Norges teknisk-naturvitenskapelige universitet

Estimation, Detection and Classification

TTT4275

Classification Project
Classifying Iris Flowers and Handwritten Numbers

Solenn Anne Sandrine Guironnet
Ingebrigt Stamnes Reinsborg

Abstract

This report details the work done in the Classification Project for the course "TTT4275 Estimation Detection and Classification" for the spring semester of 2021.

Classifying iris flowers can be done by using a linear classifier depending on 4 features. With the right training, it is possible to achieve an error rate equal to 3,33%. However, this rate tends to go higher the more we are removing features. It depends on the separability of them. A feature linearly separable from the other is going to highly affect the error rate because it is the parameter that allows us to be the most confident about our classification.

Identifying 10000 handwritten numbers from the MNIST data set can be implemented using kNN-Classifiers with and without clustering. All the different classifiers are going to give different performances and the decision of choosing one classifier in particular depends on the needs of the user. The NN-classifier without use of clustering achieves the lowest error rate but takes a long time to run. If the user wants something that is fast it may be better to choose a 1NN-classifier with clustered data. A 7NN-classifier could be a viable solution to similar tasks, but in this assignment was found to have more errors and took more time than the simpler 1NN-classifier.

All code for this project can be found in this repository on GitHub:
https://github.com/IngebrigtSR/TTT4275-EDC

Table of Contents

Abstract **1**

1 Introduction **3**

2 Theory **3**

2.1 Linear classifier . . . 3

2.2 Linear separability . . . 3

2.3 Confusion matrix and error rate . . . 4

2.4 k-Nearest Neighbour Classifier . . . 4

2.5 Clustering . . . 4

3 Presentation of the tasks **5**

3.1 Iris task . . . 5

3.2 MNIST task . . . 5

4 Implementation and results for the iris task **6**

4.1 Implementation . . . 6

4.2 Results . . . 6

4.3 Discussion . . . 7

5 Implementation and results of the MNIST task **8**

5.1 Implementation . . . 8

5.2 Results . . . 8

5.3 Discussion . . . 16

6 Conclusion **16**

References **17**

A Matlab code for training a linear classifier **18**

B Matlab code for computing a confusion matrix **19**

C Matlab code for computing the error rate **19**

D Matlab code for the kNN classifiers for task 2 **20**

E Matlab code for performing clustering **24**

F Matlab code for plotting misclassified and correctly classified data **25**

1 Introduction

Nowadays, machine learning is used in everyday life. When you wake up in the morning and unlock your smartphone by just putting your face in front of it, is a result of machine learning. Using machine learning techniques has already proven its efficiency.
The different classification techniques exposed in this report represent solid basis to understand how it works.
The aim of this project is to apply theoretical knowledge on concrete problems: classifying flowers and recognizing handwritten numbers. Those two tasks will use different classifiers, a linear classifier and the nearest neighbour algorithm. First of all, the necessary theory needed to understand the project is presented in section 2 and the tasks that should be done are detailed in the section 3. All the implementations in Matlab and the results are developed in sections 4 and 5. The conclusion can be read in section 6.

2 Theory

Classification is the core of deep learning. In fact, the goal of classification is to find similarities between a set of data and put the ones which are close to each other into a group which is called "class". There are a lot of different techniques that exist in this field [1] but, in this part, just the ones which are used in the project are going to be presented.

2.1 Linear classifier

We talk about a linear classifier when the classification is done using a linear function of the inputs. This is especially efficient for problems which are linearly separable. As any other classification problem, there are some measures that should be defined: (in the rest of this section, C is going to designate the number of classes and D the number of features)

- W, a CxD matrix which contains the weights of the features for all the classes
- MSE, the minimum square error which is defined as follows
$$MSE = \frac{1}{2}\sum_{k=1}^{N}(g_k - t_k)^T(g_k - t_k) \quad (2.1)$$
- The gradient of MSE $\nabla_W MSE$ which is equal to
$$\nabla_W MSE = \sum_{k=1}^{N}[(g_k - t_k) \circ g_k \circ (1 - g_k)]x_k^T \quad (2.2)$$
The notation $\circ$ is the element wise multiplication. This one is going to get used to compute W.
- The discriminant function $g_i(x)$, which is an array of C elements and which indicates the probability to belong to each class. For example, $g_1(x)$ is the probability that the sample x belongs to the first class. In the linear classifier presented in this project, the discriminant is a sigmoid.
$$g_{i,k} = \frac{1}{1 + e^{-Wx_k}} \qquad i \in [1; C] \quad (2.3)$$

With all these notations, it is now possible to define W,

$$W(m) = W(m-1) - \alpha\nabla_W MSE \quad (2.4)$$

W(m) is the value of W at the mth iteration and α is the step factor. For the iris task, and after running some tests, we decided to choose $\alpha = 0.0075$. The computation of W stops when the difference between W(m) and W(m-1) is small enough.

2.2 Linear separability

The linear separability for several classes and two features lies on the possibility to split the classes by drawing a line.

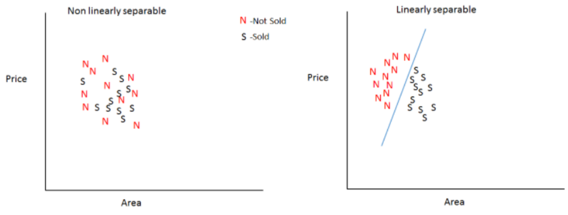


Figure 2.1: Cases of non-linear separability and linear separability (source [2])

On the left, the two are mixed up, it is impossible to split them into two non overlapping packets, contrary to the right.

2.3 Confusion matrix and error rate

The confusion matrix and the error rate are indicators that measure the efficiency of a classifier. They measure the amount of False Negatives (FN) and False Positives (FP). An FN is when you say that a sample does not belong to a certain class when it actually does. FP is when you say it belongs to a class when, in reality, it does not. When the guess is the right one, we talk about True Negatives (TN) or True Positives (TP).

The confusion matrix can be defined as

Confusion matrix			
Classified → True label ↓	Class 1	Class 2	Class 3
Class 1	TP1	FN12	FN13
Class 2	FN21	TP2	FN23
Class 3	FN31	FN32	TP3

Tab 2.1: Elements of a confusion matrix

The error rate is given by the formula

$$Err = \frac{FN + FP}{FN + FP + TF + TP} \tag{2.5}$$

2.4 k-Nearest Neighbour Classifier

The Nearest Neighbour (NN) Classifier uses a similarity metric which is most often the euclidean distance between test and training vectors. NN's decision rule is that it selects the class of data points closest to the test data. In the slightly expanded method of k-Nearest Neighbours (kNN) [3], the class is selected by a vote of the k nearest data. (NN is kNN with k=1). Normally, k should be an odd number to avoid even splits and ties in the vote. If the object to be classified has four neighbours of type A and three of type B, the classifier (7NN in this case) would classify the object as type A. This type of classifier is a supervised method, meaning it requires labeled data.[3]

2.5 Clustering

Clustering is a way of reducing the data set by creating templates consisting of several data points in one "Cluster". This greatly reduces the time it takes to sufficiently classify objects. Clustering can be done in several different ways, for example, the k-means algorithm clusters data points in such a way that each cluster is represented by a centroid/mean of the cluster.

3 Presentation of the tasks

3.1 Iris task

The first task consists in classifying different iris into their right specie. Indeed, different varieties of iris flowers exist: the Setosa (class 1), the Versicolor (class 2) and the Virginica (class 3).

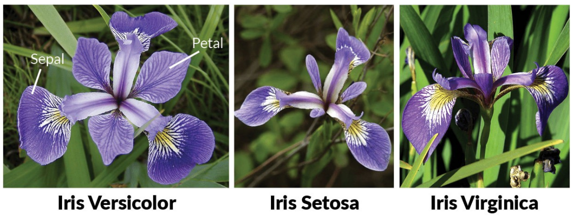


Figure 3.1: Iris varieties (source [4])

What make the difference between all of these flowers are the dimensions of their sepals and petals. Thus, the features used for the classification are:

- Sepal length (in cm)
- Sepal width (in cm)
- Petal length (in cm)
- Petal width (in cm)

Each sample is made of 5 parameters which are the 4 features above and the number of the class it belongs to. In order to recognize which specie corresponds to a certain sample, the first step is to train a linear classifier by giving it some training sets. These sets are going to get used to change the weights of the features and they should not be confused with the test sets that enable the user to know the efficiency of his system.

Another interesting aspect is the study of the features themselves. In fact, some of them may be irrelevant in the training because they are too close to each other That is the study of the linear separability which is the second part of this task.

3.2 MNIST task

The second task in this project is to create k Nearest Neighbour-based classifiers to identify handwritten numbers from the MNIST dataset. The dataset consists of 60 000 training examples and 10 000 test examples, where each image is an 8-bit grey scale of size 28x28 pixels depicting a handwritten number between 0-9.

In the first part we were tasked with creating a NN-classifier that used the Euclidian distance, and then find the confusion matrix and error rate for the test set. We were also asked to print some misclassified and correctly classified images and comment on what we saw.

In the second part of this task we were asked to implement clustering to produce smaller sets of templates for each class. We should then perform clustering of each class' 6000 training vectors into M=64 clusters.

We were then tasked with using the NN-classifier with clustering, designing a kNN-classifier with k=7, and also find the confusion matrix and error rate for these so that we could compare the three different classifiers.

4 Implementation and results for the iris task

Based on the theory exposed in the section 2, this part describes the implementation that we chose for the iris task, the results obtained and an interpretation of them.

4.1 Implementation

The implementation can be divided into 3 parts:

- Selecting the training set and the test set
- Training
- Computing the confusion matrices and the error rates for the training and the test

Most of the time, the training set was composed of the first 30 samples and the test set of the last 20 ones. Once the right samples were taken from each class, they were stored into a big array, one for the training and one for the test. The first class corresponds to the first 4 columns, the second class to the next 4 columns and so on.

Regarding the training in itself, the main issue was to update the weights for each class. At the same time, it was necessary to compute the gradient of the MSE and the matrix W, which contained all the updated weights. There is a small difference between the theory we exposed in the section 2 and our implementation. In fact, even if the training should stop when the difference between the current W and the previous W is small enough we decided to work with a simple iteration loop. We assume that after a certain amount of iterations, for example 1000, the difference is going to be small enough. During each iteration, all the samples from the training set were processed.

Then, some tools were implemented in order to check the quality of the chosen training: the confusion matrix and the error rate. To construct the confusion matrix C, the key point was to know how to identify to which class a sample belonged to. The 1D-array g, which corresponds to the output of the sigmoid function, contains the information. In fact, the predicted class is the index i of the maximum value of the array. Then, for a sample which is supposed to belong to a class c, the confusion matrix will become $C_{c,i} = C_{c,i} + 1$. To obtain the whole matrix, this should be done for all the training or the test samples.

The error rate is easy to get. It is the sum of all the coefficients which are not on the diagonal divided by the length of training/test set multiplied by the number of classes.

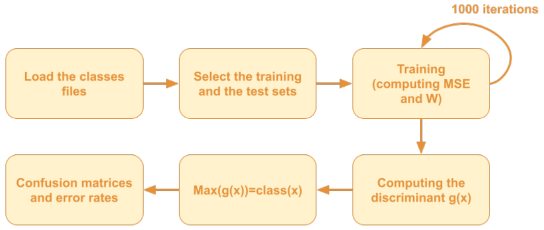

1000 iterations
Load the classes files
Select the training and the test sets
Training (computing MSE and W)
Confusion matrices and error rates
Max(g(x))=class(x)
Computing the discriminant g(x)

Figure 4.1: *Flow-diagram of the iris task*

The second part of the task was to determine if all the features were really necessary and trust full in regard to linear separability. Histograms and scatter plots can be used to observe the distribution of the different features for each class.

4.2 Results

First, it was just the normal training and test sets with 4 features which were checked. And then, the sets got changed. Their length did not change but, instead of training on the first 30 samples, the training was on the last 30 ones and the same goes for the test. The confusion matrices for those two cases can be found on the figure 4.2.

$$C_{tr30-20} = \begin{pmatrix} 30 & 0 & 0 \\ 0 & 28 & 2 \\ 0 & 1 & 29 \end{pmatrix} \quad C_{tst30-20} = \begin{pmatrix} 20 & 0 & 0 \\ 0 & 18 & 2 \\ 0 & 0 & 20 \end{pmatrix}$$

$$C_{tr20-30} = \begin{pmatrix} 30 & 0 & 0 \\ 0 & 27 & 3 \\ 0 & 2 & 28 \end{pmatrix} \quad C_{tst20-30} = \begin{pmatrix} 20 & 0 & 0 \\ 0 & 19 & 1 \\ 0 & 0 & 20 \end{pmatrix}$$

Figure 4.2: *Confusion matrices for the training and the test for 30 training-20 test and 20 test-30 training*

And for the error rates, we obtained $Err_{train30-20} = Err_{test30-20} = 0.0333$ and $Err_{train20-30} = 0.0556$; $Err_{test20-30} = 0.0167$.

Now, for the second part of the task, we created some histograms to see which feature overlaps the most with another one and removed it.

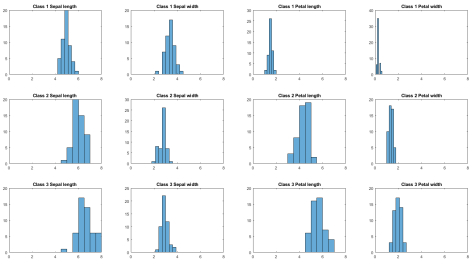


Figure 4.3: *Histograms of the distribution of the 4 features for each class*

The feature which seems to be the most confusing is the second one, the sepal width. If we consider only the 3 other features, we obtain exactly the same results as the ones from the figure 4.2 for the training set of the first 30 samples. The next feature which we can remove is the sepal length and this time, the results are not the same.

$$. \; C_{tr2feat} = \begin{pmatrix} 30 & 0 & 0 \\ 0 & 20 & 10 \\ 0 & 2 & 28 \end{pmatrix} \; C_{tst2feat} = \begin{pmatrix} 20 & 0 & 0 \\ 0 & 17 & 3 \\ 0 & 2 & 18 \end{pmatrix}$$

Figure 4.4: *Confusion matrices for the training and the test with 2 features*

The error rates are equal to $Err_{train2feat} = 0.1333$ and $Err_{tst2feat} = 0.0833$. For the last case, with one feature, just the petal width will be kept. Like this, we obtain the matrices of the figure 4.5.

$$. \; C_{tr1feat} = \begin{pmatrix} 30 & 0 & 0 \\ 0 & 19 & 11 \\ 0 & 0 & 30 \end{pmatrix} \; C_{tst1feat} = \begin{pmatrix} 20 & 0 & 0 \\ 0 & 16 & 4 \\ 0 & 1 & 19 \end{pmatrix}$$

Figure 4.5: *Confusion matrices for the training and the test with 1 feature*

The error rates are $Err_{train1feat} = 0.1222$ and $Err_{tst1feat} = 0.0833$. Another way to observe the linear separability of the features is to compare one feature to another one in a scatter plot. They are all presented in the figure 4.6.

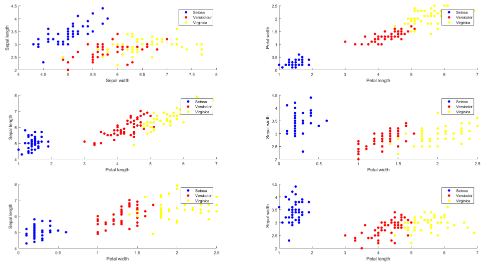


Figure 4.6: *Scatter plots of the distribution of the 4 features for each class*

4.3 Discussion

The importance of the choice of the samples for the training and the test sets is the first point which is going to be discussed.
When the training or the test set is composed of the last samples, the corresponding error rate is worse than with the first samples. There are no changes regarding the classification of the first class, the Setosa, it is still perfectly recognized. What is problematic is the way to differentiate the other classes, the Versicolor and the Virginica. When looking at the scatter plots from the figure 4.6, it seems that those two are overlapping for all the features, or at least a little bit for all of them. The reason of the difference in the error rate when choosing the sets can be that the last samples might have features for those two classes which are closer to each other than the first samples.

The second aspect which is interesting to highlight is the separability of the features. In general, the more separable the features are, the more efficient the classifier is. Here, it is the linear separability which is going to be studied. As explained in the theoretical part, two classes are linearly separable for a specific feature if each class has its specific area in the scatter plot and if the areas can be separated by a line. By observing the different graphs in the figure 4.6 , it appears that for the first class, the Setosa, all of the features are linearly separable from the other classes. However, it is not the case for the others.

If we refer to both the figures 4.3 and 4.6, some features, like for example the petal length, are nearly linearly separable, but the sepal width is far from being separable. That is the reason why it was the first feature to be removed.

The impact of removing some features affects the confusions matrices and the error rates. The numbers of false negatives or false positives do not change for the Setosa. The source of the new errors is the classification of Versicolor and Virginica. Those two indicators should be interpreted carefully. In fact, the sets that we are working on are small. It could be a good idea to work with more samples to have results even more accurate.

5 Implementation and results of the MNIST task

This section details the implementation and results from the task described in section 3.2.

5.1 Implementation

We were given a file containing the MNIST data which had to be run and loaded into the MatLab workspace before we could test our own algorithms and programs. There has been added some logic to ensure that this data is loaded before anything else is run, making it easier to clone, run on different computers, and ensure general ease of use.

The first task asked us to design a NN-classifier. This was made by creating a script calculating the Euclidian distance between a test vector i and all the training vectors through a for-loop. This gives us a 60 000-by-1 column vector containing the distances. The smallest distance is chosen, and a guess on the test vector is chosen as the corresponding training label. For the confusion matrix and error rates, we have simply used the built in MatLab-function plotconfusion(). Further information on the functionality of this function can be found by writing "help plotconfusion" in a MatLab-terminal. The code can be viewed in D.1

We were also tasked with plotting some misclassified and correctly classified examples, these can be seen in 5.1 and 5.2. The code for these tasks can be seen in F.1 and F.2. The code identifies misclassified/correctly classified images by running through the list of guesses and knowns, and checking if they are misclassified/correctly classified, then giving a list of "IDs" which is then used to find the images to be plotted. We simply decided to plot one example of each number. To correctly plot the images we had to make a helper function F.3. It takes the 1-by-784 image vector and converts it into a 28-by-28 matrix which is then rotated and made into an image.

In the second part of the task, we were to cluster the data. This was done by splitting the training images into cells for each of the ten numbers, which are then made into 64 clusters. The assignment advised us to use the kmeans()-function, this returns the centroid for each cluster, which then serve as templates for the numbers. The clustered training set consists of a matrix containing these templates. The code for the clustering can be read in E.1

We lastly created two versions of the first NN-classifier; the first modified to use the clustered training set, and the other was in addition modified to be a 7NN-Classifier by having it find the seven smallest values in the distance vector, then arranging a vote based on what number these values' indices indicate. This is done by counting how many of each number there is and returning the one with most occurrences. The code for both of these NN-Classifiers can be read in D.2 and D.3.

5.2 Results

The confusion matrix for the first NN-Classifier using no clustering can be seen in 5.3 and the error rate for the test set can be read as 3.1%.

Misclassified images can be seen in 5.1 and correctly classified images can be seen in 5.2.

The clustering was performed and both the modified 1NN-Classifier and the 7NN-Classifier were run on the clustered data. The confusion matrix for the 1NN-Classifier can be seen in 5.4, the error rate was 4.9%. The confusion matrix for the 7NN-Classifier can be seen in 5.5, the error rate was 6.2%.

The execution time for each classifier can be seen in 5.6, 5.7 and 5.8.

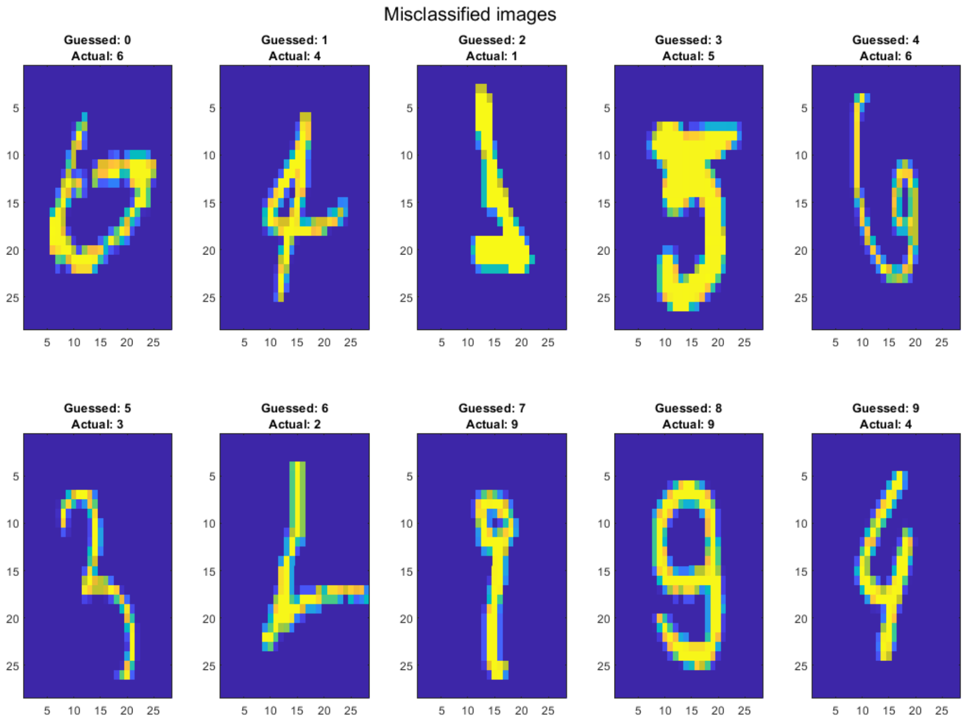


Figure 5.1: *Misclassified images*

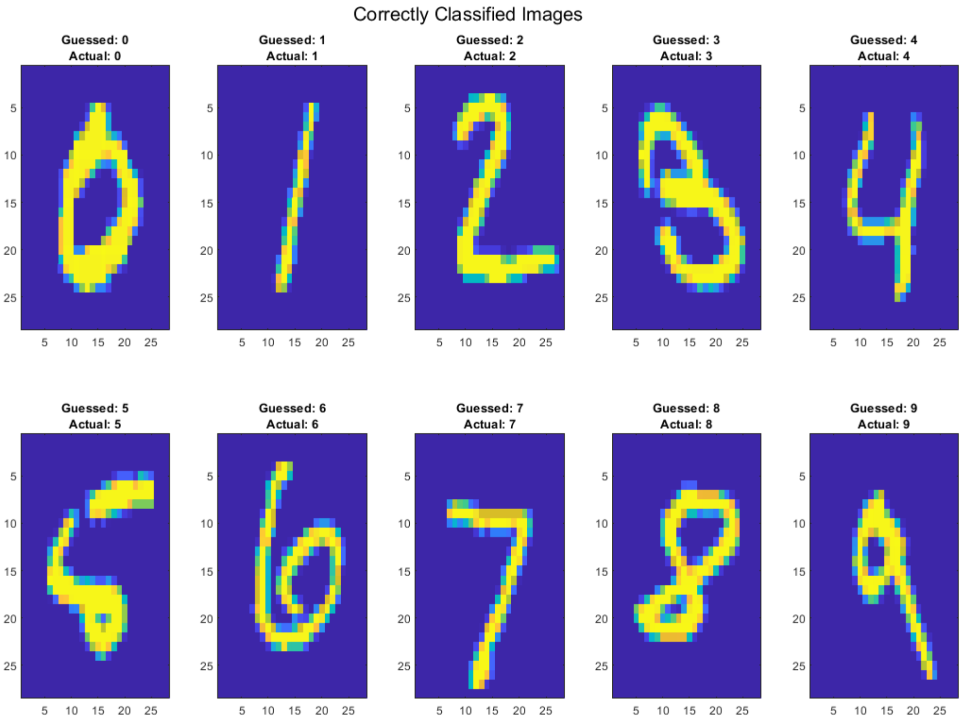
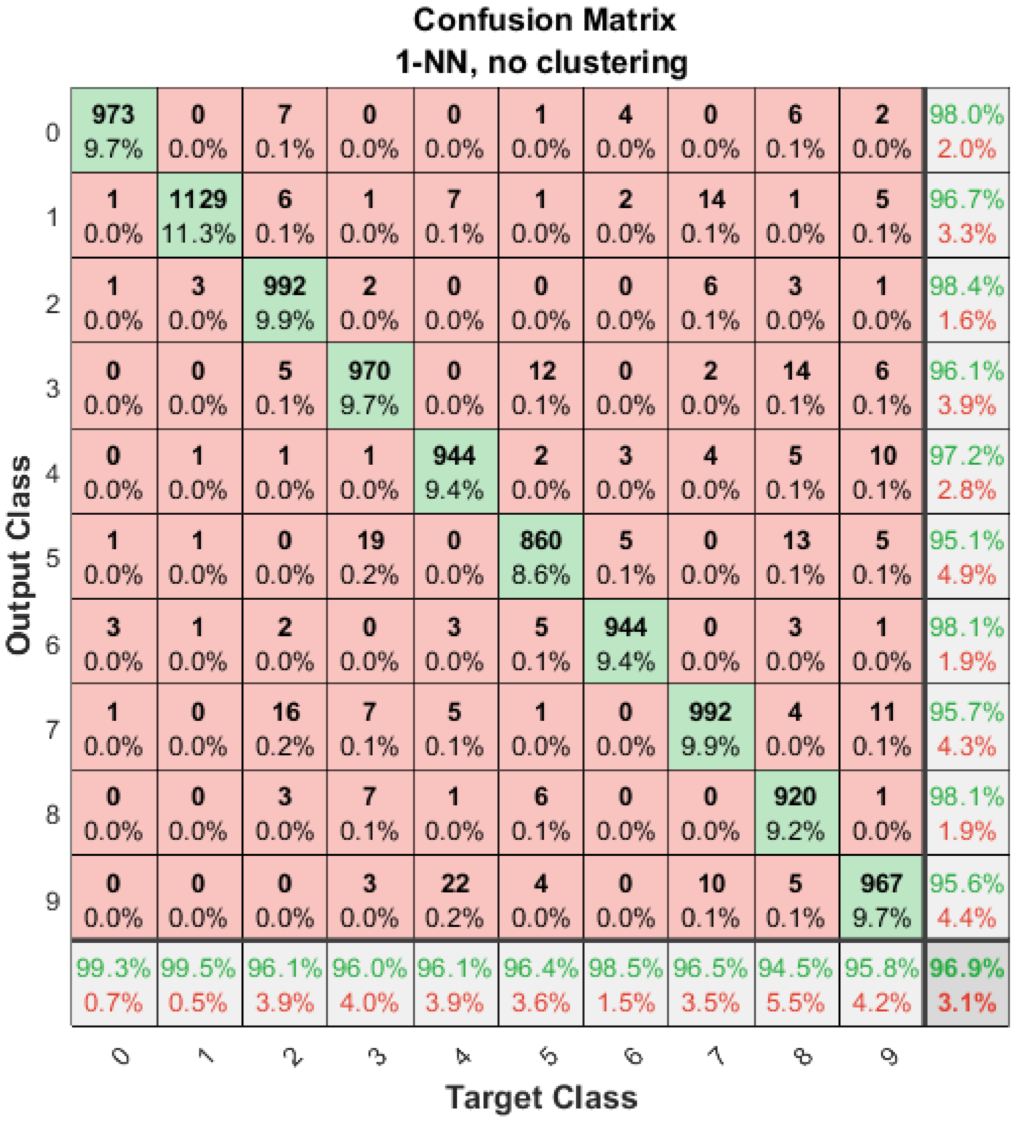


Figure 5.2: *Correctly classified images*

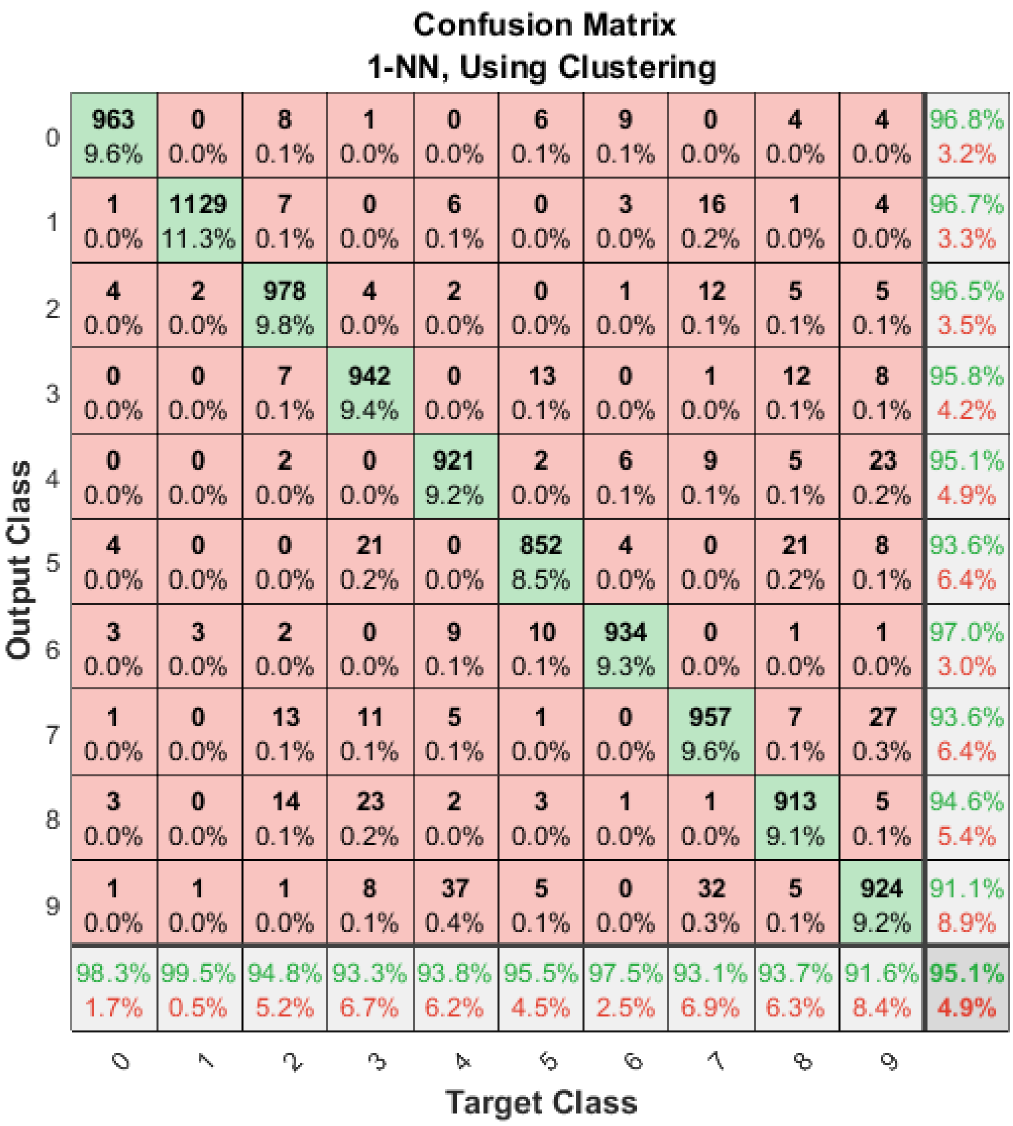


Confusion Matrix
1-NN, no clustering

Output Class	0	1	2	3	4	5	6	7	8	9	
0	973 9.7%	0 0.0%	7 0.1%	0 0.0%	0 0.0%	1 0.0%	4 0.0%	0 0.0%	6 0.1%	2 0.0%	98.0% 2.0%
1	1 0.0%	1129 11.3%	6 0.1%	1 0.0%	7 0.1%	1 0.0%	2 0.0%	14 0.1%	1 0.0%	5 0.1%	96.7% 3.3%
2	1 0.0%	3 0.0%	992 9.9%	2 0.0%	0 0.0%	0 0.0%	0 0.0%	6 0.1%	3 0.0%	1 0.0%	98.4% 1.6%
3	0 0.0%	0 0.0%	5 0.1%	970 9.7%	0 0.0%	12 0.1%	0 0.0%	2 0.0%	14 0.1%	6 0.1%	96.1% 3.9%
4	0 0.0%	1 0.0%	1 0.0%	1 0.0%	944 9.4%	2 0.0%	3 0.0%	4 0.0%	5 0.1%	10 0.1%	97.2% 2.8%
5	1 0.0%	1 0.0%	0 0.0%	19 0.2%	0 0.0%	860 8.6%	5 0.1%	0 0.0%	13 0.1%	5 0.1%	95.1% 4.9%
6	3 0.0%	1 0.0%	2 0.0%	0 0.0%	3 0.0%	5 0.1%	944 9.4%	0 0.0%	3 0.0%	1 0.0%	98.1% 1.9%
7	1 0.0%	0 0.0%	16 0.2%	7 0.1%	5 0.1%	1 0.0%	0 0.0%	992 9.9%	4 0.0%	11 0.1%	95.7% 4.3%
8	0 0.0%	0 0.0%	3 0.0%	7 0.1%	1 0.0%	6 0.1%	0 0.0%	0 0.0%	920 9.2%	1 0.0%	98.1% 1.9%
9	0 0.0%	0 0.0%	0 0.0%	3 0.0%	22 0.2%	4 0.0%	0 0.0%	10 0.1%	5 0.1%	967 9.7%	95.6% 4.4%
	99.3% 0.7%	99.5% 0.5%	96.1% 3.9%	96.0% 4.0%	96.1% 3.9%	96.4% 3.6%	98.5% 1.5%	96.5% 3.5%	94.5% 5.5%	95.8% 4.2%	96.9% 3.1%

Target Class

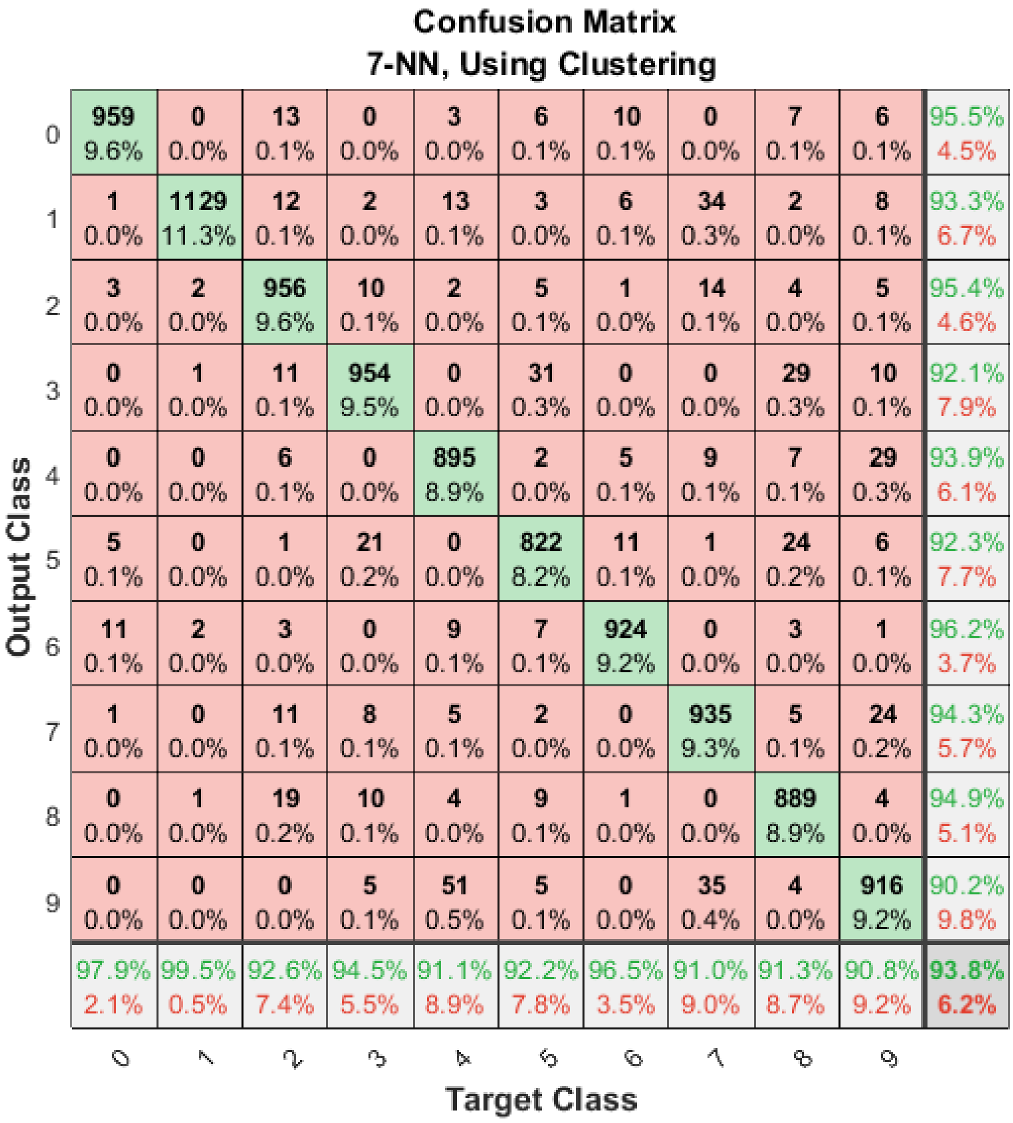
Figure 5.3: *Confusion matrix, 1-NN Classifier, no clustering*



Confusion Matrix
1-NN, Using Clustering

Output Class \ Target Class	0	1	2	3	4	5	6	7	8	9	
0	963 9.6%	0 0.0%	8 0.1%	1 0.0%	0 0.0%	6 0.1%	9 0.1%	0 0.0%	4 0.0%	4 0.0%	96.8% 3.2%
1	1 0.0%	1129 11.3%	7 0.1%	0 0.0%	6 0.1%	0 0.0%	3 0.0%	16 0.2%	1 0.0%	4 0.0%	96.7% 3.3%
2	4 0.0%	2 0.0%	978 9.8%	4 0.0%	2 0.0%	0 0.0%	1 0.0%	12 0.1%	5 0.1%	5 0.1%	96.5% 3.5%
3	0 0.0%	0 0.0%	7 0.1%	942 9.4%	0 0.0%	13 0.1%	0 0.0%	1 0.0%	12 0.1%	8 0.1%	95.8% 4.2%
4	0 0.0%	0 0.0%	2 0.0%	0 0.0%	921 9.2%	2 0.0%	6 0.1%	9 0.1%	5 0.1%	23 0.2%	95.1% 4.9%
5	4 0.0%	0 0.0%	0 0.0%	21 0.2%	0 0.0%	852 8.5%	4 0.0%	0 0.0%	21 0.2%	8 0.1%	93.6% 6.4%
6	3 0.0%	3 0.0%	2 0.0%	0 0.0%	9 0.1%	10 0.1%	934 9.3%	0 0.0%	1 0.0%	1 0.0%	97.0% 3.0%
7	1 0.0%	0 0.0%	13 0.1%	11 0.1%	5 0.1%	1 0.0%	0 0.0%	957 9.6%	7 0.1%	27 0.3%	93.6% 6.4%
8	3 0.0%	0 0.0%	14 0.1%	23 0.2%	2 0.0%	3 0.0%	1 0.0%	1 0.0%	913 9.1%	5 0.1%	94.6% 5.4%
9	1 0.0%	1 0.0%	1 0.0%	8 0.1%	37 0.4%	5 0.1%	0 0.0%	32 0.3%	5 0.1%	924 9.2%	91.1% 8.9%
	98.3% 1.7%	99.5% 0.5%	94.8% 5.2%	93.3% 6.7%	93.8% 6.2%	95.5% 4.5%	97.5% 2.5%	93.1% 6.9%	93.7% 6.3%	91.6% 8.4%	95.1% 4.9%

Figure 5.4: *Confusion matrix, 1-NN Classifier, with clustering*



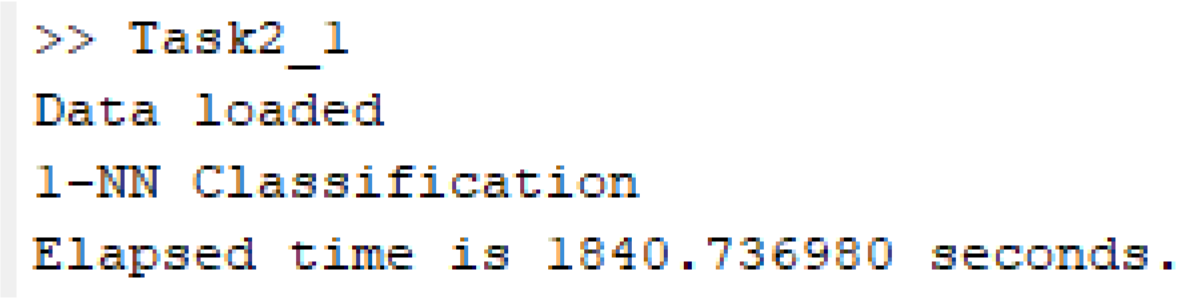
Confusion Matrix
7-NN, Using Clustering

Output Class

	0	1	2	3	4	5	6	7	8	9	
0	959 9.6%	0 0.0%	13 0.1%	0 0.0%	3 0.0%	6 0.1%	10 0.1%	0 0.0%	7 0.1%	6 0.1%	95.5% 4.5%
1	1 0.0%	1129 11.3%	12 0.1%	2 0.0%	13 0.1%	3 0.0%	6 0.1%	34 0.3%	2 0.0%	8 0.1%	93.3% 6.7%
2	3 0.0%	2 0.0%	956 9.6%	10 0.1%	2 0.0%	5 0.1%	1 0.0%	14 0.1%	4 0.0%	5 0.1%	95.4% 4.6%
3	0 0.0%	1 0.0%	11 0.1%	954 9.5%	0 0.0%	31 0.3%	0 0.0%	0 0.0%	29 0.3%	10 0.1%	92.1% 7.9%
4	0 0.0%	0 0.0%	6 0.1%	0 0.0%	895 8.9%	2 0.0%	5 0.1%	9 0.1%	7 0.1%	29 0.3%	93.9% 6.1%
5	5 0.1%	0 0.0%	1 0.0%	21 0.2%	0 0.0%	822 8.2%	11 0.1%	1 0.0%	24 0.2%	6 0.1%	92.3% 7.7%
6	11 0.1%	2 0.0%	3 0.0%	0 0.0%	9 0.1%	7 0.1%	924 9.2%	0 0.0%	3 0.0%	1 0.0%	96.2% 3.7%
7	1 0.0%	0 0.0%	11 0.1%	8 0.1%	5 0.1%	2 0.0%	0 0.0%	935 9.3%	5 0.1%	24 0.2%	94.3% 5.7%
8	0 0.0%	1 0.0%	19 0.2%	10 0.1%	4 0.0%	9 0.1%	1 0.0%	0 0.0%	889 8.9%	4 0.0%	94.9% 5.1%
9	0 0.0%	0 0.0%	0 0.0%	5 0.1%	51 0.5%	5 0.1%	0 0.0%	35 0.4%	4 0.0%	916 9.2%	90.2% 9.8%
	97.9% 2.1%	99.5% 0.5%	92.6% 7.4%	94.5% 5.5%	91.1% 8.9%	92.2% 7.8%	96.5% 3.5%	91.0% 9.0%	91.3% 8.7%	90.8% 9.2%	93.8% 6.2%

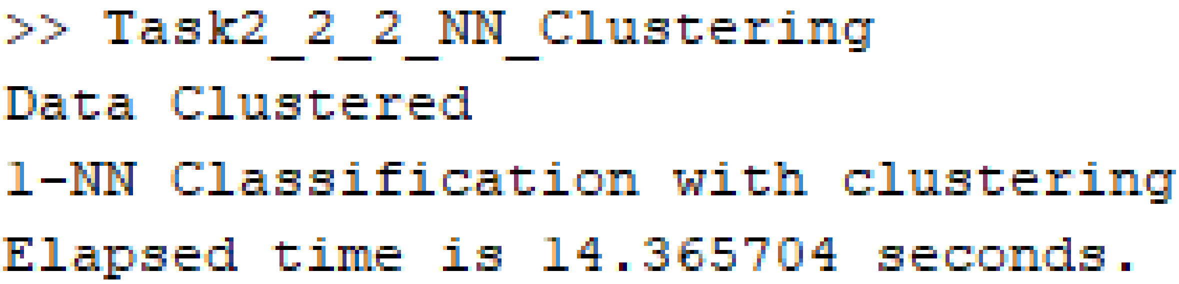
Target Class

Figure 5.5: *Confusion matrix, 7-NN Classifier, with clustering*



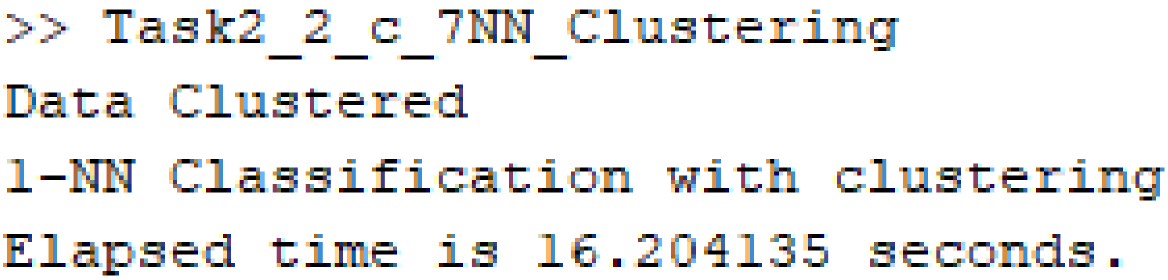
```
>> Task2_1
Data loaded
1-NN Classification
Elapsed time is 1840.736980 seconds.
```

Figure 5.6: *Time used with 1-NN Classifier without clustering*



```
>> Task2_2_2_NN_Clustering
Data Clustered
1-NN Classification with clustering
Elapsed time is 14.365704 seconds.
```

Figure 5.7: *Time used with 1-NN Classifier with clustering*



```
>> Task2_2_c_7NN_Clustering
Data Clustered
1-NN Classification with clustering
Elapsed time is 16.204135 seconds.
```

Figure 5.8: *Time used with 7-NN Classifier with clustering*

5.3 Discussion

The first NN-Classifier with no clustering got the lowest error rate at 3.1%, however it took significantly more time than the two others, it needed a bit more than 30 minutes. By using clustering we achieved a slightly higher error rate of 4.9%, but it used only slightly above 14 seconds which is 120 times better than the no-cluster version. The 7NN-Classifier got the highest error rate of the three at 6.5% and used about 16 seconds, which is still fast. It is understandable that the classifiers using clustering have a lower accuracy, since they are using training data consisting of reduced information.

By comparing these results it is clear that the 1NN-Classifier using clustering is the best performing of the three. However, if there is significant need for accuracy, the clustering should be omitted. As a curiosity, I also ran the kNN-Classifier for k=5 and k=3. 5NN gave an error rate of 6.0% and took 15.7s, 3NN gave an error rate of 5.8% and took 15.6s. This was not required from the task, but it was fun to try.

By looking at the images in 5.1 it is understandable that these were misclassified as they were. For example the "3" which was actually a 5 had a squeezed top part making it almost impossible to see where the "arch" curves. As for the rest, one can see similar structures between the guessed numbers and what they actually were. 4s and 9s oftentimes look similar in the fact that both have an enclosed area (not always for 4s) with a line going down on the right side. This can also be seen in the confusion matrices. Mistaking 4 for 9 and vice versa is one of the most common errors across all classifiers. The 9 which was classified as an 8, I would assume it was because the bottom part almost touches the upper ring segment, giving it a strong overlap with the 8-data. All classifiers seem to be making the same mistakes in varying degrees.

Most of the correctly classified images in 5.2 seem to make sense. We would have labeled them the same as the classifier.

6 Conclusion

In the first task, we have implemented a linear classifier used to classify three different iris species, the Setosa, the Versicolor and the Virginica. This permitted to highlight the importance of choosing the right sets to train and test. In fact, changing the sets can significantly affect the error rates.
Another parameter that can impact this rate is the separability of the different features which are used to characterize the different classes. Removing the features which are not separable are usually not going to change the error rate that much but, if they are separable, the error rate is going to be worse. It is understandable since we are removing what makes us able to clearly identify each class.

In the second task we have implemented kNN-Classifiers with and without clustering for the purpose of identifying 10000 handwritten numbers from the MNIST data set. The NN-classifier without use of clustering achieved the lowest error rate and took a long time to run, while the 1NN-classifier using clustering achieved the fastest time with some loss to the accuracy. The 7NN-classifier performed slightly slower and had a higher error rate compared to the 1NN with clusterring.
Depending on demands for either speed or accuracy, a 1NN-classifier with or without use of clustering can be used, while a 7NN-classifier can not be recommended based on our findings.

References

[1] Magne H. Johnsen. *"Estimation, detection and classification", Compendium Part 3: Classification, NTNU.* December 18, 2017.

[2] Rahul Kumar. *Machine learning quick reference, "Linear Separability".* January 2019. URL: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781788830577/2/ch02lvl1sec26/linear-separability.

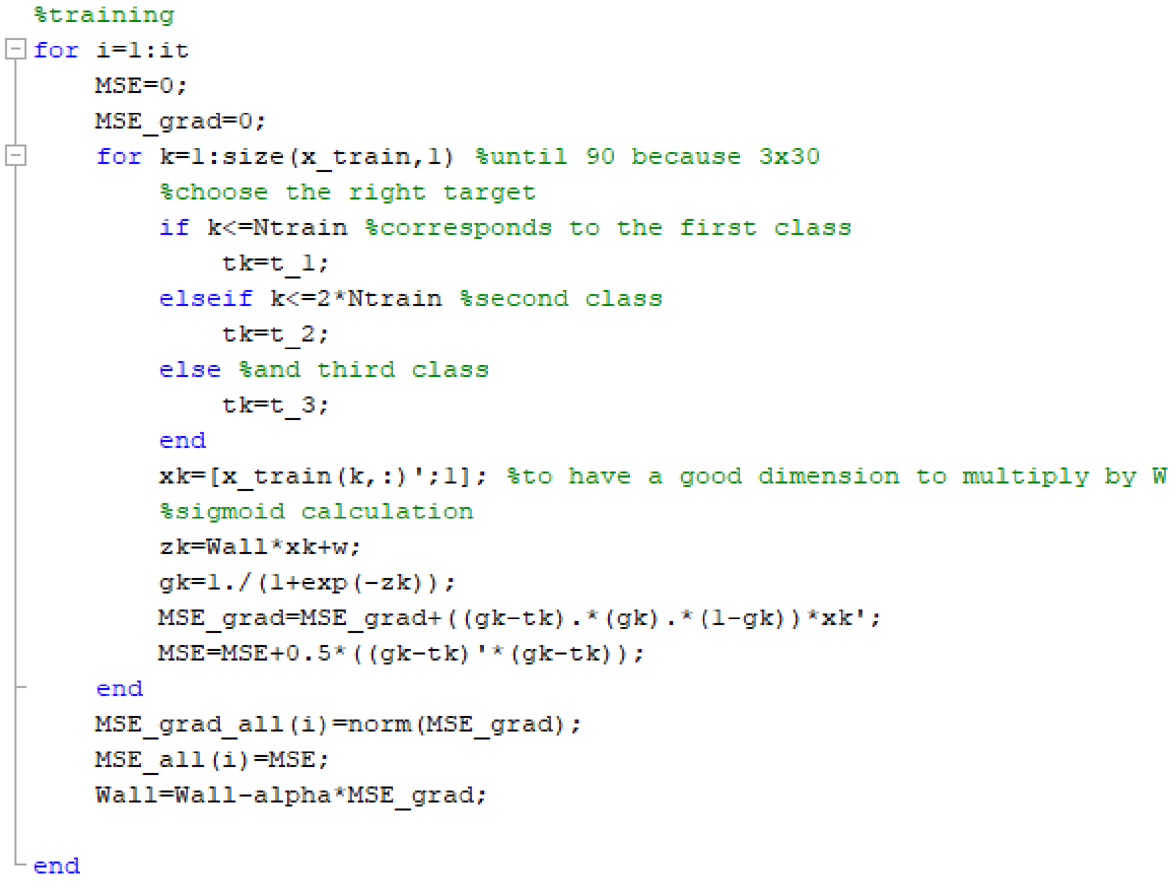
[3] Onel Harrison. *Machine Learning Basics with the K-Nearest Neighbors Algorithm.* September 10, 2018. URL: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

[4] Hitesh Gupta. *Prepare Your First Model In Machine Learning In Few Steps!* January 08, 2021. URL: https://dockship.io/articles/5ff57af32e6d744ab77c5fa8/prepare-your-first-model-in-machine-learning-in-few-steps!.

A Matlab code for training a linear classifier

The code A is the one used to train the classifier based on the formulas given in the section 2.1.

Also, a quick note on the code: everything can be found in our repository on github in case anyone reading this would want to run the code for themselves: https://github.com/IngebrigtSR/TTT4275-EDC

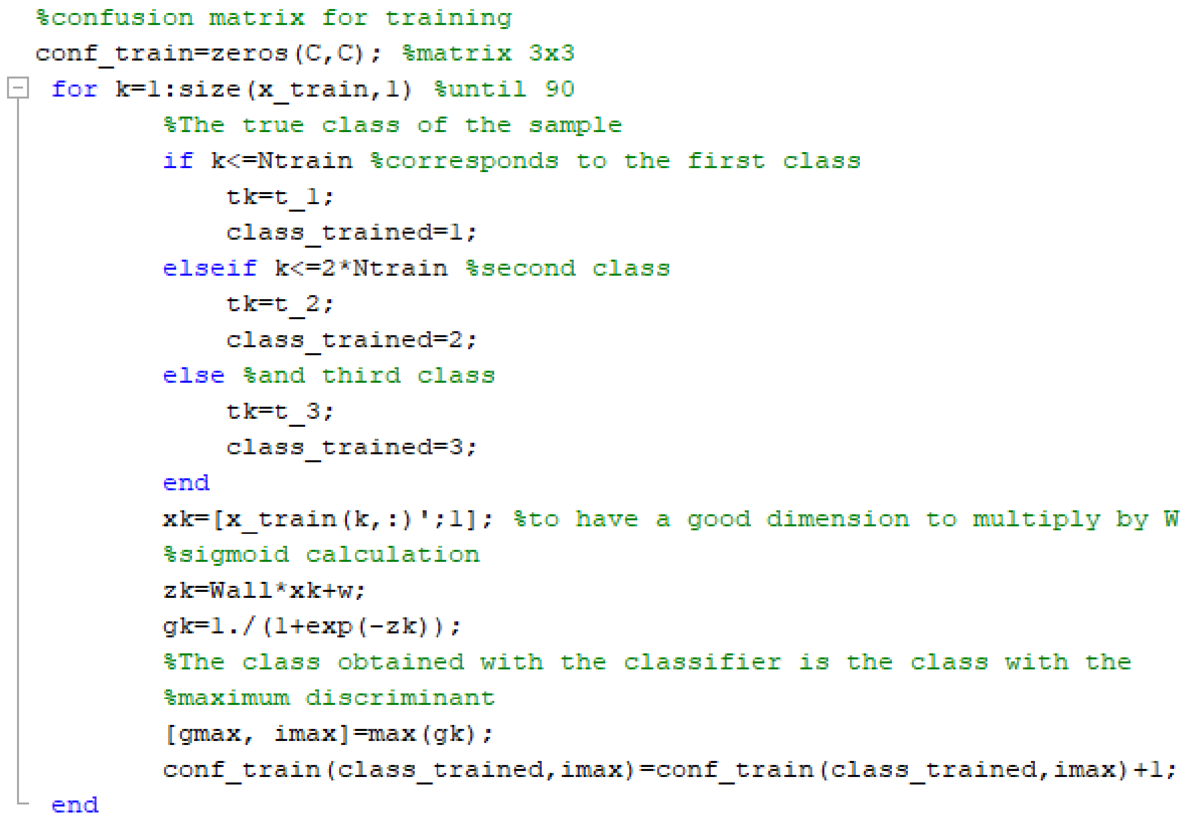


```
%training
for i=1:it
    MSE=0;
    MSE_grad=0;
    for k=1:size(x_train,1) %until 90 because 3x30
        %choose the right target
        if k<=Ntrain %corresponds to the first class
            tk=t_1;
        elseif k<=2*Ntrain %second class
            tk=t_2;
        else %and third class
            tk=t_3;
        end
        xk=[x_train(k,:)';1]; %to have a good dimension to multiply by W
        %sigmoid calculation
        zk=Wall*xk+w;
        gk=1./(1+exp(-zk));
        MSE_grad=MSE_grad+((gk-tk).*(gk).*(1-gk))*xk';
        MSE=MSE+0.5*((gk-tk)'*(gk-tk));
    end
    MSE_grad_all(i)=norm(MSE_grad);
    MSE_all(i)=MSE;
    Wall=Wall-alpha*MSE_grad;

end
```

Figure A.1: *Matlab code corresponding to the training of the classifier for the iris task*

B Matlab code for computing a confusion matrix

The code B is the one used to find the confusion matrix for the training set of the iris task.

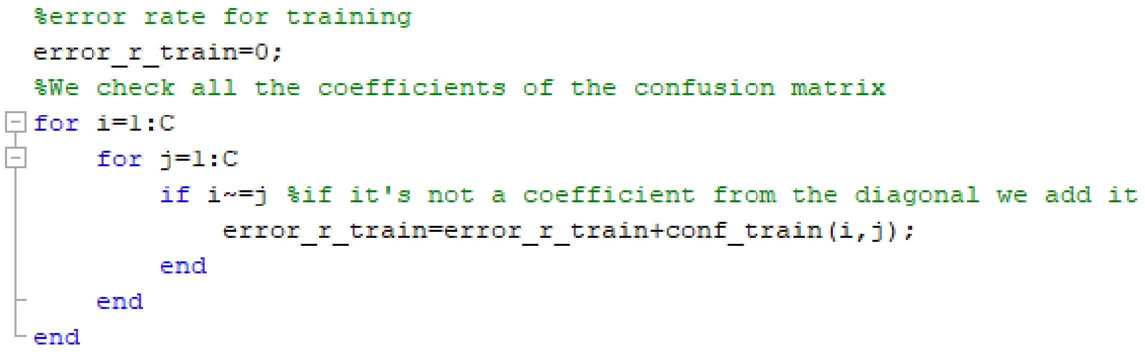


```
%confusion matrix for training
conf_train=zeros(C,C); %matrix 3x3
for k=1:size(x_train,1) %until 90
        %The true class of the sample
        if k<=Ntrain %corresponds to the first class
            tk=t_1;
            class_trained=1;
        elseif k<=2*Ntrain %second class
            tk=t_2;
            class_trained=2;
        else %and third class
            tk=t_3;
            class_trained=3;
        end
        xk=[x_train(k,:)';1]; %to have a good dimension to multiply by W
        %sigmoid calculation
        zk=Wall*xk+w;
        gk=1./(1+exp(-zk));
        %The class obtained with the classifier is the class with the
        %maximum discriminant
        [gmax, imax]=max(gk);
        conf_train(class_trained,imax)=conf_train(class_trained,imax)+1;
end
```

Figure B.1: *Matlab code corresponding to the confusion matrix for the training set of the iris task*

C Matlab code for computing the error rate

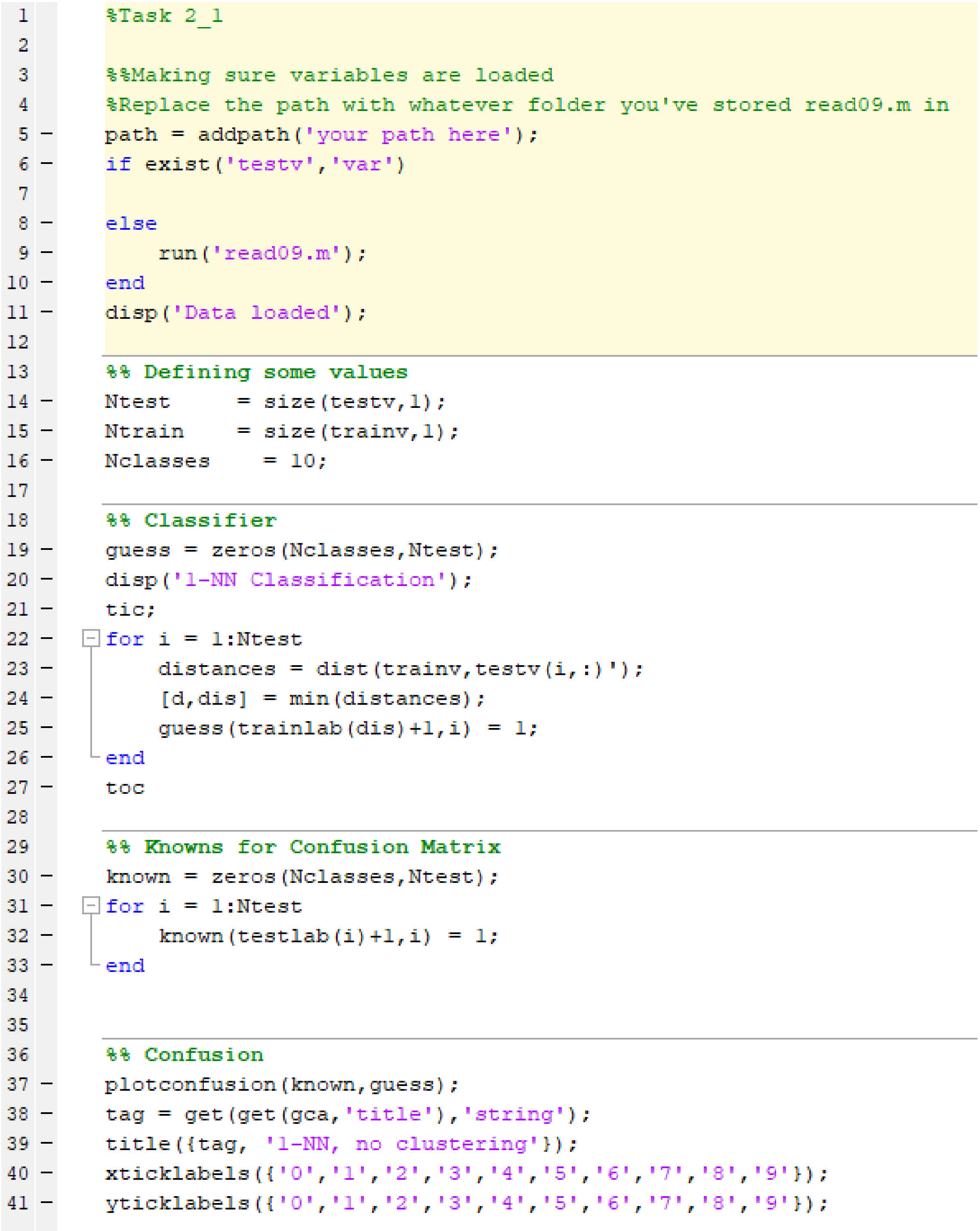
The code C is the one used to find the error rate for the training set of the iris task.



```
%error rate for training
error_r_train=0;
%We check all the coefficients of the confusion matrix
for i=1:C
    for j=1:C
        if i~=j %if it's not a coefficient from the diagonal we add it
            error_r_train=error_r_train+conf_train(i,j);
        end
    end
end
```

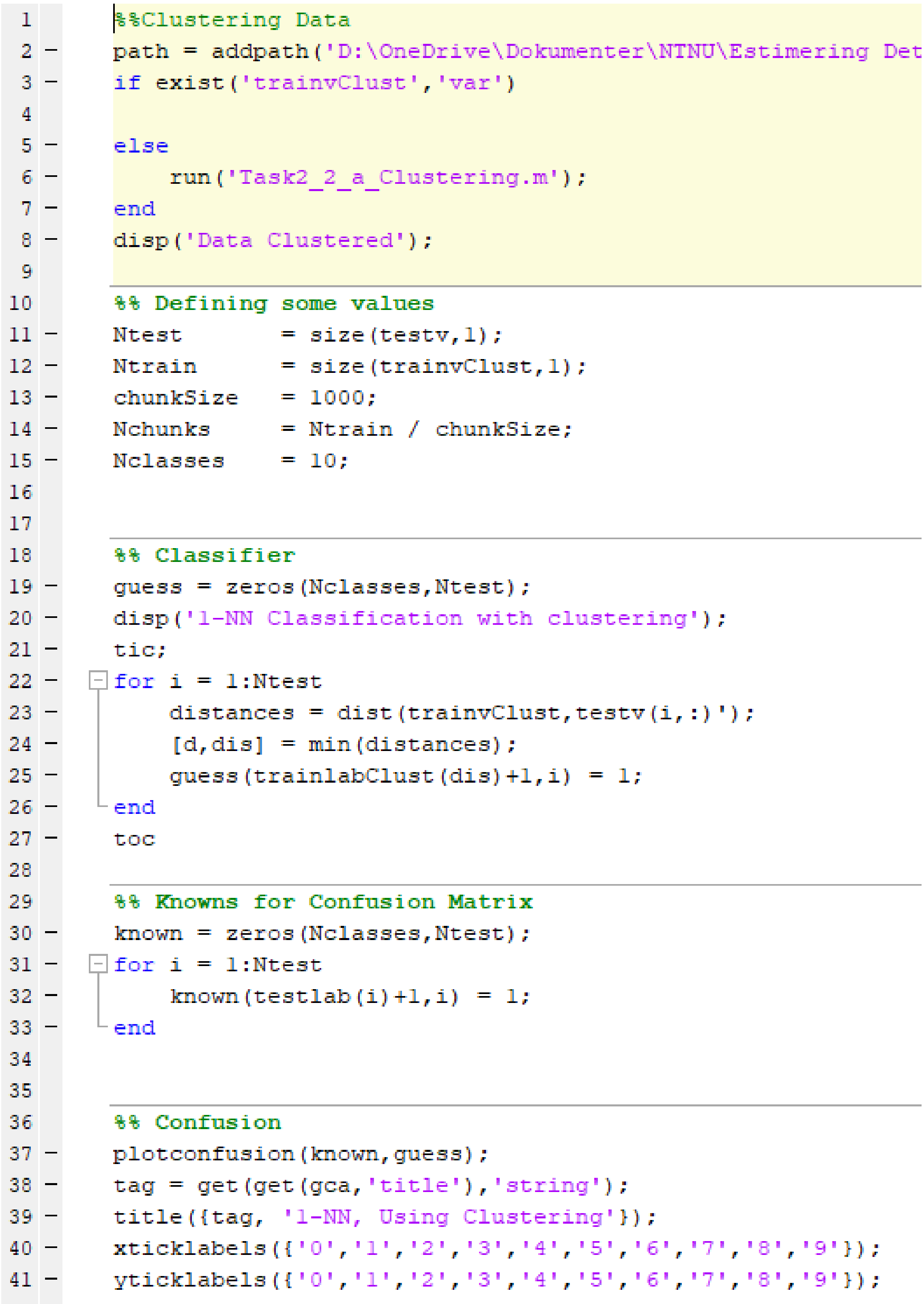
Figure C.1: *Matlab code corresponding to the error rate for the training set of the iris task*

D Matlab code for the kNN classifiers for task 2



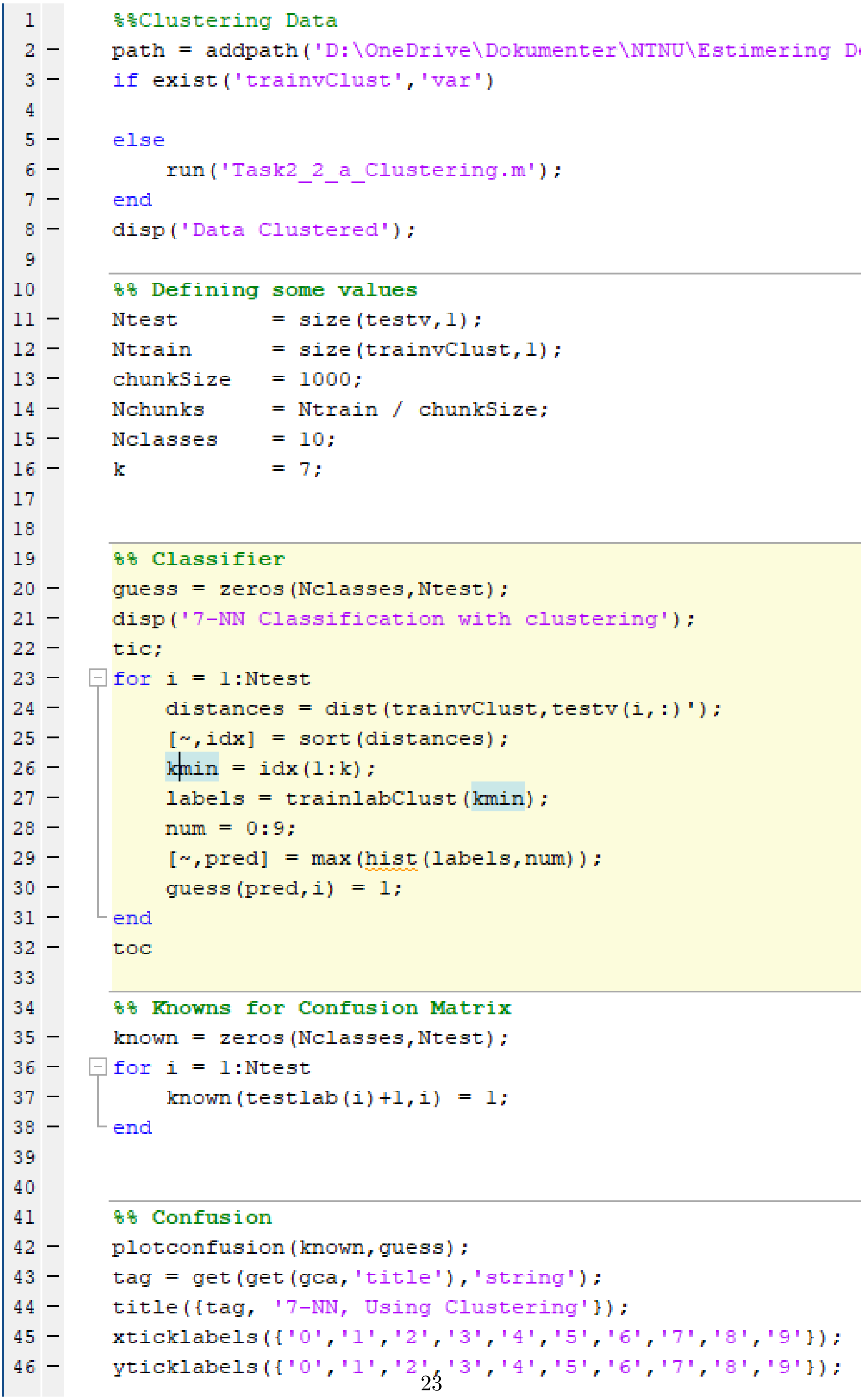
```
%Task 2_1

%%Making sure variables are loaded
%Replace the path with whatever folder you've stored read09.m in
path = addpath('your path here');
if exist('testv','var')

else
    run('read09.m');
end
disp('Data loaded');

%% Defining some values
Ntest     = size(testv,1);
Ntrain    = size(trainv,1);
Nclasses    = 10;

%% Classifier
guess = zeros(Nclasses,Ntest);
disp('1-NN Classification');
tic;
for i = 1:Ntest
    distances = dist(trainv,testv(i,:)');
    [d,dis] = min(distances);
    guess(trainlab(dis)+1,i) = 1;
end
toc

%% Knowns for Confusion Matrix
known = zeros(Nclasses,Ntest);
for i = 1:Ntest
    known(testlab(i)+1,i) = 1;
end


%% Confusion
plotconfusion(known,guess);
tag = get(get(gca,'title'),'string');
title({tag, '1-NN, no clustering'});
xticklabels({'0','1','2','3','4','5','6','7','8','9'});
yticklabels({'0','1','2','3','4','5','6','7','8','9'});
```

Figure D.1: *The code for the NN-classifier without clustering*



```
%%Clustering Data
path = addpath('D:\OneDrive\Dokumenter\NTNU\Estimering Det
if exist('trainvClust','var')

else
    run('Task2_2_a_Clustering.m');
end
disp('Data Clustered');

%% Defining some values
Ntest       = size(testv,1);
Ntrain      = size(trainvClust,1);
chunkSize   = 1000;
Nchunks     = Ntrain / chunkSize;
Nclasses    = 10;


%% Classifier
guess = zeros(Nclasses,Ntest);
disp('1-NN Classification with clustering');
tic;
for i = 1:Ntest
    distances = dist(trainvClust,testv(i,:)');
    [d,dis] = min(distances);
    guess(trainlabClust(dis)+1,i) = 1;
end
toc

%% Knowns for Confusion Matrix
known = zeros(Nclasses,Ntest);
for i = 1:Ntest
    known(testlab(i)+1,i) = 1;
end


%% Confusion
plotconfusion(known,guess);
tag = get(get(gca,'title'),'string');
title({tag, '1-NN, Using Clustering'});
xticklabels({'0','1','2','3','4','5','6','7','8','9'});
yticklabels({'0','1','2','3','4','5','6','7','8','9'});
```

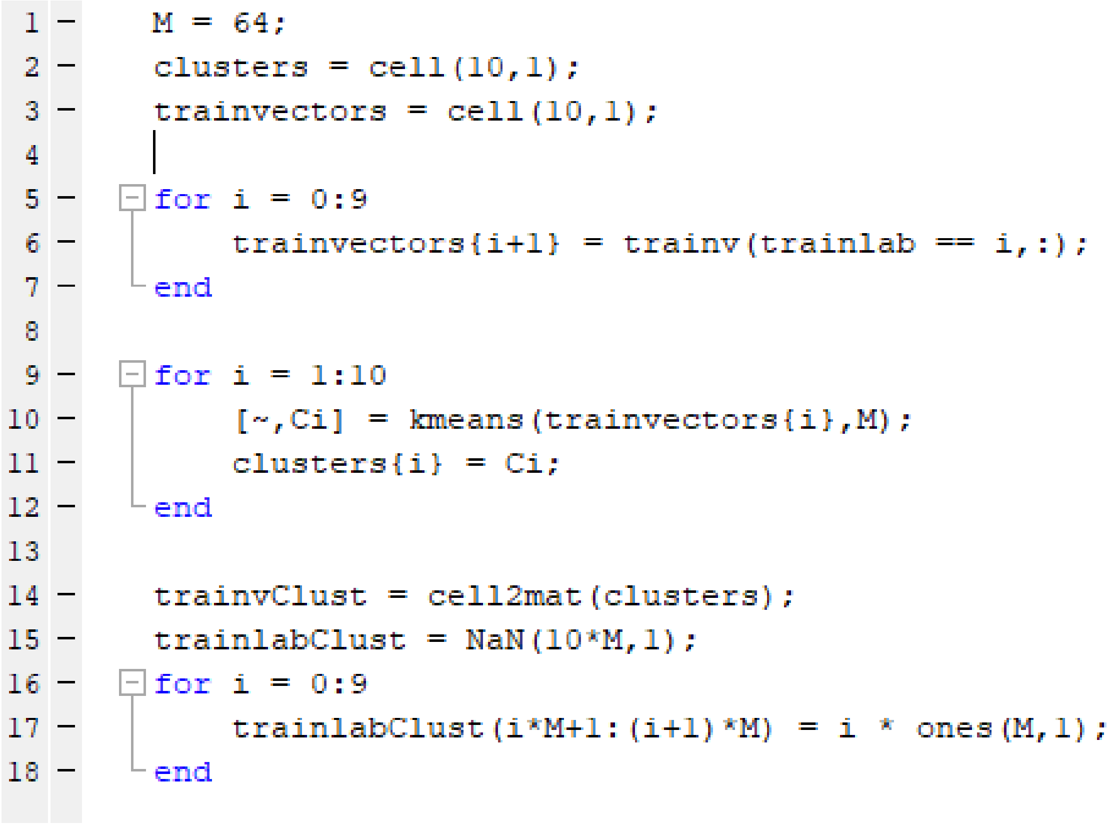
Figure D.2: *The code for the NN-classifier with clustering*



```
%%Clustering Data
path = addpath('D:\OneDrive\Dokumenter\NTNU\Estimering D
if exist('trainvClust','var')

else
    run('Task2_2_a_Clustering.m');
end
disp('Data Clustered');

%% Defining some values
Ntest         = size(testv,1);
Ntrain        = size(trainvClust,1);
chunkSize     = 1000;
Nchunks       = Ntrain / chunkSize;
Nclasses      = 10;
k             = 7;


%% Classifier
guess = zeros(Nclasses,Ntest);
disp('7-NN Classification with clustering');
tic;
for i = 1:Ntest
    distances = dist(trainvClust,testv(i,:)');
    [~,idx] = sort(distances);
    kmin = idx(1:k);
    labels = trainlabClust(kmin);
    num = 0:9;
    [~,pred] = max(hist(labels,num));
    guess(pred,i) = 1;
end
toc

%% Knowns for Confusion Matrix
known = zeros(Nclasses,Ntest);
for i = 1:Ntest
    known(testlab(i)+1,i) = 1;
end


%% Confusion
plotconfusion(known,guess);
tag = get(get(gca,'title'),'string');
title({tag, '7-NN, Using Clustering'});
xticklabels({'0','1','2','3','4','5','6','7','8','9'});
yticklabels({'0','1','2','3','4','5','6','7','8','9'});
```

Figure D.3: *The code for the 7NN-classifier with clustering*

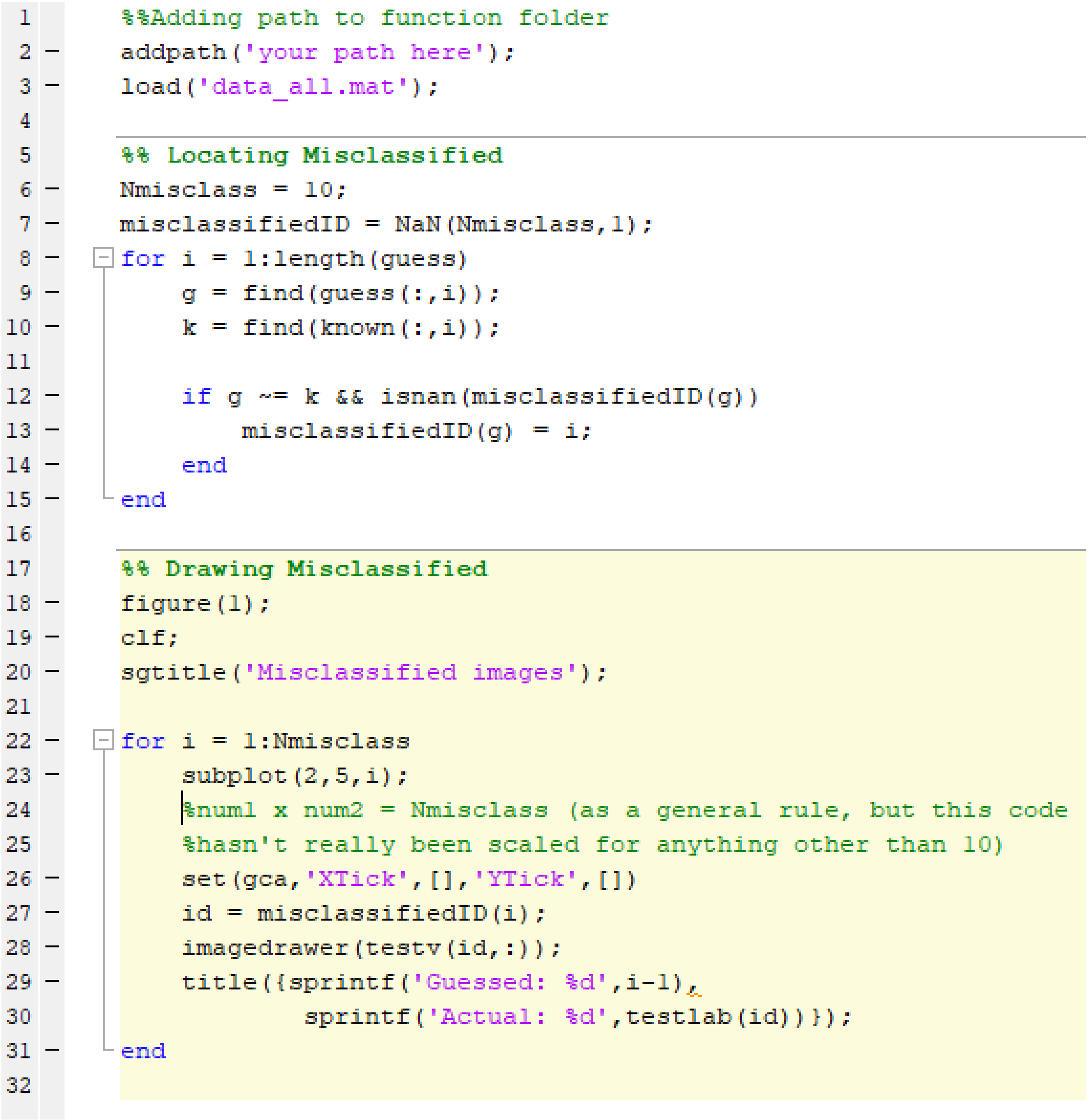
E Matlab code for performing clustering



```
M = 64;
clusters = cell(10,1);
trainvectors = cell(10,1);

for i = 0:9
    trainvectors{i+1} = trainv(trainlab == i,:);
end

for i = 1:10
    [~,Ci] = kmeans(trainvectors{i},M);
    clusters{i} = Ci;
end

trainvClust = cell2mat(clusters);
trainlabClust = NaN(10*M,1);
for i = 0:9
    trainlabClust(i*M+1:(i+1)*M) = i * ones(M,1);
end
```

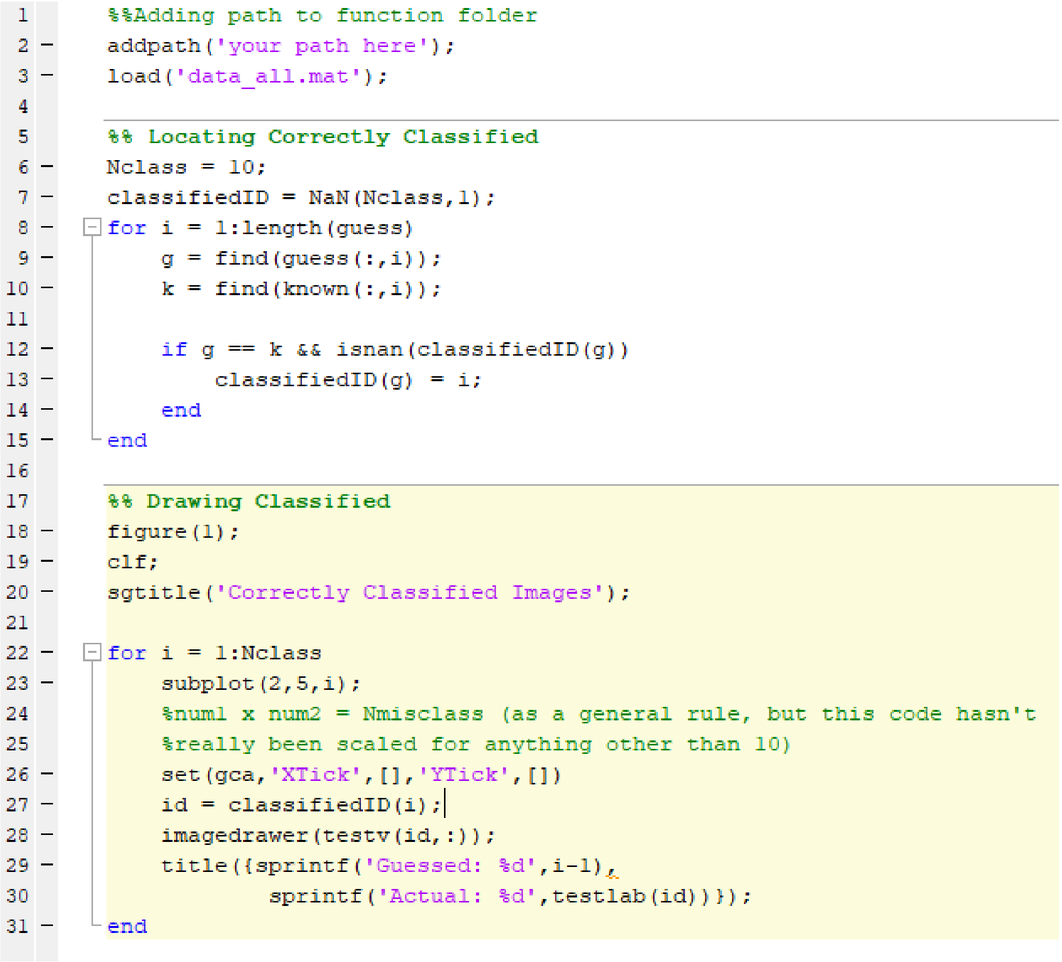
Figure E.1: *Code for clustering*

F Matlab code for plotting misclassified and correctly classified data



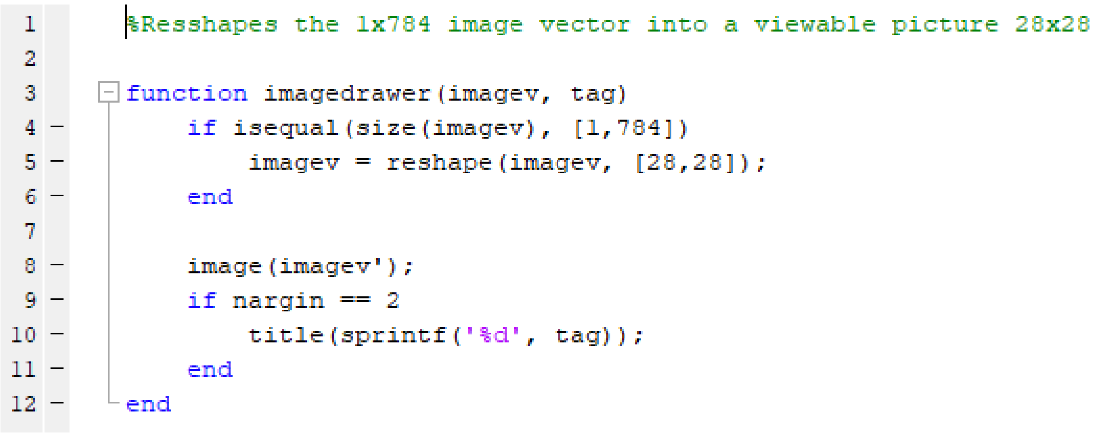
```
%%Adding path to function folder
addpath('your path here');
load('data_all.mat');

%% Locating Misclassified
Nmisclass = 10;
misclassifiedID = NaN(Nmisclass,1);
for i = 1:length(guess)
    g = find(guess(:,i));
    k = find(known(:,i));

    if g ~= k && isnan(misclassifiedID(g))
        misclassifiedID(g) = i;
    end
end

%% Drawing Misclassified
figure(1);
clf;
sgtitle('Misclassified images');

for i = 1:Nmisclass
    subplot(2,5,i);
    %num1 x num2 = Nmisclass (as a general rule, but this code
    %hasn't really been scaled for anything other than 10)
    set(gca,'XTick',[],'YTick',[])
    id = misclassifiedID(i);
    imagedrawer(testv(id,:));
    title({sprintf('Guessed: %d',i-1),
            sprintf('Actual: %d',testlab(id))});
end

```

Figure F.1: *Code for plotting misclassified data*



```
%%Adding path to function folder
addpath('your path here');
load('data_all.mat');

%% Locating Correctly Classified
Nclass = 10;
classifiedID = NaN(Nclass,1);
for i = 1:length(guess)
    g = find(guess(:,i));
    k = find(known(:,i));

    if g == k && isnan(classifiedID(g))
        classifiedID(g) = i;
    end
end

%% Drawing Classified
figure(1);
clf;
sgtitle('Correctly Classified Images');

for i = 1:Nclass
    subplot(2,5,i);
    %num1 x num2 = Nmisclass (as a general rule, but this code hasn't
    %really been scaled for anything other than 10)
    set(gca,'XTick',[],'YTick',[])
    id = classifiedID(i);
    imagedrawer(testv(id,:));
    title({sprintf('Guessed: %d',i-1),
            sprintf('Actual: %d',testlab(id))});
end
```

Figure F.2: *Code for plotting correctly classified data*



```
%Resshapes the 1x784 image vector into a viewable picture 28x28

function imagedrawer(imagev, tag)
    if isequal(size(imagev), [1,784])
        imagev = reshape(imagev, [28,28]);
    end

    image(imagev');
    if nargin == 2
        title(sprintf('%d', tag));
    end
end
```

Figure F.3: *This function draws the images for both the plottingscripts*