Miniproject 2
Classification of handwritten digits using SVD

Linus Falk

October 6, 2022

Introduction

Pattern recognition can be solved with various methods, this project aims to solve the classic recognition of handwritten digits with the use of singular value decomposition or SVD for short.

Singular value decomposition

The SVD decomposition or factorization exist for any a real or complex matrix, rectangular or square. The SVD factorization of a matrix M is presented below:

$$M = U\Sigma V^* \tag{1}$$

Where U and V^* are unitary matrices and the columns are sets of orthonormal vectors and can be seen as basis vectors of the column space and row space. These two matrices represent rotation and reflection while Σ act as scaling [1].

The data set

The data set to create this model to recognise handwritten digits consists of a training and test set with pictures of handwritten digits made of 28x28 pixels. These pictures are flatten to a column of 784 numbers representing the gray scale of in the picture. The sets are accompanied by the labels for each digit in the training and test set [2] . Example of digit presented below:

Method

The idea is to use the SVD decomposition of ten different matrices of constructed of training digit pixel values: $A_0, A_1, \ldots, A_9$ where each columns consists of the 784 pixel values for a digit. Using the SVD of each matrix we obtain the orthonormal basis for the column space of each A. Then the test images can be represented in terms of these orthonormal basis. Because most of the columns of A are nearly linearly dependent (caused by similarity between the representation of the same digit) we only need a couple of the to represent the digit. To test the similarity between a test digit and one of theses bases we calculate the following residual:

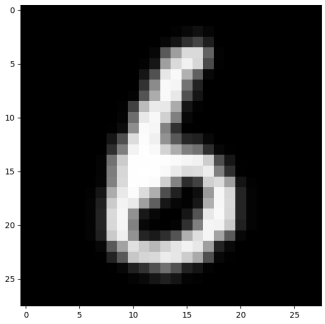
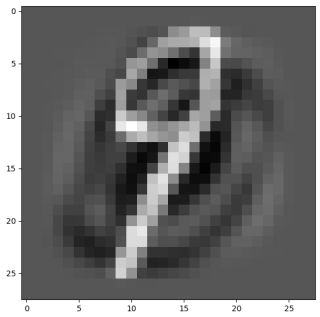


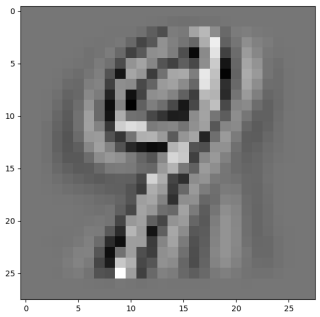
Figure 1: Example of digit

$$\|(I - U_k^T U_k)d\|_2 \tag{2}$$

If the test digit would be a perfect representation captured by the chosen columns of U, the residual would be zero. By testing the test digit with the different U's and compare the residual we can conclude what digit it is most likely to be.



(a) U_0 with test digit 9



(b) U_9 with test digit 9

Figure 2: A digit 9 tested with 20 columns of U for the digit zero and nine

Implementation

The implementation was done using a Python script were the first step was to sort all test digits in a matrix with the same digit: A_0 with all zeros and so on. This was done for two cases: 1: with all training digits and 2: with 7500 training digits. The SVD was then calculated for each of these cases and type of digit.

Comparing the residuals of a test digit with equation (2) and choosing the one with lowest yielded the predictions that was then checked with the label.

```
A = [[],[],[],[],[],[],[],[],[],[]]

for j in range(1000):
    A[TrainLabels[0][j]].append(TrainDigits[:, j])

U = []
for i in range(10):
    d = A[i]
    d = np.asarray(d)
    d = d.transpose()
    u, s, vh = np.linalg.svd(d)
    U.append(u)
```

Testing different number of columns was done using a reduced number of test digits (200) and a for loop to change the number of columns.

```
NumerOfTests = 200
for t in range(2,31,1):
    precictions = []
    for i in range(NumerOfTests):
        result = []
        for x in range(10):
            u1 = U[x]
            u1 = u1[:, :t]
            result.append(np.linalg.norm((I -
            u1@u1.transpose())@TestDigits[:, i]))

        min_value = min(result)
        min_index = result.index(min_value)
        if min_index == TestLabels[0][i]:
            precictions.append(1)
        else:
            precictions.append(0)

    precentage = (sum(precictions)/NumerOfTests)*100
    print(str(t) + ' ' + str(precentage) + '%')
```

When the approximation of the optimal number of columns was done was the complete set of test digits tested.

Result and discussion

The result of the tests with 200 test digits to establish some guidance to choose how many columns: **n** to use is presented below:

n	acc%	n		n		n		n		n	
2	81.0	**7**	91.5	**12**	93.0	**17**	91.0	**22**	90.5	**27**	92.5
3	85.5	**8**	91.0	**13**	92.5	**18**	92.0	**23**	91.5	**28**	92.5
4	84.0	**9**	93.0	**14**	92.0	**19**	92.0	**24**	90.5	**29**	93.0
5	87.0	**10**	92.0	**15**	92.5	**20**	92.0	**25**	92.0	**30**	93.5
6	88.0	**11**	92.5	**16**	92.0	**21**	91.5	**26**	92.5		

Table 1: Accuracy, trained with 7500 training digits

n	acc%	n		n		n		n		n	
2	80.0	**7**	91.0	**12**	92.5	**17**	91.0	**22**	91.0	**27**	92.5
3	86.0	**8**	90.0	**13**	92.5	**18**	91.5	**23**	91.5	**28**	92.0
4	87.0	**9**	92.0	**14**	91.5	**19**	91.5	**24**	92.5	**29**	92.0
5	87.0	**10**	91.5	**15**	92.5	**20**	92.0	**25**	93.5	**30**	92.0
6	89.5	**11**	93.0	**16**	92.0	**21**	91.5	**26**	93.0		

Table 2: Accuracy: trained with all training digits

The number of columns didn't seem to affect the result except with very few, < 8. Increasing the number of training digits didn't improve the result either. The final test was therefore done with 7500 training digits to construct the U's and 20 columns of these. The number of columns select was something in the middle of the tested span of > 10 columns. This resulted in an accuracy of: **95.44%**. The result show that the SVD is a powerful tool in data analysis and can be used for simpler image recognition tasks.

References

[1] Wikipedia. *Singular value decomposition*, 2022. https://en.wikipedia.org/wiki/Singular$_v alue_d ecomposition$.

[2] Davoud Mirzaei. Linear least squares, qr and svd, 2022.