Miniproject 2: Classification of Handwritten Digits

Scientific Computing for Data Analysis, 2024

Davoud Mirzaei

Classification of handwritten digits is a standard problem in pattern recognition. A typical application is the automatic reading of zip codes on envelopes. This miniproject aims to practice a classification technique using SVD.

The problem is as follows: *Given a set of manually classified digits as a training set, classify a set of unknown digits as the test set.*

In Figure 1 a sample of handwritten digits $0, 1, \dots, 9$ is illustrated. The images are downloaded from[1]. Each image is a 28×28 grayscale image but we stack the columns of the image above each other so that each image is represented by a vector of size 784.

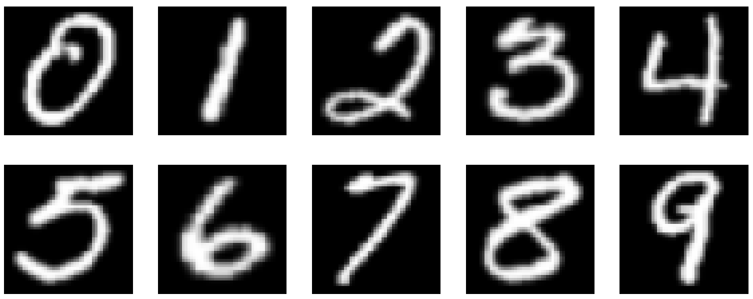

Figure 1: A sample of handwritten digits $0, 1, \dots, 9$.

The training set of each kind of digits $0, 1, \dots, 9$ can be considered as a matrix A of size $m \times n$ with $m = 784$ and $n =$ 'the number of training digits of one kind'. Usually, $n \geqslant m$ although the case $n < m$ is also possible. So, we have a total of 10 training matrices, each corresponding to one of the digits $0, 1, \dots, 9$. Assume that A is one of them. The columns of A span a linear subspace of $\mathbb{R}^{784}$. However, this subspace cannot be expected to have a large dimension, because columns of A represent the same digit with different handwritings. One can use SVD $A = U\Sigma V^T$ to obtain an orthogonal basis for the column space of A (or range(A)) via the columns of factor U. Then any test image of that kind can be well represented in terms of the orthogonal basis $\{\boldsymbol{u}_1, \boldsymbol{u}_2, \dots, \boldsymbol{u}_m\}$. Let us describe it in more detail. We have

$$A = U\Sigma V^T = \sum_{j=1}^{m} \sigma_j \boldsymbol{u}_j \boldsymbol{v}_j^T,$$

[1]https://www.nist.gov/itl/products-and-services/emnist-dataset

thus the ℓ-th column of A (the ℓ-th training digit) can be represented as

$$\boldsymbol{a}_\ell = \sum_{j=1}^{m} (\sigma_j v_{j\ell}) \boldsymbol{u}_j$$

which means that the coordinates of image $\boldsymbol{a}_\ell$ in terms of orthogonal basis $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m\}$ are $\sigma_j v_{j\ell} =: x_j$, i.e.

$$\boldsymbol{a}_\ell = \sum_{j=1}^{m} x_j \boldsymbol{u}_j = U\boldsymbol{x}$$

This means, in another point of view, that solving the least squares problem

$$\min_{\boldsymbol{x}} \|U\boldsymbol{x} - \boldsymbol{a}_\ell\|_2$$

results in an optimal vector $\boldsymbol{x} = [\sigma_1 v_{1\ell}, \ldots, \sigma_m v_{m\ell}]^T$ with residual 0. As we pointed out, most columns of A are nearly linearly dependent as they represent different handwritten for the same digit. If we translate it to columns of U, this means that a few leading columns of U should well capture the subspace. For this reason the orthogonal vectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots,$ are called *singular images.* See Task 2 below. Consequently, we can use a k-rank approximation of A where $k \ll m$. In this case each $\boldsymbol{a}_\ell$ can be well represented by orthogonal basis $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k\}$ in a least squares sense with a small residual.

Now, let $\boldsymbol{d}$ be a digit of size $m = 784$ outside the training set (which is called a *test digit*). It is reasonable to assume that $\boldsymbol{d}$ can be well represented in terms of singular images $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k\}$ of its own kind with a relatively small residual. We must compute how well $\boldsymbol{d}$ can be represented in the 10 different bases, each corresponding to one of the digits $0, 1, \ldots, 9$. This can be done by computing the residual vector in the least squares problems of the type

$$\min_{\boldsymbol{x}} \|U_k\boldsymbol{x} - \boldsymbol{d}\|_2 \tag{0.1}$$

where $\boldsymbol{d}$ represents a test digit and U_k is a $m \times k$ matrix consisting of the first k columns of U.

Altogether, a SVD-based classification algorithm of handwritten digits consists of two steps:

- **Step 1 (training):** For the training set of known digits, compute the SVD of each set of digits of one kind (Compute ten SVDs for digits $0, 1, \ldots, 9$).
- **Step 2 (classification):** For a given test digit $\boldsymbol{d}$, compute its relative residual in all ten bases using equation (0.2) below. If a residual in a class is smaller than all the others, classify $\boldsymbol{d}$ in that class.

To get started with the miniproject, download the *training* set `TrainDigits.npy` consisting of 240000 images (24000 images per any digit), and the *test* set `TestDigits.npy` consisting of 40000 images (4000 images per digit) as a zip document from the Miniproject page in Studium. The document contains *label* files `TrainLabels.npy` and `TestLabels.npy` as well, which are manual classifications of training and test sets, respectively. The training labels will be used to learn and the test labels to verify your algorithm.

Note: In tasks 2 and 3 below, for each digit consider using only the initial 400 images out of the total 24000 training images.

- **Task 1:** (optional) Show that the solution of least square problem (0.1) is $\boldsymbol{x} = U_k^T \boldsymbol{d}$ and the residual is

$$residual = \|(I - U_k U_k^T)\boldsymbol{d}\|_2. \tag{0.2}$$

  Hint: Use the orthogonality of columns of U.

- **Task 2:** For training matrices of digits 3 and 8 (each of size 784×400), compute the SVD and plot the singular values (two plots). Use linear scales for both axes. Report your observation and try to justify it. Also plot the first three singular images $\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3$ for digits 3 and 8, separately (six plots), and report your observation.

- **Task 3:** Implement the above SVD-based classification algorithm in a Python code. Use only $n = 400$ training images for each digit (ten matrices each of size 784×400). However, test the algorithm on all 40000 test images and compare your classifications with the exact labels `TestLabels.npy`. Use different values $k = 5, 6, \ldots, 15$ (number of basis functions), and report the percentage of the success of this SVD-based algorithm for each digit with different k values.

Some Comments

The following comments may help you to write a more optimized code for your miniproject.
1. To load images you can write, e.g., `TrainDigits = numpy.load('TrainDigits.npy')`. Be sure that .npy files are copied in the directory you are working in. To plot the digits you need to reshape the columns into 28×28 matrices using `numpy.reshape` and then use `imshow` command to plot the digit. For example here we plot the first digit in the training set:

```
import numpy as np
import matplotlib.pyplot as plt
TrainDigits = np.load('TrainDigits.npy')
d = TrainDigits[:,0]            # The first digit in the training set
D = np.reshape(d, (28, 28)).T  # Reshaping a vector to a matrix
plt.imshow(D, cmap ='gray')    # Plot of the digit
```

2. If `a` is a Numpy array and `i` is a scalar value then `a==i` performs an element-wise comparison between them. In other words, if we do `index = (a==i)`, then `index[j]` is True if `a[j]==i` and False if `a[j]!=i`. If we then do `np.sum(index)`, we will get the number of Trues in `index`. And, you can use boolean arrays to index other Numpy arrays

of suitable size. For example, if `len(index)` is equal to the number of columns in `X`, then `X[:,index]` would return the columns of `X` for which the corresponding element in `index` is True. Use this comment to identify specific training labels `i` (for `i=0,1,...,9`) from the big array `TrainLabels` and extract corresponding training digits from big array `TrainDigits`. Then use only the first 400 images for each digit.

3. You must precompute all ten SVDs and store U_k matrices for all ten digits (store U_k for maximum k, here 15, and then pick up smaller subsets for $k = 5, 6, \dots, 15$). Then use the precomputed matrices for all 40000 test digits. Also, avoid the products $U_k U_k^T$ for each test digit; once again, precompute these products.

4. In the statement above, the equations you need are formulated for a single test vector $\boldsymbol{d}$ (one "flattened" test image). But if you use that approach in your code, it will take a very long time to run. In contrast, matrix-matrix multiplications are much more efficient on modern computers. To take full advantage of that, simply replace vector $\boldsymbol{d}$ in Equation (0.2) with the entire test matrix. Then you can apply `np.linalg.norm(..,axis = 0)` to get residuals for all test points, in one vector of length 40000.

5. If A is a matrix, the function `np.argmin(A, axis=0)` returns a vector containing the row number of the minimal value in each column of A. If you store the aforementioned residual vectors for all 10 digits in one matrix, you can use this function to efficiently predict digits from residuals. In general, take note of this "axis" argument in many Numpy functions: it lets you perform an operation along a specific dimension of a matrix (that is, along rows or columns).

Report

Your mini-project report should be submitted as a PDF file, and it should include your Python code snippets and figures, along with discussions, and your comments on your observations. Provide proper captions and labels for figures. You can write the report using stochSS, Google Colab, or your preferred environment. For instructions on how to write a report using stochSS and Google Colab, visit the page "About Miniprojects" on the Studium course page.