] @ TestMat[:, x]) for key in Uk_Ukt]

        # Find the minimum residual and the corresponding expected number
        residual_min = min(residuals)
        expected_number = residuals.index(residual_min)

        # Update counters
        actual_number = TestLabel[0][x]
        if expected_number == actual_number:
            correct_predictions += 1
            correct_digit_counts[actual_number] += 1
        digit_counts[expected_number] += 1

    # Calculate total accuracy
    total_percentage = (correct_predictions / 40000) * 100
    print(f'Accuracy: {total_percentage:.2f}%')

    # Calculate and print the accuracy for each digit
    for digit in range(10):
        digit_accuracy = (correct_digit_counts[digit] / 4000) * 100
        print(f'Percentage of correct {digit} with k={k}: {digit_accuracy:.2f}%')

    return correct_predictions, correct_digit_counts
```

3.2 Task 2

Using the approach and the residual function given above, we have acquired the corresponding U matrices and the S vector, which we utilize for Singular Image plotting and Singular Value plotting respectively for the task 2. One plot for each of the digits 3 and 8, showing singular values gotten from the SVD computation, are given below where one can see that singular values for 3(1) and 8(2) show a fairly similar decay:

Additionally, using the results from the SVD computation, we've also plotted the first three

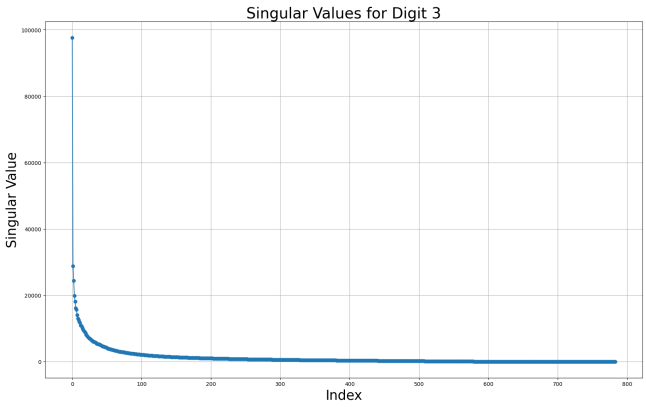

Figure 1: Singular values for 3

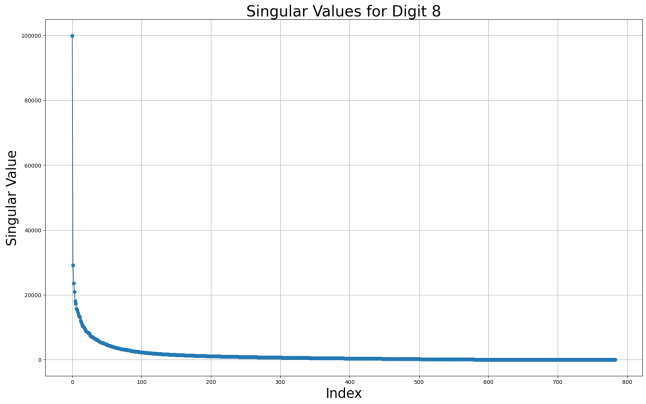

Figure 2: Singular values for 8

singular images for the digits 3 and 8, separately which can be seen on Figure 3.

3.3 Task 3: Testing and Evaluation

The algorithm has been tested on all test digits, and the classification accuracy has been calculated for various values of k. The success percentage of the algorithm has been compared with the corresponding test labels by different number of basis functions (see Figure 4).

4 Discussion

In the task 2, we have observed that the singular values for 3 and 8 show a fairly similar decrease index by index with near values. We can also observed that as the first singular value for each digit is fairly greater than any other. The similarity of the values for the digits 3 and 8 might be due to their shared features in terms of their shape structure, similar handwriting styles and overlapping variability in data in their corresponding train dataset.

For the Task 3, it can be seen from the figure 5 that the rate of increase in accuracy caused by the increase in number of basis functions decrease as the number increase. We can suspect that there is a certain threshold where the increase in accuracy stop or even we may observe a decrease as the number of basis functions increase. This may due to the ill-relevancy of considering a new dimension. Further investigation could focus on optimizing the selection of basis functions or exploring alternative classification techniques to enhance accuracy.

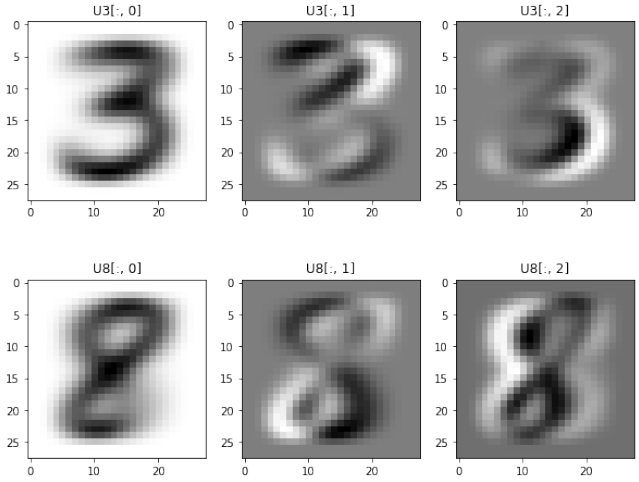

Figure 3: First three singular images for Digit 8 and 3

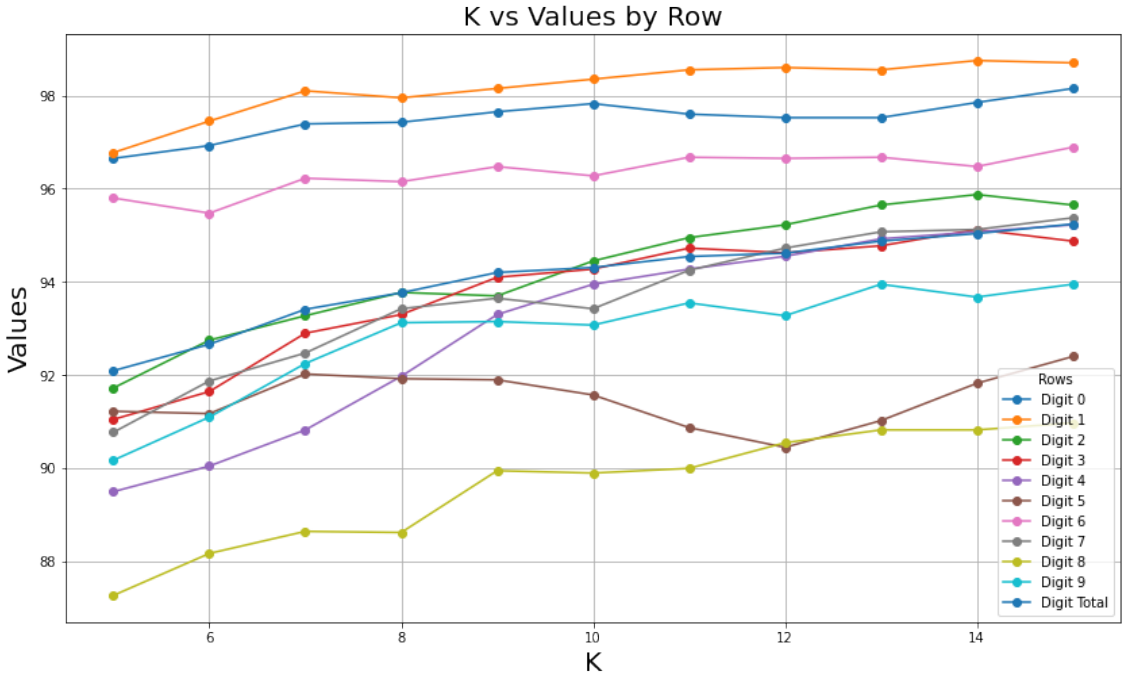

Figure 4: The accuracy of the model vs. the number of basis functions

5 Conclusion and Outlook

In conclusion, we have developed a digit classification algorithm based on a basis vectors approach using Singular Value Decomposition. The algorithm demonstrates promising results in accurately classifying handwritten digits. Future work could involve refining the algorithm parameters, exploring additional feature extraction techniques, and integrating machine learning models for enhanced classification performance. It was interesting to see certain changes in images using different singular values, and comparing different number of basis functions helped us to understand the method more deeply.

6 Appendix

A The Whole Code

```
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('png','pdf')
TrainMat = np.load('TrainDigits.npy')
TrainLabel = np.load('TrainLabels.npy')
TestMat = np.load('TestDigits.npy')
TestLabel = np.load('TestLabels.npy')
print("TrainMat shape:", TrainMat.shape)
print("TrainLabel shape:", TrainLabel.shape)
print("TestMat shape:", TestMat.shape)
print("TestLabel shape:", TestLabel.shape)
def sort_function(i, TrainLabel, TrainMat):
    # Flatten the TrainLabel array
    flattened_list = np.ndarray.flatten(TrainLabel)

    # Initialize an empty matrix with shape (784, 0)
    num_matrix = np.empty((784, 0))

    # Counter to track the number of columns added
    n = 0

    # Loop through indices where the flattened list equals the given i
    for num in np.argwhere(flattened_list == i).flatten():
        if n < 2000:
            # Extract the column to add from TrainMat
            column_to_add = TrainMat[:, num].reshape(-1, 1)

            # Horizontally stack the new column to the num_matrix
            num_matrix = np.hstack((num_matrix, column_to_add))

            # Increment the counter
            n += 1

    # Convert the matrix to float32 type
    num_matrix = num_matrix.astype(np.float32)

    return num_matrix
    def plot_digit_3_and_8_function(TrainLabel, TrainMat):
    # Get the sorted matrices for digits 3 and 8
    A3 = sort_function(3, TrainLabel, TrainMat)
    A8 = sort_function(8, TrainLabel, TrainMat)

    # Perform Singular Value Decomposition (SVD) on the matrices
    U3, S3, Vt3 = np.linalg.svd(A3)
    U8, S8, Vt8 = np.linalg.svd(A8)

    # Prepare the x-axis values (0 to 783)
    x = list(range(784))

    # Prepare the y-axis values for singular values
    y3 = list(S3)
    y8 = list(S8)
```

```
# Create subplots for the singular values
fig, axes = plt.subplots(1, 2, figsize=(10, 4))

axes[0].scatter(x, y3)
axes[0].set_title('Number 3 Singular Values')

axes[1].scatter(x, y8)
axes[1].set_title('Number 8 Singular Values')

# Create subplots for the reshaped vectors
fig, axarr = plt.subplots(2, 3, figsize=(10, 8))

# Reshape the vectors to 28x28 matrices and transpose
D1 = np.reshape(U3[:, 0], (28, 28)).T
D2 = np.reshape(U3[:, 1], (28, 28)).T
D3 = np.reshape(U3[:, 2], (28, 28)).T

D4 = np.reshape(U8[:, 0], (28, 28)).T
D5 = np.reshape(U8[:, 1], (28, 28)).T
D6 = np.reshape(U8[:, 2], (28, 28)).T

# Display the reshaped matrices
axarr[0, 0].imshow(D1, cmap='gray')
axarr[0, 0].set_title('U3[:, 0]')

axarr[0, 1].imshow(D2, cmap='gray')
axarr[0, 1].set_title('U3[:, 1]')

axarr[0, 2].imshow(D3, cmap='gray')
axarr[0, 2].set_title('U3[:, 2]')

axarr[1, 0].imshow(D4, cmap='gray')
axarr[1, 0].set_title('U8[:, 0]')

axarr[1, 1].imshow(D5, cmap='gray')
axarr[1, 1].set_title('U8[:, 1]')

axarr[1, 2].imshow(D6, cmap='gray')
axarr[1, 2].set_title('U8[:, 2]')

plt.show()

def plot_function(Uk):
# Reshape the first column of Uk to a 28x28 matrix and transpose it
D = np.reshape(Uk[:, 0], (28, 28)).T

# Display the matrix as a grayscale image
plt.imshow(D, cmap='gray')
plt.title('Digit Representation')
plt.axis('off') # Hide the axis for better visualization
plt.show()
```

```
def plot_all_digits(digit_dict):
# Create subplots with 1 row and 10 columns
fig, axarr = plt.subplots(1, 10, figsize=(15, 3))

for i in range(10):
    # Reshape the first column of each digit's matrix to a 28x28 matrix and transpose it
    D = np.reshape(digit_dict[i][:, 0], (28, 28)).T
    # Display the reshaped matrix as a grayscale image
    axarr[i].imshow(D, cmap='gray')
    axarr[i].axis('off') # Hide the axis for better visualization
    axarr[i].set_title(f'Digit {i}', fontsize=10) # Add title for each subplot

plt.show()
def Compute_SVD(M):
return np.linalg.svd(M, full_matrices=False)
def Compute_UK(k, TrainLabel, TrainMat):
uk_dict = {}
projection_dict = {}

for num in range(10):
    # Compute the SVD of the sorted function's result
    U, S, Vt = Compute_SVD(sort_function(num, TrainLabel, TrainMat))

    # Get the first k columns of U
    U_k = U[:, :k]
    uk_dict[num] = U_k

    # Compute the orthogonal projection matrix
    U_kt = U_k.T
    projection_matrix = np.identity(784) - (U_k @ U_kt)
    projection_dict[num] = projection_matrix

return projection_dict

def compute_residual(k, TestMat, TestLabel, TrainLabel, TrainMat):
# Initialize counters
correct_predictions = 0
digit_counts = {i: 0 for i in range(10)}
correct_digit_counts = {i: 0 for i in range(10)}

# Compute the projection matrices for each digit
Uk_Ukt = Compute_UK(k, TrainLabel, TrainMat)

# Loop through each sample in the test set
for x in range(40000):
    # Calculate the residuals for the current sample
    residuals = [np.linalg.norm(Uk_Ukt[key] @ TestMat[:, x]) for key in Uk_Ukt]

    # Find the minimum residual and the corresponding expected number
    residual_min = min(residuals)
    expected_number = residuals.index(residual_min)

    # Update counters
```

```
    actual_number = TestLabel[0][x]
    if expected_number == actual_number:
        correct_predictions += 1
        correct_digit_counts[actual_number] += 1
    digit_counts[expected_number] += 1

# Calculate total accuracy
total_percentage = (correct_predictions / 40000) * 100
print(f'Accuracy: {total_percentage:.2f}%')

# Calculate and print the accuracy for each digit
for digit in range(10):
    digit_accuracy = (correct_digit_counts[digit] / 4000) * 100
    print(f'Percentage of correct {digit} with k={k}: {digit_accuracy:.2f}%')

return correct_predictions, correct_digit_counts

compute_residual(15, TestMat, TestLabel, TrainLabel, TrainMat)

plot_digit_3_and_8_function(TrainLabel, TrainMat)
```