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

May 9, 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.

2.3 Residual Calculation

We compute the residuals for the testing data using the given formula 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_residuals(Uk_matrices, test_data):
    residuals = np.zeros((10, test_data.shape[1]))
    for digit in range(10):
        Uk = Uk_matrices[digit]
        I_minus_Uk_UkT = np.identity(Uk.shape[0]) - Uk @ Uk.T
        for i in range(test_data.shape[1]):
            d = test_data[:, i]
            residual = np.linalg.norm(I_minus_Uk_UkT @ d)
            residuals[digit, i] = residual
    return residuals
```

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 successfully acquired:

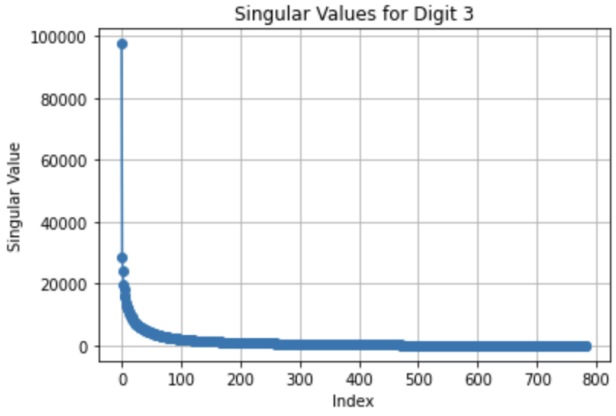

Figure 1: Singular values for 3

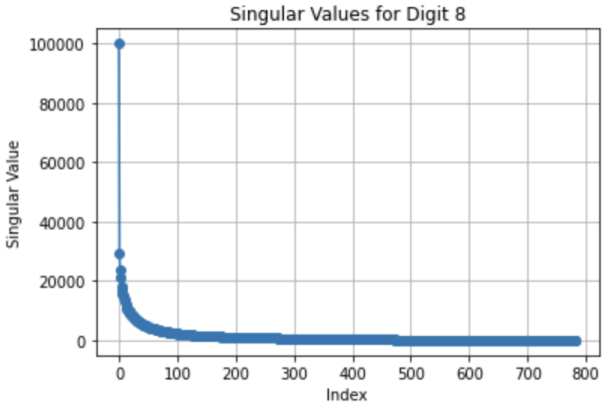

Figure 2: Singular values for 8

Additionally, using the results from the SVD computation, we've also plotted the first three singular images for the digits 3 and 8, separately:

Figure 3: First three singular images for Digit 3

Figure 4: First three singular images for Digit 8

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:

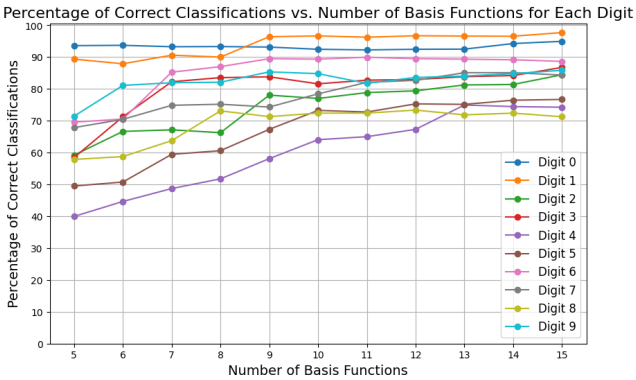

Figure 5: The accuracy of the model vs. the number of basis functions

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, so we can understand why methods like power method can successfully work. 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.

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').flatten()
TestMat = np.load('TestDigits.npy')
TestLabel = np.load('TestLabels.npy').flatten()

print("TrainMat shape:", TrainMat.shape)
print("TrainLabel shape:", TrainLabel.shape)
print("TestMat shape:", TestMat.shape)
print("TestLabel shape:", TestLabel.shape)

fig, axs = plt.subplots(1, 10, figsize=(15, 3))
for i in range(10):
    digit_index = np.where(TrainLabel == i)[0][0]
    digit = TrainMat[:, digit_index].reshape(28, 28)
    axs[i].imshow(digit, cmap='gray')
    axs[i].set_title(f"Digit {i}")
    axs[i].axis('off')
plt.show()

def sort_data_by_label(data, labels, digit):
    indices = np.where(labels == digit)[0]
    return data[:, indices]

digit_3_train = sort_data_by_label(TrainMat, TrainLabel, 3)
print("Sorted data for digit 3 shape:", digit_3_train.shape)

def compute_svd_and_plot_singular_values(train_data, digit, num_samples):
    digit_train_data = sort_data_by_label(train_data, TrainLabel, digit)
    digit_train_data_subset = digit_train_data[:, :num_samples].astype(np.float32)
```

```
    _, S, _ = np.linalg.svd(digit_train_data_subset)
    plt.plot(S, marker='o')
    plt.title(f'Singular Values for Digit {digit}')
    plt.xlabel('Index')
    plt.ylabel('Singular Value')
    plt.grid(True)
    plt.show()

compute_svd_and_plot_singular_values(TrainMat, digit=3, num_samples=2000)
compute_svd_and_plot_singular_values(TrainMat, digit=8, num_samples=2000)

def plot_singular_images(U_matrix, digit):
    for i in range(3):
        plt.subplot(1, 3, i+1)
        plt.imshow(U_matrix[:, i].reshape(28, 28), cmap='gray')
        plt.title(f'Singular Image {i+1}')
        plt.axis('off')
    plt.suptitle(f'First Three Singular Images for Digit {digit}')
    plt.show()

def compute_Uk_matrices(train_data, num_basis_functions):
    Uk_matrices = {}
    for digit in range(10):
        digit_train_data = sort_data_by_label(train_data, TrainLabel, digit)
        digit_train_data_subset = digit_train_data[:, :num_basis_functions]
        # Convert to single precision floats
        digit_train_data_subset = digit_train_data_subset.astype(np.float32)

        # Compute SVD
        U, _, _ = np.linalg.svd(digit_train_data_subset)

        # Store only the first k columns of U
        Uk_matrices[digit] = U[:, :num_basis_functions]
    return Uk_matrices

Uk_matrices = compute_Uk_matrices(TrainMat, num_basis_functions=2000)
# Plot first three singular images for digits 3 and 8
plot_singular_images(Uk_matrices[3], digit=3)
plot_singular_images(Uk_matrices[8], digit=8)


Uk_matrices = compute_Uk_matrices(TrainMat, num_basis_functions=15)

def compute_residuals(Uk_matrices, test_data):
    residuals = np.zeros((10, test_data.shape[1]))
    for digit in range(10):
        Uk = Uk_matrices[digit]
        I_minus_Uk_UkT = np.identity(Uk.shape[0]) - Uk @ Uk.T
        for i in range(test_data.shape[1]):
            d = test_data[:, i]
            residual = np.linalg.norm(I_minus_Uk_UkT @ d)
            residuals[digit, i] = residual
    return residuals
```

```
test_residuals = compute_residuals(Uk_matrices, TestMat)

def classify_digits(residuals):
    return np.argmin(residuals, axis=0)

predicted_labels = classify_digits(test_residuals)
print("Predicted labels shape:", predicted_labels.shape)

correct_classifications = np.sum(predicted_labels == TestLabel)
total_digits = TestLabel.shape[0]
percentage_correct = (correct_classifications / total_digits) * 100
print("Percentage of correct classifications:", percentage_correct)

num_basis_functions_range = range(5, 16)
digit_success_percentage = {}

for digit in range(10):
    success_percentages = []
    digit_test_data = sort_data_by_label(TestMat, TestLabel, digit)
    digit_test_data = digit_test_data[:, :2000]
    digit_test_label = np.full((digit_test_data.shape[1],), digit)
    for num_basis_functions in num_basis_functions_range:
        Uk_matrices = compute_Uk_matrices(TrainMat, num_basis_functions=num_basis_functions)
        test_residuals = compute_residuals(Uk_matrices, digit_test_data)
        classified_digits = classify_digits(test_residuals)
        correct_classification = np.sum(classified_digits == digit_test_label)
        success_percentage = correct_classification / len(digit_test_label) * 100
        success_percentages.append(success_percentage)
    digit_success_percentage[digit] = success_percentages

# Plotting
plt.figure(figsize=(10, 6))
for digit in range(10):
    plt.plot(num_basis_functions_range, digit_success_percentage[digit], marker='o', label=f'Di
plt.title('Percentage of Correct Classifications vs. Number of Basis Functions for Each Digit'
plt.xlabel('Number of Basis Functions', fontsize=14)
plt.ylabel('Percentage of Correct Classifications', fontsize=14)
plt.grid(True)
plt.xticks(num_basis_functions_range)
plt.yticks(range(0, 101, 10))
plt.legend(loc='lower right', fontsize=12)
plt.show()
```