Univerza *v Ljubljani*
Fakulteta *za računalništvo*
*in informatiko*

MATHEMATICAL MODELING

PROJECT ASSIGNMENT

# Hand-written numerals recognition

*Authors:*
Andrej Hafner
Anže Mur

*Mentors:*
as. dr. Damir Franetič
prof. dr. Nežka Mramor Kosta

June 11, 2018

# Contents

# List of Figures

# 1 Problem introduction

Handwritten numerals recognition is the ability of a computer to receive and interpret handwritten input from sources such as paper documents, photographs and other medias. Early optical character recognition (OCR) may be traced to technologies involving telegraphy and creating reading devices for the blind and from that cause we received many good OCR methods. In past decade handwritten digit recognition (and other OCR methods) has become very important task in every day life. The reason for its increasing popularity is because of its enormous set of practical applications. Hand written digit recognition helps us to solve various complex problems and saves us a lot of time. We can see some of its everyday practical uses in automatic processing of bank checks, postal zip code recognition, signatures validation and many others.

In our project assignment we had to tackle the problem with two different handwritten digit recognition methods- Least squares method and Singular-value decomposition (SVD) method.

## 2 Data collecting

In order to use those two methods we had to collect some data. We needed a big enough data set so the recognition would work better and we would have enough testing samples.We created a template and pass it around to our acquaintances. The template format is very simple - we have ten squares and atop of every square there is a digit (from 0-9) a user should write to the center of the square.

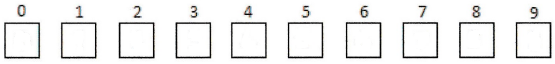


Figure 1: Data collection template.

Our data collection was very successful and we gathered samples from 80 people (thats 800 handwritten digits). We soon realized that our data was not formated correctly. People have different style of writing so some of them didn't wrote the digits in the center of the square. That problem would significantly decrease the precision of the algorithms. Therefore we wrote a program in C++ using OpenCV libraries. It detects a square and cuts it out of an image and then computes the mass center of it. Then it creates a new square which is four times bigger than the original and pastes the the cut out square in the middle of the new square so that the mass center and the center of the big square are aligned. When the number is in place program computes the new angles of the square around the number and cuts it out of the big square. Now we have a perfectly aligned handwritten data piece.

We divided our data in two sets - training set and testing set. Most of the data (90%) is in our training set and the rest (10%) is in our testing set.

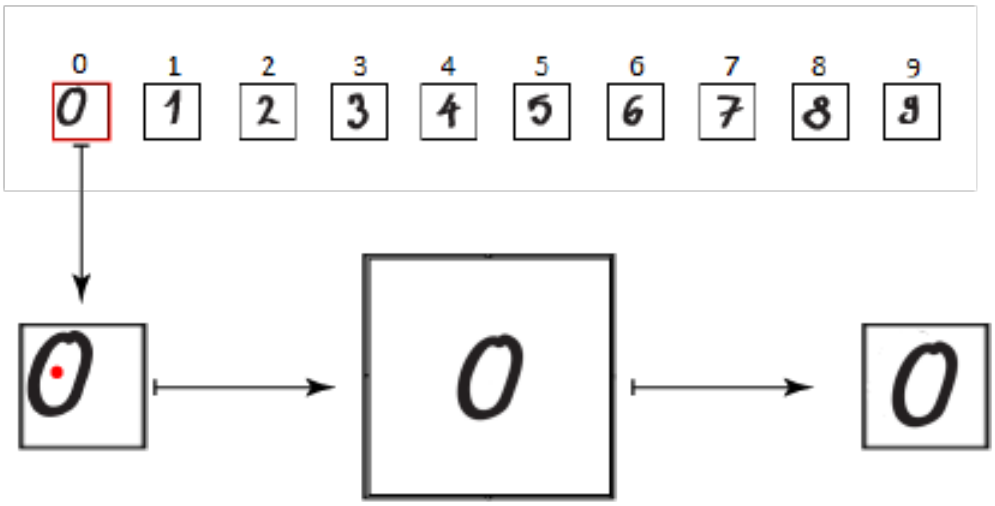


Figure 2: Process diagram of our C++ program.

# 3 Used methods

For our assignment we used two different methods for numeral handwritten recognition.

## 3.1 Least squares method

### About the method

The method of least squares is a standard approach in regression analysis. We can approximate the solution of overdetermined systems that consists of sets of equations in which there are more equations than unknowns. Expression "least squares" means that the overall solution minimizes the sum of the squares and the errors made in the results of every single equation.

**Handwritten digit detection with least squares method:**

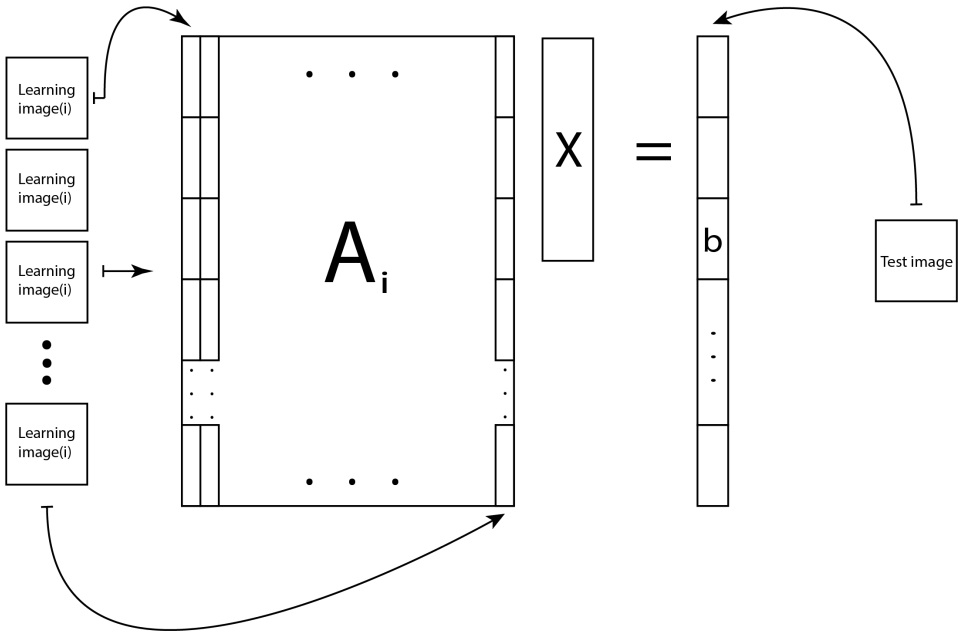


Figure 3: Process diagram of least squares method matrix preparation.

Let's say that we want know if one of our images from test set represents the digit $i$ (where $i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ ).We take all of the images that represents digit $i$ from our learning set and we put them in a vector - we take every column from an image and we put them in the right order one under another. We do this for every image and we take those vectors and put them next to one another (as we can see on the diagram above) to construct a matrix $A_i$.

Now lets say that $b$ is a vector that represents our test image (we construct vector $b$ in the same way that we construct image vectors). Now we have a system $A_i x = b$. In general this system doesn't have a solution but we can solve it with the use of minimal norm - so we get the best approximation of the solution. We create the matrices $A_i$ for every digit from our learning set and we compute the solution of the system $x_i = A_i^+ b$ for every $i$. Then we choose the $i$ for which the value of $||b - A_i x_i||$ is minimal. The $i$ that we get is the digit that we recognized.

With the use of the least squares method, we achieved 94% precision. You can see the implementation in file $digitRecognitionLeastSquares.m$.

## 3.2 Singular-value decomposition (SVD) method

### About the method

In linear algebra, the singular-value decomposition is a factorization of a real or complex matrix. We define SVD of a matrix **A** as:

$$A = USV^T$$

Where the matrices:

- $U$: has columns that are the left singular vectors
- $S$: has singular values and is diagonal
- $V^T$: has rows that are the right singular vectors

The SVD represents the original data in a coordinate system where the covariance matrix is diagonal.

### Handwritten digit detection with SVD method:

The least squares method is usually not really efficient and it takes allot of time to compute the answer. So it is better to use the SVD method. We prepare the data in the same way as before so that:

- $A_i$ is our matrix built from learning images (for $i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$),
- $b$ is our input/test image as a vector,
- and $x_i$ is solution we are looking for.

After we prepare our data we compute the singular value decomposition of the matrix $A_i$:

$$A_i = U_i S_i V_i^T$$

And then we search for solutions of the systems

$$U_i S_i y_i = b; \quad \text{for every } i$$

where:

$$y_i = V_i^T x_i$$

We get the solution in a similar way as before as we choose the $i$ for which the value of $||U_i^T b - S_i y_i||$ is minimal.

With the use of the SVD method, we achieved 96% precision. You can see the implementation in file *digitRecognitionSvd.m*.

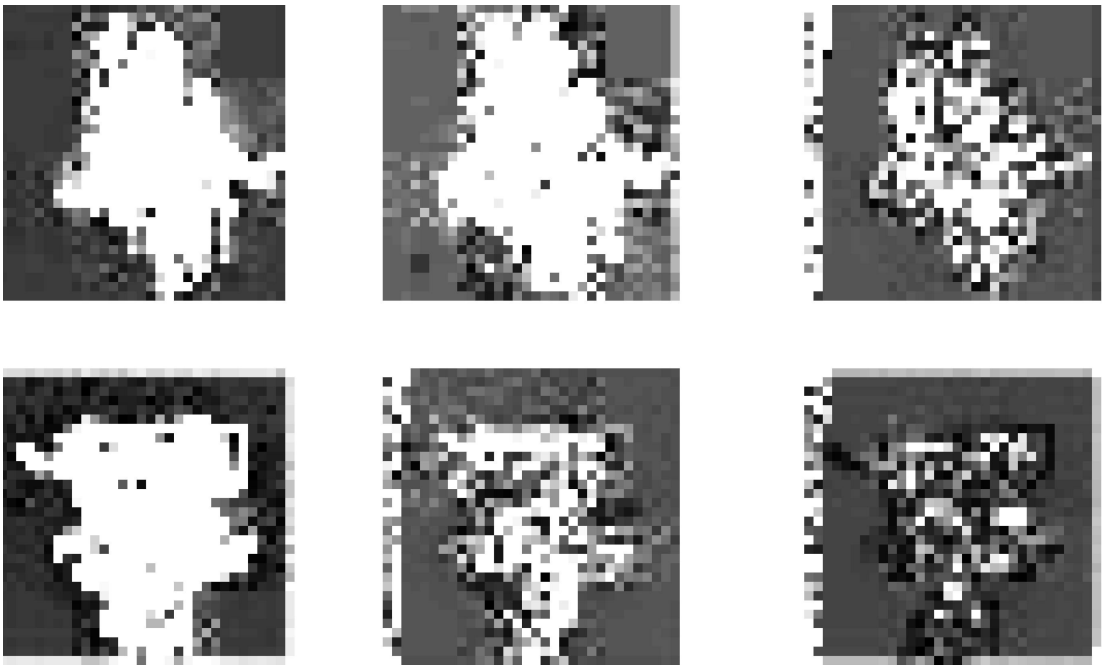

Figure 4: Left singular vectors as images. In the first row 5th, 200th and 800th left singular vector of number 4, and in the second row 5th, 200th and 800th left singular vector of number 7.

# 4 Theoretical basis

We use the singular value decomposition to transform matrix $A_i$ into $U_i S_i V_i^T$. Digit recognition is the performed by solving finding solutions for systems of equations $U_i S_i y_i = b$, where $y_i = V^T x_i$ and $b$ is the column vector of an image with the digit we are trying to recognize. The $U_i^T b - S_i y_i$ with the minimal norm is the solution and the digit recognized.

**Show that** $||x_i|| = ||y_i||$
We use SVD on matrix $A$:

$$A^{k \times p} = U^{k \times k} S^{k \times p} (V^T)^{p \times p}$$

Where $p$ is the number of learning examples and $k$ equals 1024 since our learning examples images were of size $32 \times 32$ and when we stack their columns one on top of another, we get a vector with dimension $1024 \times 1$. Then we stack these vectors next to eachother to create the matrix $A^{k \times p}$.

Because $y_i = V^T x_i$ we get:

$$||x_i^{p \times 1}|| = ||(V^T)^{p \times p} x_i^{p \times 1}||$$

where $V^T$ (and also $U$) is a real unitary matrix, of which columns form a set of *orthonormal* vectors. A set of vectors is orthonormal if they are *orthogonal* to each other and are *unit* vectors. An unit vector has a length of 1 and if you multiply a vector by an unit vector, it will only change it's direction, but not it's *norm* (length). By the definition of a unitary matrix, the same is true for conjugate transposes of $V^T$ and $U$. This means that columns of $V$, $V^T$, $U$ and $U^T$ all form a set of orthonormal vectors.

We can conclude that multiplying $x_i$ by $V^T$ which results in $y_i$ will only change it's direction and not it's norm. Therefore it holds that $||x_i|| = ||y_i||$.

**Show that** $||b - A_i x_i|| = ||U_i^T b - S_i y_i||$
If we remove the norms from both sides of equation we get:

$$b^{k\times 1} - A_i^{k\times p} x_i^{p\times 1} = (U_i^T)^{k\times k} b^{k\times 1} - S_i^{k\times p} y_i^{p\times 1}$$

where $k$ and $p$ have the same meaning as in the previous proof. Then we multiply the equation from the left side with $U_i^{k\times k}$ and use $y_i = V^T x_i$.

$$U_i^{k\times k} b^{k\times 1} - U_i^{k\times k} A_i^{k\times p} x_i^{p\times 1} = U_i^{k\times k} (U_i^T)^{k\times k} b^{k\times 1} - U_i^{k\times k} S_i^{k\times p} (V_i^T)^{p\times p} x_i^{p\times 1}$$

Because $U$ is a unitary matrix, multiplying it by it's conjugate transpose yields an *identity* matrix ($UU^T = I$). Multiplying a vector by an identity matrix will yield an identical vector. By removing the dimensions, using $A_i = U_i S_i V_i^T$ and the distributivity of matrix multiplication we can simplify the equation to:

$$U_i(b - A_i x_i) = b - A_i x_i$$

The matrix $U$ has columns that form a set of orthonormal vectors (further explanation in the previous proof). Multiplying a vector by $U$ will only change it's direction, but not the norm. This means that multiplying $b - A_i x_i$ by $U_i$ will not change it's norm. If we introduce back the norm on both sides we have shown that $||b - A_i x_i|| = ||U_i^T b - S_i y_i||$.

$$||U_i(b - A_i x_i)|| = ||b - A_i x_i||$$

# 5 Conclusion

We used SVD to generate matrices $U$ $S$ and $V^T$. For recognition we can only store matrices $U$ and $S$. Because of the way that SVD works, the first $n$ singular values and left singular vectors hold most of the "patterns" or information needed for recognition. Because of this we can use only first $n$ singular values and left singular vectors for recognition, and still keep good accuracy at recognition. This means that we can only store those first $n$ values for each number. By doing this we reduce the size of the files (a file with U and V matrix for 80 learning examples can be as big as 10MB) and accelerate the calculation since there is less multiplication. We achieved best accuracy of 96% when using 11 singular values. Using a lot of singular values can actually reduce the accuracy, since there is a lot of noise in the higher left singular vectors.

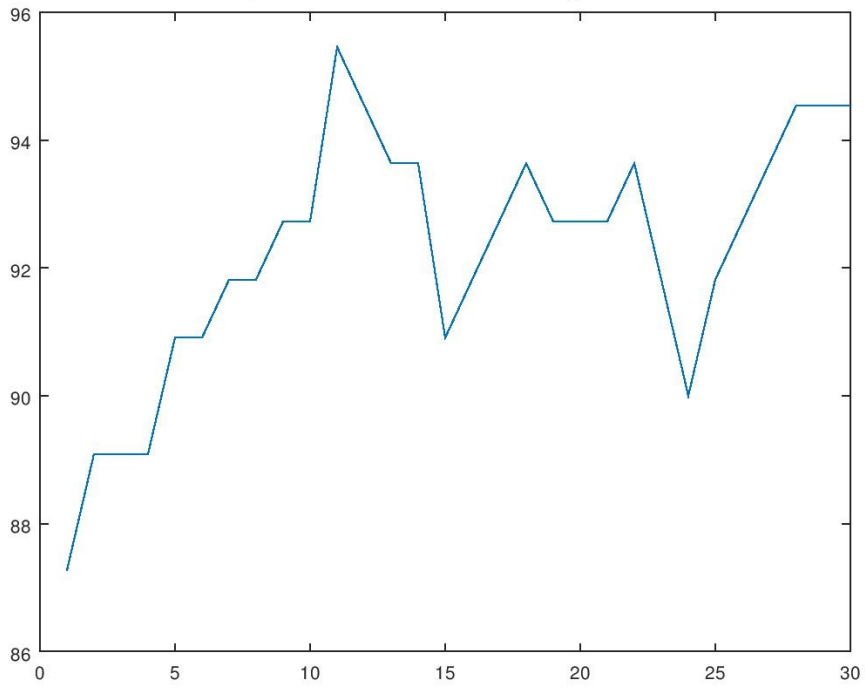


Figure 5: Precision of the forecast depending on the number of singular values.