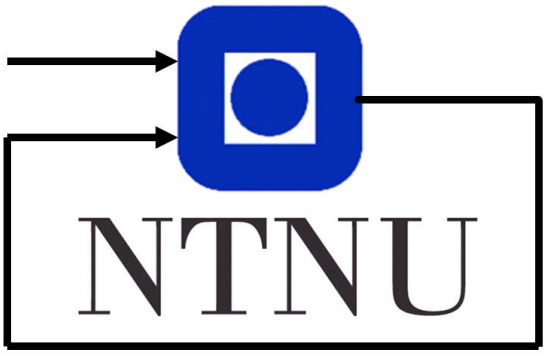
Part I:
Linear classifier analysis on the Iris flower data set

Part II:
Classifying handwritten numbers using nearest neighbor

Group 23

Even Theodor Fosby
Simon Klovning

April 30, 2024



Department of Engineering Cybernetics

Abstract

This report contains two classification projects, where Part I considers design, implementation and evaluation of an MSE based linear classifier with the goal of classifying the Iris flower data set. Part II concerns classifying of handwritten numbers 0-9 using variants of the template based classifier (nearest neighbor).

The Iris flower data set is classified using MSE based training of s linear classifier, over 3000 iterations and a step factor of $\alpha = 0.01$. The results from the classification problem demonstrates that overlapping features can significantly impact the precision of a linear classifier with error rates consistently below 6%. Removing features shows that the relative separability of the data set increases, while it also reduces the amount of data utilized, resulting in a slight increase in error rates.

Classification of the handwritten numbers using the entire data set for training resulted in a low error rate of 3.09%, but demanded a significant amount of processing time of 11 minutes. With the introduction of clustering, the processing time was 26.5 seconds which is 96% faster, with an increase in error rate to 8.81%. Advancing to a KNN-classifier resulted in a reduction of error rate to 6.57% while maintaining a reasonable processing time of 41.4 seconds.

The project resulted in a deeper understanding of how classifiers work, the importance of linear separability, clustering, data set sizes and how theoretical knowledge can be used in practice. The experience gained can be considered as fundamental knowledge in the world of classification and machine learning.

Keywords: classification, linear separability, nearest neighbor, clustering

Contents

1 Introduction

The main goal of the classification projects described in this report is to gain a deeper understanding of the working principles of classifiers. The project focuses on design, implementation of selected classifiers, as well as evaluation and comparison of their performance.

In the field of machine learning, the classifier is an algorithm which automatically categorizes data into a set of *classes*, and are commonly used in supervised learning systems[1]. By learning from the training data, it becomes able to accurately predict data it have never seen before [1]. Classifiers are widely used in a variety of fields within technology today, and are of great interest to society because they can assist with automation and optimization, risk management and scientific research across a variety of fields.

The structure of this report is based on the tasks presented in the Project Description [2], and is divided into Part I: *The Iris flower dataset* and Part II: *Classification of handwritten numbers*. In Part I, the theory behind the linear classifier can be found in Ch. 2.2. Ch. 2.3 focuses on the design and evaluation of a linear classifier, and further analyzes the relative importance of features with respect to linear separability in Ch. 2.4. This is done with the aims to design and train a linear classifier that successfully can classify three different types of Iris flowers.

Part II presents variants of a nearest neighbour classifier, which are then implemented on a much larger dataset and compared to one another. The relevant theory for the nearest neighbourhood classifier can be found in Ch. 3.2.2. In Ch. 3.4 the concept of clustering is introduced, which heavily impacts the speed of which the classifier is able to operate. The KNN classifier is implemented in Ch. 3.5. Finally, the conclusion is presented in Ch. 4.

[1]The use of labeled data sets to train algorithms to classify data or predict outcomes accurately (Source: ibm.com).

2 Part I The Iris flower data set

2.1 Project description

The Iris flower are separated into three different variants called Setosa, Versicolor and Virginica. Their main discriminators are the different lengths and widths of the Sepal (large leaves) and Petal (small leaves), which can be seen in Figure 1. These four measurements is intended to be used as the input features for a classifier.

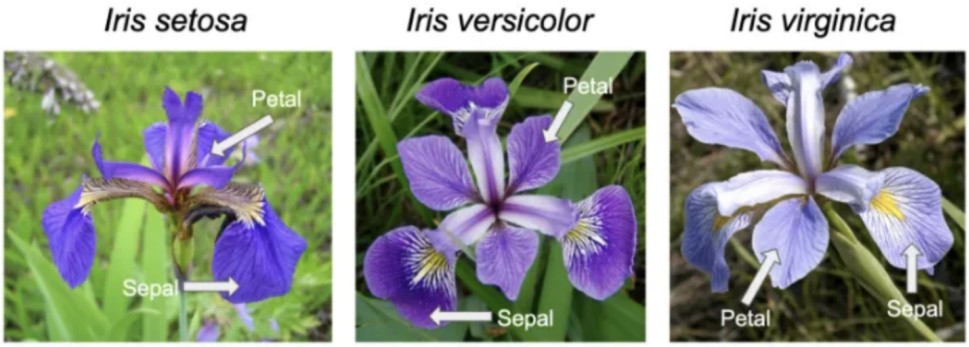


Figure 1: The three Iris variants (Image source: medium.com).

The *Iris flower data set*[2] will be utilized throughout this section, consisting of 50 examples of each variant. Since this problem is close to linearly separable, an error free linear classifier can be designed for the data set [3]. This is the intention to demonstrate in the following chapters.

For the first part of the Iris project, this report will document the preparation of the data set, MSE based training of the linear classifier, including tuning of the step factor α until the training converges. Following, the process will be repeated for different partitioning of the data set. The confusion matrix and error rate will be presented and analyzed for each case, following a comparison of the results.

In the second part, a closer study of the feature and classes will be presented. The feature which shows most overlap between the classes will be removed first, followed by training and testing of the classifier with the remaining features. This process will be repeated until the classifier is left with only one feature, and all the results for each experiment will be analyzed and discussed.

2.2 Theory of the linear classifier

2.2.1 Classifier basics and the linear classifier

To fully comprehend the classifier implementation in the following chapters, this section present the relevant theory based on the classification compendium written by Magne H. Johnsen in 2017. The main goal of a classifier is to identify the class of a previously unknown object based on its characteristics. These characteristics is often referred to as features. Classifiers can be used in a variety of contexts, such as pattern recognition, data organization and even prediction and decision making. This project will focus on the use of classifiers for pattern recognition, more specifically linear classifiers.

[2]Also called *Fisher's Iris data set*. For a detailed description of the data set, refer to Wikipedia.

2.2.2 Linear separation

The linear classifier uses a linear combination of the objects features to predict the class of new objects. It does this by defining a linear decision boundary in the feature space[4]. This boundary can be a plane or a hyperplane in multi dimensional spaces, or as a line in a two dimensional feature spaces, such as in this case. Figure 2 shows how the points can be separated in to two classes by a straight line. The classifier then bases its decision on what class an object belongs based on what side of the line it is placed. A linear classifier is adequate for solving linearly separable problems [3]. This is however uncommon in practical applications, thus will a nonlinear classifier usually have better performance. There are still many applications where the loss of performance is accepted in exchange for the relative simplicity of the linear classifier relative to its alternatives.

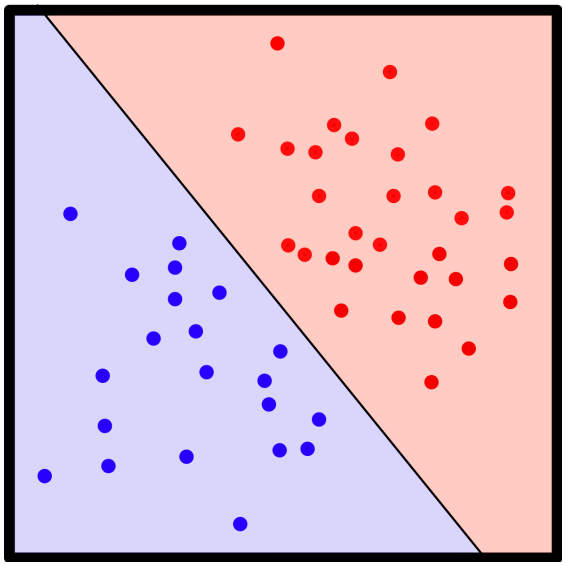


Figure 2: Example of a linearly separable data set (Image source: Wikipedia).

As shown in the scatter plot in Figure 3 the petals and sepals of the three iris flowers are close to being linearly separable, with the exception of the sepals of the Iris Versicolor and the Iris Virginica, which have quite a prominent overlap. Due to these characteristics it is possible to train a set of decision boundaries that, in theory, grants us an error free linear classifier.

2.2.3 The discriminant linear classifier

A discriminant linear classifier has an output given by $g = Wx + \omega_0$, where W is the weight matrix, x is the input and ω_0 is the bias term [3]. After a suitable classifier have been chosen, it must be trained to produce the needed performance. This is done by iteratively correcting W using a predetermined optimization criteria and a labeled training set. A labeled training set is a collection of data instances and their corresponding class. It will usually consist of an input $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N]$ and corresponding class labels $\mathbf{T} = [t_1, t_2, ..., t_N]$. The class labels can be a simple scalar value or on a $1 \times N$ vectorial form. The mapping of the continuous output g_k to the binary target vector t_k should ideally be done using the Heaviside function [3]. We do however require the derivative to compute the gradient decent, thus the function needs to be smooth. An

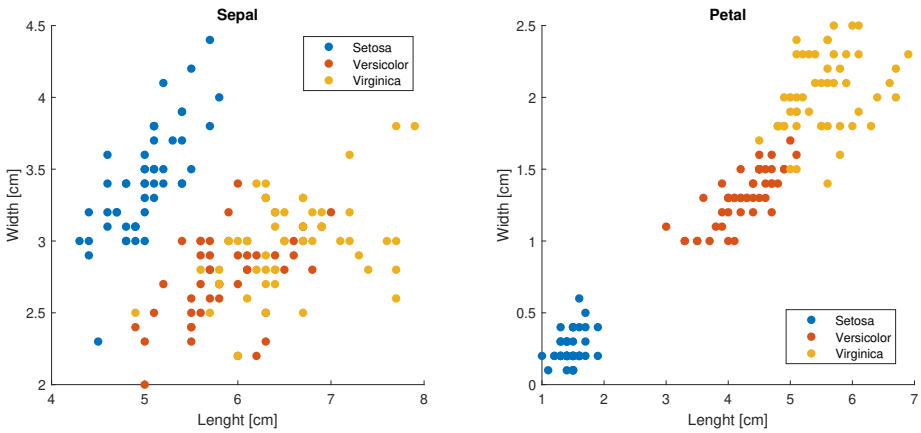


Figure 3: Scatter plot displaying the feature overlap for sepals and petals

acceptable approximation called a sigmoid function is used. This function is given by

$$g_{ik} = sigmoid(x_{ik}) = \frac{1}{1 + e^{-z_{ik}}} \quad i = 1, ..., c \tag{1}$$

It is possible to train the classifier with a range of different optimization criteria. A common one is minimum square error (MSE), which calculates the square difference between the actual value and the target. Due to this it is required to have the class labels on vectorial form

$$MSE = \frac{1}{2} \sum_{k=1}^{N} (g_k - t_k)^\top (g_k - t_k) \tag{2}$$

There is however no explicit solution to (2), and we must therefor use a form of gradient descent to compute W,

$$W(m) = W(m-1) - \alpha \nabla_W MSE \tag{3}$$

Here m is the number of iterations, α is a predetermined step factor and $\nabla_W MSE$ is

$$\nabla_W MSE = \sum_{k=1}^{N} \nabla_{g_k} MSE \nabla_{z_k} g_k \nabla_W z_k \tag{4}$$

where

$$\nabla_{g_k} MSE = (g_k - t_k)$$
$$\nabla_{z_k} g_k = g_k \circ (1 - g_k)$$
$$\nabla_W z_k = x_K^\top$$

This process applies the entire training set to the training of W for a given number of iterations, until the MSE stops decreasing, i.e. converges.

2.3 Design and training of the linear classifier

This chapter presents the process of designing, implementing and MSE based training of a linear classifier. We will begin with preparing the data set followed by implementation

and training in MATLAB. Then we will discuss tuning of the step factor parameter, and present the results. Following the results, the experiment will be repeated with a different partition in the data set.

2.3.1 Preparing the data set

Before the linear classifier can be implemented, the data set has to be prepared. In order to generalize well, one will need a labeled training set for parameter estimates, which needs to be large to achieve robust estimates [3]. In our case, this is done by splitting the data set of 50 examples into a training set consisting of the first 30 samples and the last 20 for testing, for each class. This yields a training set of 90 samples and leaves 60 samples left for the test set. The MATLAB code for the data set preparations can be found in the main program in Appendix A.1. In order to enhance the readability of the code, a custom function named `partition_dataset` was implemented to perform this operation. Note that already at this point, it is worth mentioning that this is a very small dataset, the impact of which will be demonstrated and discussed in the following chapters.

2.3.2 Implementing and training of an MSE based linear classifier

The implementing and training of the linear classifier was done in MATLAB, according to the theory presented in Ch. 2.2. The essential goal of the implementation is for the MSE to converge. The implementation involves multiple steps to process the data, including the sigmoid function found in Eq. 1. Figure 4 illustrates how the data flows and interacts with the main parts. The MATLAB code for implementing and training the linear classifier can be seen at lines 48-76 in the main program in Appendix A.1.

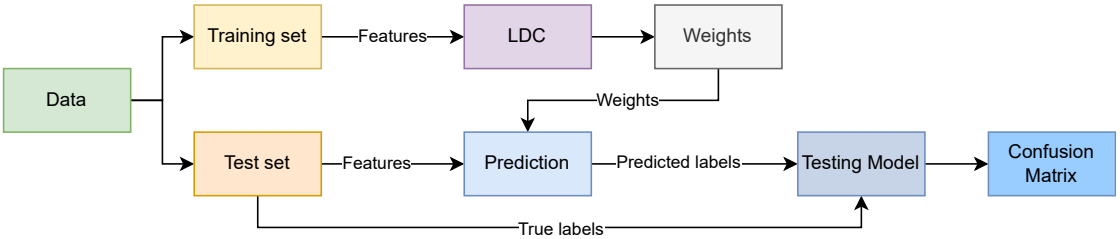


Figure 4: Data flow diagram of the MSE-based training of the linear classifier.

2.3.3 Tuning of the step factor

One important constant which must be chosen with care is the step factor α, also referred to as the learning rate [3]. This parameter controls the size of the steps taken while descending along the gradient. While testing different values for α descending from 1, the results showed that the MSE did not converge properly until a value $\alpha = 0.01$. Figure 5 shows how different values for α affect the convergence of the MSE, where the divergent values are left out from the plot. Citing the compendium, "A too large value will give a corresponding large change in W and results in large fluctuations in the MSE. Too small value will demand unnecessary many iterations in order to even get close to the minimum MSE." [3]. From the plot it is clear that a learning rate of $\alpha = 0.01$ provides the lowest MSE in the shortest amount of iterations. Further decreasing of α converges,

but at a much slower rate. Therefore, $\alpha = 0.01$ is the learning rate of choice for the further sections.

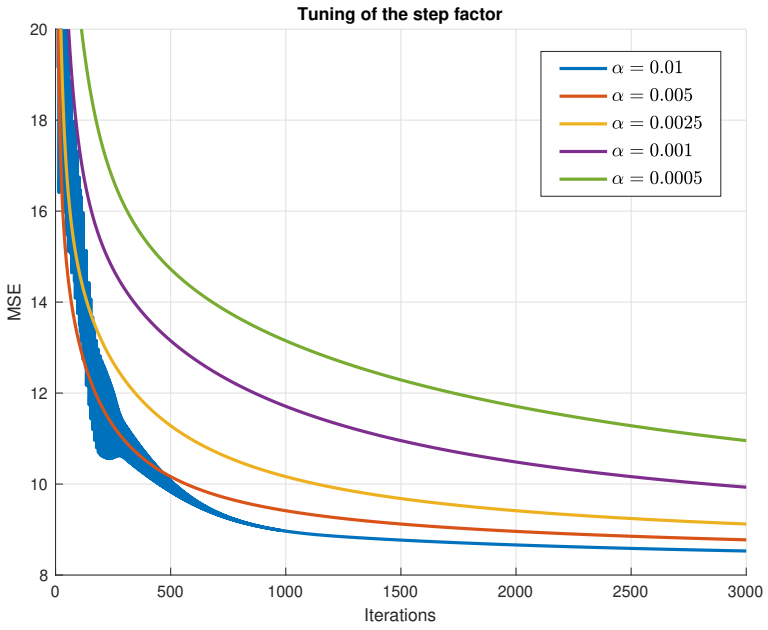


Figure 5: Results of MSE based training of the linear classifier with different values for the step factor α.

2.3.4 Results

After training the linear classifier with a step factor $\alpha = 0.01$ and 3000 iterations, the resulting confusion matrices and error rates can be found in Figure 6 and Table 1a respectively. The lower error rate of the training set compared to the test set stems from the model's tendency to optimize its parameters specifically for the training set, leading to reduced generalization ability on unseen data. A lower error rate in the training result compared to the test set is generally expected. If the difference between the errors was larger, it could indicate overfitting, which occurs when the model cannot generalize enough, and is optimized too closely to the training set. This can often be an issue when the number of available samples is inadequate [5].

Initial partitioning data set results

Data set	Error rate (%)
Training	2.22
Test	3.33

(a) Resulting error rates for the training and the test set.

Re-partitioned data set results

Data set	Error rate (%)
Training	5.56
Test	0.00

(b) Resulting error rates for the re-partitioned data set.

Table 1: Resulting error rates for (a) the initial partitioning and (b) the re-partitioned data set.

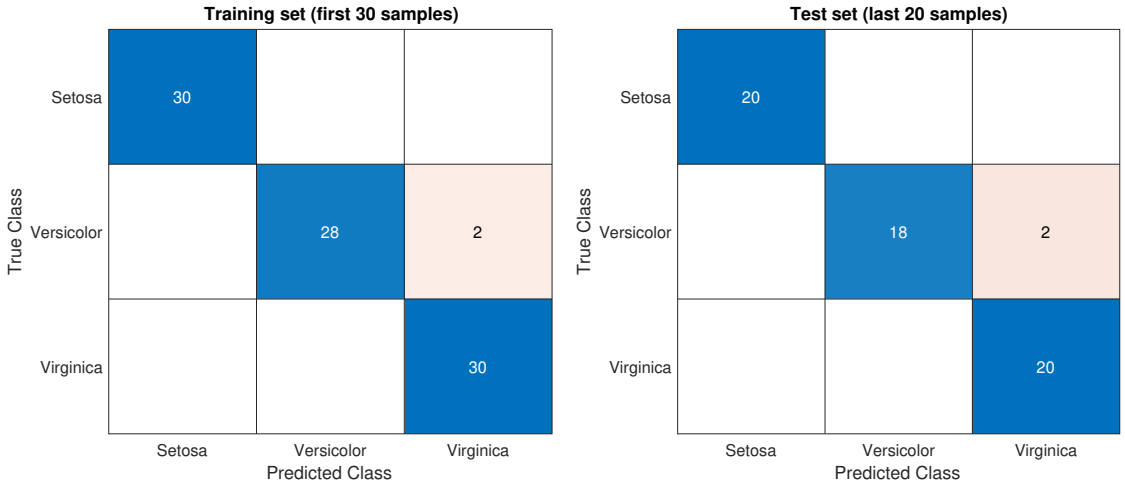


Figure 6: Resulting confusion matrices for the training set and the test set. Using all four features.

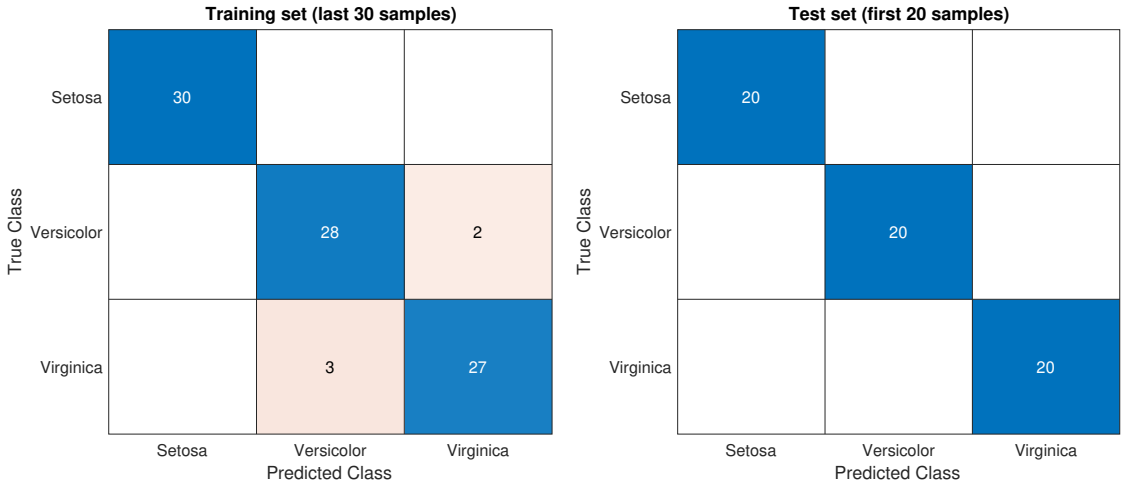


Figure 7: Alternative resulting confusion matrices for the re-partitioned data set.

2.3.5 Alternative partitioning of the data set

In order to further study the Iris data set, the next experiment involves a re-partitioning of the data set. This time, the training set and test set will remain the same, but for each class the last 30 samples will be used for training, and the first 20 will be used for testing. The MATLAB code for this operation is not directly presented in to keep the codebase concise. In the appended code found in Appendix A, re-partitioning only involves modifying the partition index at line 31, and swapping the return values at lines 32-34.

2.3.6 Results with alternative partitioning

The results from re-partitioning the data set can be seen in Figure 7 and Table 1b. While the results demonstrate an increase in error rate for the training set, observing the error rate and confusion matrix for the test set is somewhat surprising. One would expect that the error rate should be more or less the same in both cases, since the data is supposed

to be similar. Especially the fact that the error rate for the test set is zero, does not make sense intuitively. However, as mentioned in Sc. 2.3.1 the data set is significantly small. Assuming that the code has been implemented correctly, the possibility that the overlap is much smaller or even non-existent in that specific partition of the data set still exists. This result is a gentle reminder of the golden rule for machine learning "... the more data the better..." [3].

2.4 Features and linear separability

This section will concentrate on the importance of features and linear separability. As for humans using several clues/features to make decisions in different scenarios, it is well established that the error rate decreases as the number of features increases also for classifiers [3].

2.4.1 Feature and class analysis of the Iris data set

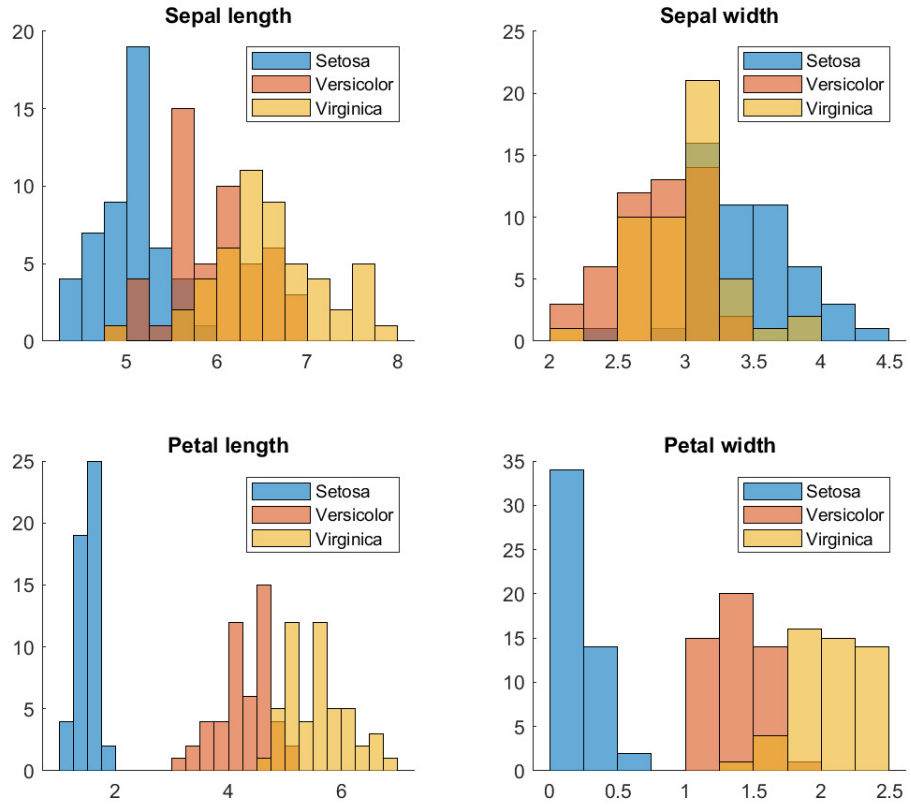


Figure 8: Histograms displaying the overlap between classes for each feature.

The histograms displayed in Figure 8 shows an approximation of the distribution of data points for each feature. It is apparent that the petal length and width are good candidates for discriminating between the three classes. There is however a larger degree of overlap in sepal width and length, especially between the Versicolor and Virginica class. The scatter plots of the feature space in Figure 3 further illustrates this overlap.

Such overlapping data can lead to a higher error rate, as it is more difficult to find a decision boundary that separates these classes precisely. This can result in lower accuracy for the classifier, which translates to an increase in misclassification. There are several ways to handle this problem, one of which is to remove the parts of the data set that contains the most overlapping values. The downside to this method is that it reduces the amount of data used to train the classifier, which can also reduce the accuracy.

2.4.2 Eliminating overlapping features

Before the following experiments, the training and test sets are reverted to their initial partitioning. To begin the eliminating process, the feature with the largest degree of overlap between the classes should be removed. In this case, it can easily be seen that this applies to the Sepal width. Repeating the process, the feature next in line is the Sepal length, and finally the Petal length. Note that the decisions in the removal process are done through visual inspection of Figure 5 and Figure 3. The process of removing each feature was done in MATLAB, and can be seen in the main program, lines 24-28 in Appendix A.1.

2.4.3 Results

The confusion matrices and error rates for classifying the data set using the four different feature combinations described above can be seen in Figures 6, 9, 10, 11, and Table 2. As excpected, the results display an increased error rate even when features are strategically removed. When reducing to only one feature, two options are displayed in the table, since it is not clearly visible which one of Petal length and Petal width has the most overlap. Note that only the confusion matrix for Petal width is presented. Including only Petal length showed a fairly big increase in the training set error, while halving the test set error. For only Petal width, the training set perform better than with two, which introduces some confusion. The test set error is still larger. These varying results might stem from the small size of the data set, introducing uncertainty to the classifier. Overall, these results show that until the point where there is no overlap we can usually not be guaranteed an error free classifier, and sticking to the maximum amount of data leads to the most reliable results.

Feature removal results

Features	Training set error (%)	Test set error (%)
All four	2.22	3.33
sl, pl, pw	3.33	3.33
pl, pw	5.56	5.00
pw	4.44	6.67
pl	11.11	3.33

Table 2: Resulting error rates with different sets of features included. The features are strategically removed, starting with most to least overlap (sl = Sepal length, pl = Petal length, pw = Petal width).

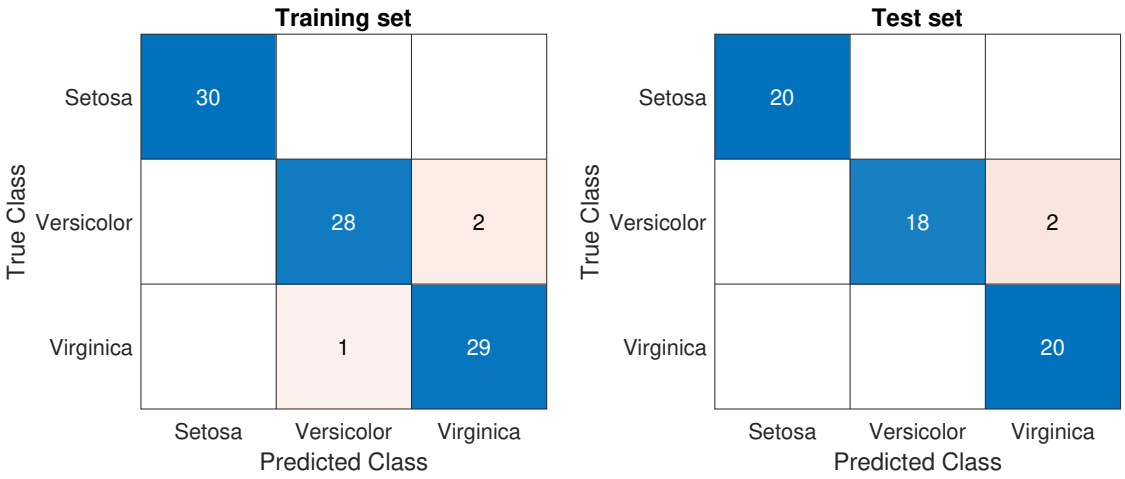


Figure 9: Resulting confusion matrices when utilizing three features.

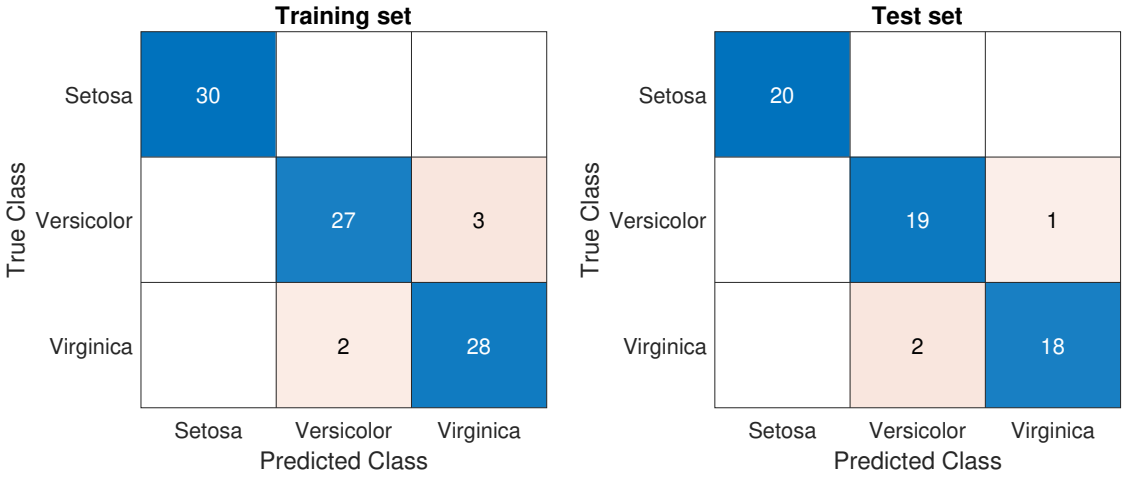


Figure 10: Resulting confusion matrices when utilizing two features.

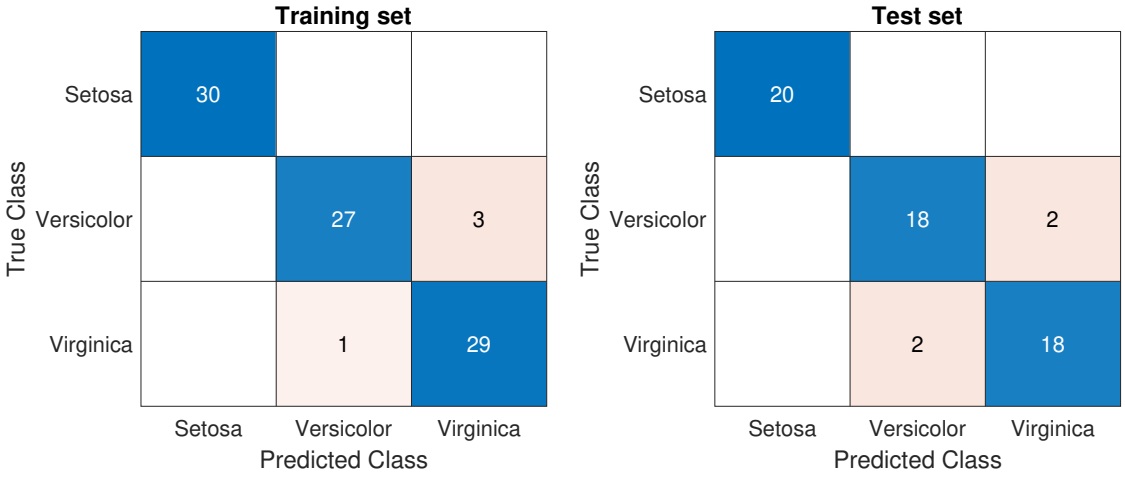


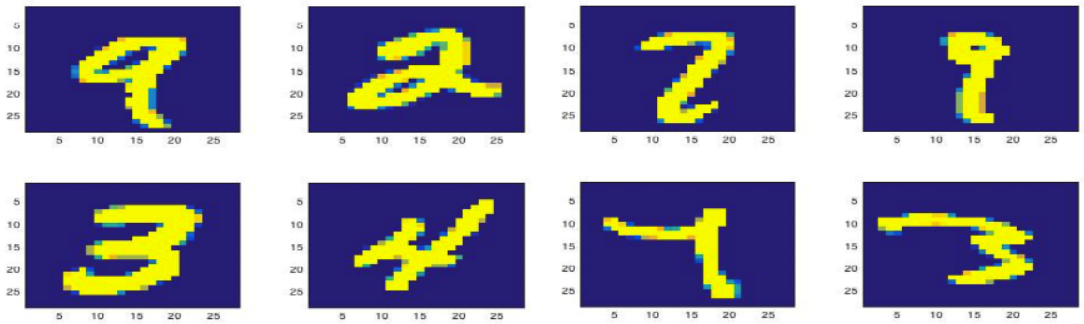
Figure 11: Resulting confusion matrices when utilizing one feature (Petal width).

3 Part II - Classification of handwritten numbers

3.1 Project description

This part of the report studies the implementation of variants of the nearest neighbor classifier with the goal of accurately and efficiently classifying handwritten numbers.

The database to be used for this part is called MNIST database[3], consisting of images of handwritten numbers 0-9. The pictures have dimension 28x28 pixels with pixel values between 0-255 (grayscale). With over 60 000 training examples 10 000 test, this database provides a solid foundation for a classification study. Numerous classifiers have previously been designed for this case, which makes the results we achieve easy to evaluate in terms of performance compared to similar cases or solutions that are considered state-of-the-art (deep neural networks) [2].



(a) "Easy" (b) "Hard"

Figure 12: Examples of handwritten numbers from the data set, with the most "dubious" examples in (b). Figure taken from [2].

This part is structured according to the two main parts of the task description [2]. In the first part, the whole training set will be used as templates, and a NN-based classifier using the Euclidean distance is implemented in Sc. 3.3.1. In order to avoid using excessive time during development the test data set was split into chunks of 1000 at a time, but all the final results presented in this report rely on the whole data set. In order to evaluate the performance, the error rate and confusion matrices will be presented in all cases. In Sc. 3.3.2, a selection of misclassified pictures is extracted for a human evaluation. This way, it is possible to get a deeper understanding of the minor details that the implemented classifiers fail to observe. The common expected error rates for classifying this database ranges from $1 - 10\%$.

In the second part the procedure of clustering is introduced in Ch. 3.4, which in essence means small(er) sets of templates for each class will be used for training instead of the whole data set. Lastly, the KNN-classifier will be introduced in Ch. 3.5.

3.2 Theory of the nearest neighbor classifier

The theory described in this chapter provides the foundation for the knowledge needed to sufficiently understand the implementations and results in the following chapters.

[3]For more detailed information on the database, see `yann.lecun.com/exdb/mnist`.

3.2.1 Euclidean distance

The NN-based classifier designed in the next chapter uses the Euclidean distance. "The Euclidean distance or Euclidean metric is the ordinary distance between two points that one would measure with a ruler. It is the straight line distance between two points. In N dimensions, the Euclidean distance between to points p and q is defined as

$$d(p,q) = \sum_{k=1}^{N} \sqrt{(p_i - q_i)}^2 \tag{5}$$

where p_i and q_i are the coordinates in dimension i" [6]. It is well suited to problems such as the one in focus of this project, as the MNIST data set consists of only real valued vectors. Note that there are several alternatives to consider for distance measuring. A brief overview can be found in Appendix C.

3.2.2 The template based classifier - Nearest Neighbor

The nearest neighbor (NN) classifier has a simple decision rule as basis for its working principle. The main idea is that the input x is matched towards a set of references (templates) on the same form as x. The reference which is closest to x is selected, and it is then assumed that x belongs to the same class as this reference. An illustration of the principle can be seen in Figure 13, where in the case of a NN-classifier the closest reference will be selected. There exists several variants of the NN-classifier, but in this experiment the Euclidean distance described in Sc. 3.2.1 is utilized.

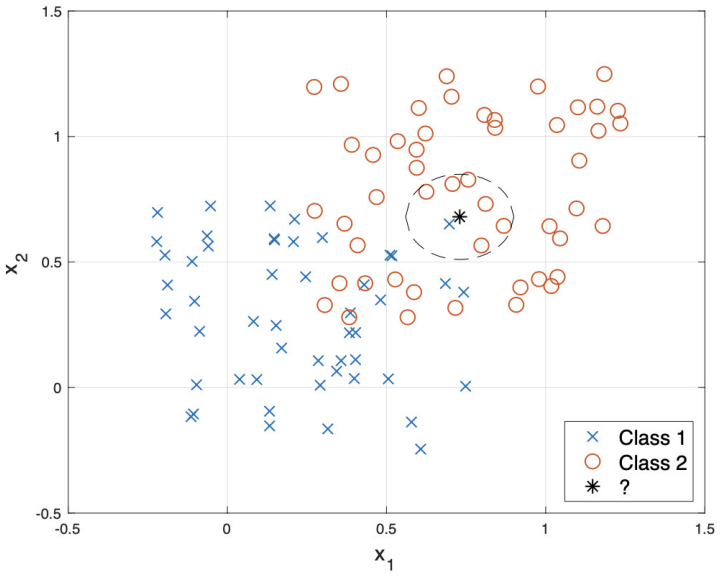


Figure 13: Classification of a single point (∗). The dotted line illustrates the interpretation of a KNN-classifier. Figure taken from [7].

3.2.3 K-means clustering

While the golden rule "... the more data the better..." still applies, training on large data sets certainly is computationally demanding and can take a long time. Drawing random

references from the training set will not help with regard to the lack of reference covariance, and a much better approach is to use a method which guarantees good coverage [3]. One of the various techniques simplifying computation and accelerating convergence is the elementary, approximate method called K-means clustering [5]. K-means clustering (also referred to as Lloyd's algorithm) is an iterative, data-partitioning algorithm[4] that assigns n observations to exactly one of k clusters defined by the centroids, where the algorithm starts with a predefined k [8].

3.2.4 The KNN-classifier

Figure 13 also demonstrates the decision rule for the KNN-classifier. Here, the classifier is extended to include the $K > 1$ nearest references. A majority vote of the candidates then decides which class the data point belongs to, where the closest class among them is selected if several classes end up with the same number [3].

3.3 The NN-based classifier using Euclidean distance

3.3.1 Designing and implementing the NN-based classifier

To design the NN-based classifier described in Sc. 3.2.2, the distance between the data point and the reference must be measured. By using the Euclidean distance as described in Sc. 3.2.1 we are able to retrieve the correct distance and compare the test image with the entire training set. The training image with the smallest distance to the test image is then selected, hence the predicted label is found. The MATLAB code for implementing this operation can be seen at lines 46-87 in the main program in Appendix B.1.

3.3.2 Results

Classifier	Error (%)	Processing time (s)
NN	3.09	660.9

Table 3: Resulting error and processing time for the test set, no clustering.

The confusion matrix, error rates and processing time for using the NN-based classifier with Euclidean distance can be seen in Figure 14 and Table 3 respectively. The error rate of 3.09% indicates fairly good accuracy considering its simple design. Inspecting the confusion matrix, it does not display any clear trends in regards to the classifying mistakes. The largest number of misclassifications seems to be classifying 9 as 4, and 5 as 3, with 22 and 19 misclassifications respectively.

Due to working on a much larger dataset of 60 000 samples, we now see that the processing time is significantly long. This is mainly due to the distance matrix being very large, meaning that each time we perform the comparison described in Ch. 3.4, we have to iterate through a huge matrix. Methods to combat this problem will be implemented in the next sections.

Although the resulting error rate is satisfactory, further investigation of the misclassified samples is interesting. As mentioned in Ch. 3.1, laying human eyes on the same examples helps with understanding why the classifier fails in some cases. Figure 15 reveals four examples of misclassified samples, and four correctly classified examples can

[4]For a detailed description of the algorithm, we refer to the MATLAB documentation.

be seen in Figure 16. While the first four clearly are challenging examples, a (very non scientific) experiment involving the authors of this report revealed that humans correctly classified these examples. There is however no doubt that a better state-of-the art classifier with an even lower error rate probably could manage to as well. The correctly classified pictures displayed in Figure 16 are generally easier to identify, and the human evaluation agrees with the classifier. The second picture from the left could easily have been mistaken for a "1" instead of "7", which in contrary to the examples in 15 was classified correctly.

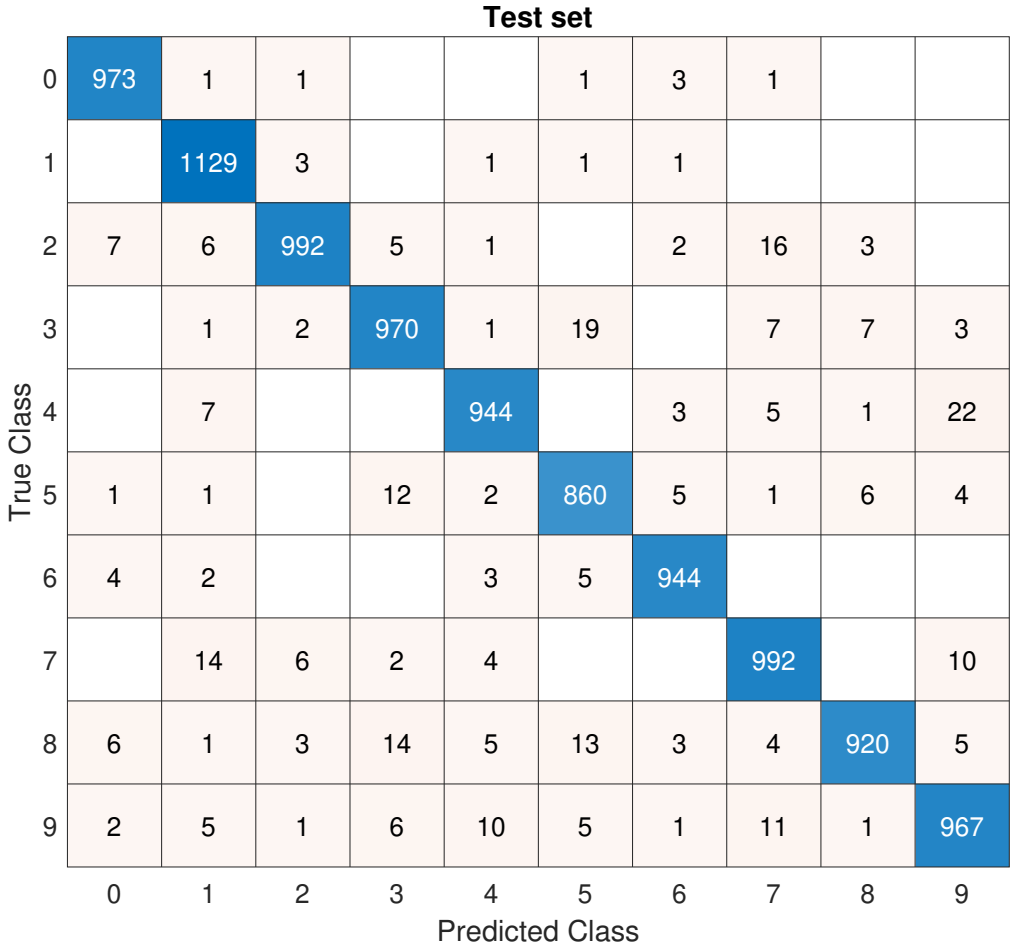


Figure 14: Confusion matrix for the test set using the NN-based classifier.

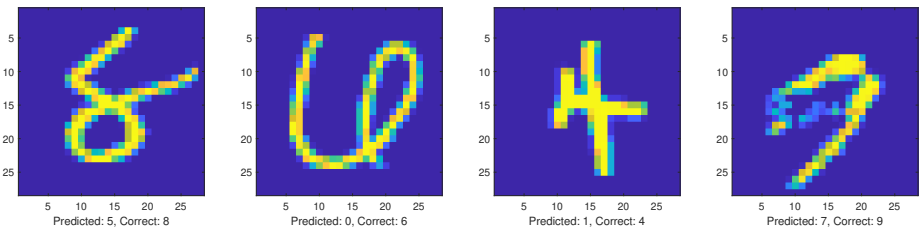


Figure 15: Misclassified pictures.

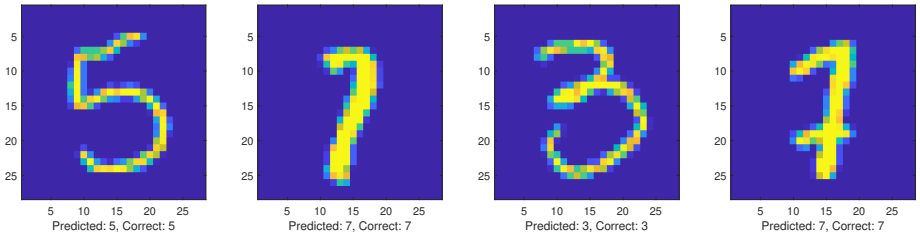


Figure 16: Correctly classified pictures.

3.4 The clustering procedure

3.4.1 Implementation of K-means clustering on the training vectors

This section introduces an efficient method for drastically reducing the processing time of classifying called K-means clustering, as described in Sc. 3.4. The implementation of K-means clustering in our project relies on the MATLAB function `kmeans`, which performs K-means clustering to partition the observations of the $n \times p$ data matrix X into k clusters, and returns an $n \times 1$ vector containing cluster indices of each observation [8]. The code implementation in MATLAB can be found in the main program, on lines 22-44 in Appendix B.1.

3.4.2 Results

The confusion matrix and error rate from using the NN-classifier again but this time with clustering can be seen in Figure 17 and Table 4. The confusion matrix now reveals that more numbers are misclassified, while also demonstrating a trend of misclassification for the numbers 2 through 5. The most notable change from the previous case without clustering, is that classifying 5 as 4 has gone from zero to 108 instances. This might be a result of the clustering procedure, which greatly reduces the size of the data set, and effectively reduces its resiliency against noise. When only using a single data point (centroid) for the classification, the accuracy of the classifier will also be reduced, as the results demonstrate.

The processing time is a significant 96% faster than without clustering, which is a substantial reduction i processing time. This does however come at the cost of a fairly large increase in misclassifications, giving the clustered NN-classifier an error rate of 8.81%. When a clustering algorithm is applied to the data set, some of the information is lost, as a larger set of data points is reduced into a single centroid. The larger volume of error is therefor a result of this information loss. Depending on the application, the difference in these results demonstrate that both these factors are important considerations one must take when selecting classifiers.

Classifier	Error (%)	Processing time (s)
NN w/ clustering	8.81	26.5

Table 4: Resulting error rate and running time for the test set using NN-classifier with clustering.

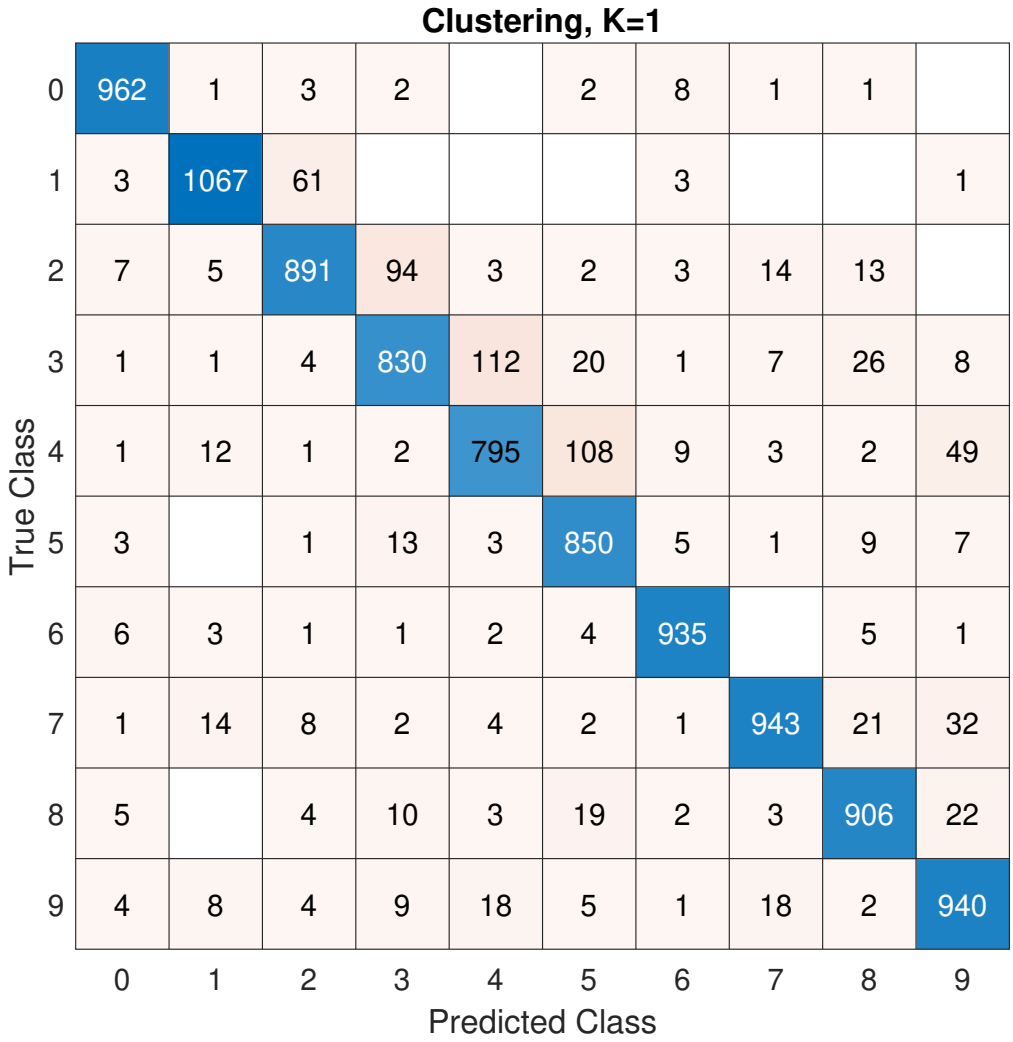


Figure 17: Confusion matrix for the test set using the KNN-classifier with clustering.

3.5 Advancing to the KNN-classifier with clustering

Based on the theory presented in Sc. 3.2.4, this section is a study of how the amount of nearest neighbors influences the result, both in terms of accuracy and processing time. As we now increase the needed computations by introducing the extra step involving selecting and voting among more neighboring reference points, we expect a slight increase in processing time, but with the benefit of better accuracy.

3.5.1 Design and implementation

Advancing from a NN-classifier to the KNN-classifier is fairly simple. The main difference from the previous implementation, is the use of the MATLAB function `mink` instead of `min`, which returns the k smallest elements of the array instead of just one. We then utilize the MATLAB function `mode` to perform the voting mechanism described in Sc. 3.2.4. The implementation of these operations can be found in the main program, on lines 65-66 and lines 71-73 in Appendix B.1.

3.5.2 Results

Classifier	Error (%)	Processing time (s)
KNN w/ clustering	6.57	41.4

Table 5: Resulting error rate and processing time for the test set using KNN-classifier with clustering and $k = 7$.

The confusion matrix, error rate and processing time can be found in Figure 18 and Table 5 respectively. As expected, the results show that the error rate has decreased, while the processing time has doubled. The increase in processing time stems from the increased computational demand of evaluating the seven nearest centroids. Comparing to the first experiment, this still means a solid 93% faster processing time, while also slightly reducing the error rate.

Through evaluation of the two clustering cases it seems that the over all performance is similar, however using $k = 7$ nearest neighbors instead of only one significantly increases the effect of noise on the classifier output. Inspecting the specific case of classifying 5 as 4 again, this has now gone back to zero, supporting the claim that the KNN-classifier is more noise resilient. The misclassification of "harder to classify numbers", for instance 9 as 4 is still present, which is understandable since this also happens more frequently for humans, due to the nature of the digits similarity.

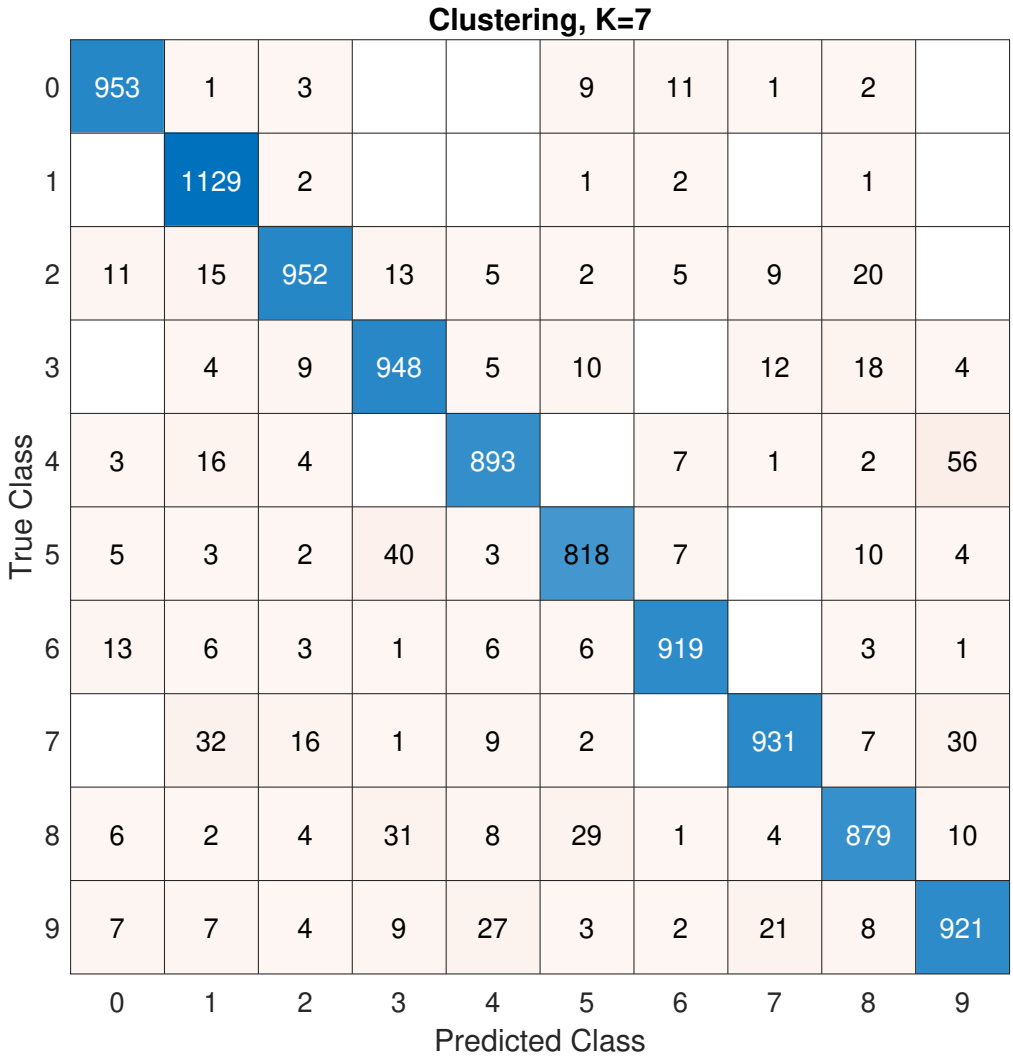


Figure 18: Confusion matrix for the test set using the KNN-classifier with clustering.

4 Conclusion

4.1 The Iris flower data set

The Iris flower classification problem consisted of designing and training of a linear discriminant classifier for determining the Iris flower type. The classifiers output is $g = Wx + \omega_0$, where W is the weight matrix. This was trained using minimum square error (MSE) with 3000 iterations and a step factor of $\alpha = 0.01$. The previously discussed results showed that the nearly linearly separable data set was classifiable using a linear classifier, with an error rate consistently below 6%. The initial partition of the data set had an error rate of 2.22% and 3.33% for the traing and testing respectivly. It was however apparent that the small size of the Iris data set introduced some unexpected behaviour, when re-partitioning the data set. By

Overlapping features can significantly impact the precision of a linear classifier. Each feature was analyzing visually with the help of histograms and scatter plots. Then the features were gradually removed based on the degree of overlap displayed. The features were removed in the following order: sepal width, sepal length, petal length. This resulted in a gradual increase in the error rate of the classifier, as shown in Figure 2. While the relative separability of the data set increases, it does also reduce the amount of data used for training and testing, resulting the slight increase in error rates.

The over all interpretation of part I of the project is that linear discriminant classifiers are well suited for applications where the data is linearly separable. There is however many instances where such a classifier preferable over more advanced classifiers. This is due to its simplistic nature and can often yield satisfactory precision depending the specifics of the problem and quality of the data.

4.2 Classification of handwritten numbers 0-9

Part II of this report introduced and studied the necessary theory for the project. The Euclidean distance is the straight line distance between two points defined by Eq. 5. The template based classifier (Nearest Neighbor) is based on a simple decision rule where the the reference which is closet to the input x is selected, classifying x to the respective class. The KNN classifier is an slightly more advanced version, introducing multiple neighboring points, where the result of a majority between them decides the classifying outcome. K-means clustering is an iterative, data-partitioning algorithm used to cluster the data points in the data set, in order to reduce processing time.

With the goal of correctly classifying handwritten numbers in the classes 0-9, an NN-based classifier using the Euclidean distance was designed and implemented in Ch. 3.3. The results demonstrated that the simple classifier achieved a fairly good accuracy of 3.09%, while the processing time took an significant time of over 11 minutes. This was mainly due to the distance matrix growing very large when using the entire data set for training. By inspecting the misclassified samples in Sc. 3.3.2, the authors of this report managed to classify samples that the classifier didn't.

With the intention of reducing the processing time, Ch. 3.4 introduced the clustering procedure. The implementation heavily relied on the MATLAB function `kmeans`, returning an $n \times 1$ vector of cluster indices of each observation. The clustering resulted in more numbers being misclassified, since the data set effectively was shrunk, making the classifier less resilient against noise. The error rate increased to 8.81%, but the processing time dropped to an impressive 26.5 seconds, which is more than 96% faster than

without clustering. When selecting classifiers, both these factors are therefore important considerations.

In the final part, the classifier was advanced to a KNN-classifier with clustering. This involved an increase in needed computations due to the extra steps with selecting and voting between multiple neighboring reference points. The implementation only involved small changes, introducing the MATLAB functions `mink` and `mode`. The results shown in Figure 18 and Table 5 respectively show a better error rate, with an almoust double in processing time. This increase is due to computational because of increased complexity, as expected. The decrease in error rate when using KNN-classifier with $k = 7$ and clustering seems to be the best compromise between processing time and error rate.

Although we are satisfied with the results of this report, there are many aspect one could have studied further. For the misclassified samples presented in Sc. 3.3.2, more detailed analysis of what is going on "inside" the classifiers could have provided a more nuanced understanding of why the results turned out the way they did. Experimenting more with the different distance measurements described in Appendix C could also have provided different results.

Upon the intention of the task, the overall experience with the project has been very instructive, and has given us gained understanding of how classifiers work. By comparing the two main parts of this project, seeing the importance of data set sizes in action was very thought-provoking. Another key learning point from this project has been to learn how all this theory actually can be implemented in MATLAB. It has also become clear to us that if we were to use Python (or even in MATLAB too), there exist a lot of built in functionality, or easily imported libraries that would speed up the implementation of these concepts. However, being forced to do it "manually" has given us valuable insight in the basic principles of these operations. The world of classification has taught us many important aspects throughout the project, keeping in mind that these fairly simple principles has built the foundation for where the technology and state-of-the art stands today.

References

[1] DeepAI. *Classifier*. https://deepai.org/machine-learning-glossary-and-terms/classifier. Accessed: 2024-04-23.

[2] Magne H. Johnsen. *Project descriptions and tasks in classification.*, 2018.

[3] Magne H. Johnsen. *TTT4275 Estimation, detection and classification: Compendium Part III - Classification*, 2017.

[4] Deepchecks. *Decision Boundary*. https://deepchecks.com/glossary/decision-boundary/. Accessed: 2024-04-23.

[5] David G. Stofk Richard O. Duda, Peter E. Hart. *Patter Classification, Second Edition*. Wiley, 2001.

[6] Suchita Gupta Shraddha Pandit. *A Comparative Study on Distance Measuring Approaches for Clustering. Journal of Research in Computer Science, 2 (1): pp. 29-31,*, 2011.

[7] NTNU Lecture notes. *Estimation, Detection and Classification, Lecture 14-15, Classification and Classification Systems*, 2024.

[8] MathWorks Help Center. *kmeans*. https://se.mathworks.com/help/stats/kmeans.html#buefthh-3. Accessed: 2024-04-26.

[9] IBM. *What is the KNN algorithm?* https://www.ibm.com/topics/knn. Accessed: 2024-04-28.

A Part I - MATLAB code - The Iris flower data set

A.1 Main program

```
%   Description: Code for designing and training a linear classifier for
%                classification of the Iris flower data set.
%
%   Author: E. T. Fosby, S. Klovning
%
%   Date: 26-04-2024

clear all; clc; close all

%% Initialization of the dataset
% Constant values
C = 3;          % number of classes
D = 4;          % number of features
N = 30;         % size of training set
M = 20;         % size of test set

iter = 3000;

% Load data set
c1_all = load('Data/class_1'); % Setosa
c2_all = load('Data/class_2'); % Versicolor
c3_all = load('Data/class_3'); % Virginica

% Remove feature from dataset
% feature_number = 2;
% c1_all = remove_feature(c1_all, feature_number);
% c2_all = remove_feature(c2_all, feature_number);
% c3_all = remove_feature(c3_all, feature_number);

% Split data set into training set and test set
partition_index = 30;
[c1_training, c1_test] = partition_dataset(c1_all,partition_index);
[c2_training, c2_test] = partition_dataset(c2_all,partition_index);
[c3_training, c3_test] = partition_dataset(c3_all,partition_index);

% Merge datasets
c_all = [c1_all; c2_all; c3_all;]';
c_training = [c1_training; c2_training; c3_training]';
c_test = [c1_test; c2_test; c3_test]';

%% Design/training and generalization
% Targets
t1 = [1 0 0]' .* ones(1, 30);
t2 = [0 1 0]' .* ones(1, 30);
t3 = [0 0 1]' .* ones(1, 30);
T = [t1 t2 t3];

% MSE based training of linear classifier
W = zeros(C, D);              % Initialize weight matrix
w0 = zeros(C, 1);
W = [W w0];

alpha = 0.01;                 % step factor
MSE_training = zeros(1,iter);
```

```
gradients_MSE_training = zeros(1, iter);

for m = 1:iter
    gradient = 0;
    MSE = 0;

    for k = 1:size(c_training,2)
        xk = [c_training(:,k); 1];

        tk = T(:, k);

        zk = W * xk + w0;
        gk = sigmoid(zk);

        gradient = gradient + (gk-tk) .*gk.*(1-gk)*xk';
        MSE = MSE + 1/2 * (gk-tk)'*(gk-tk);
    end

    W = W - alpha * gradient;
    MSE_training(m) = MSE;
    gradients_MSE_training = norm(gradients_MSE_training);
end

% Confusion matrix
predicted_training_labels = zeros(1, N*C);
predicted_test_labels = zeros(1, M*C);
actual_training_labels = kron(1:C, ones(1, N));
actual_test_labels = kron(1:C, ones(1, M));

% Classify training set
for k = 1:size(c_training,2)
    xk = [c_training(:,k); 1];
    zk = W * xk + w0;
    gk = sigmoid(zk);
    [¬, predicted_label] = max(gk);
    predicted_training_labels(k) = predicted_label;
end

% Classify test set
for k = 1:size(c_test,2)
    xk = [c_test(:,k); 1];
    zk = W * xk + w0;
    gk = sigmoid(zk);
    [¬, predicted_label] = max(gk);
    predicted_test_labels(k) = predicted_label;
end

% Compute confusion matrix
confusion_matrix_training = confusionmat(actual_training_labels, ...
    predicted_training_labels);
confusion_matrix_test = confusionmat(actual_test_labels, ...
    predicted_test_labels);

% Calculate error rate
error_rate_training = 1 - sum(diag(confusion_matrix_training)) / ...
    sum(sum(confusion_matrix_training));
error_rate_test = 1 - sum(diag(confusion_matrix_test)) / ...
    sum(sum(confusion_matrix_test));

% Display confusion matrix and error rates
```

```
111 disp('Confusion Matrix (Training Set):');
112 disp(confusion_matrix_training);
113 fprintf('Error Rate (Training Set): %.2f%%\n', error_rate_training * 100);
114
115 disp('Confusion Matrix (Test Set):');
116 disp(confusion_matrix_test);
117 fprintf('Error Rate (Test Set): %.2f%%\n', error_rate_test * 100);
118
119 %% Features and linear separability
120 % Histograms of features
121 all_histogram_feature(c1_all, c2_all, c3_all, 1);
122
123 %Scatter plot of features
124 scatter_plot(c1_all, c2_all, c3_all, 5);
125
126
127 %% Sigmoid function
128 function y = sigmoid(x)
129     y = 1 ./ (1 + exp(-x));
130 end
```

A.2 Support functions

```
1     function [first, last] = partition_dataset(dataset,partitionIndex)
2 % remove_feature: Splits the dataset into two separate datasets at the
3 %                 partitionIndex
4 %
5 % Output:         first: The dataset before the partitionIndex
6 %                 last: The dataset after the partitionIndex
7 %
8 %
9 % Input:          dataset: The dataset that is to be modified.
10 %                patritionIndex: Index of where det dataset should be
11 %                split.
12
13     first = [dataset(1:partitionIndex, :)];
14     last = [dataset(partitionIndex+1:end, :)];
15 end
```

```
1     function modified_dataset = remove_feature(dataset, feature_number)
2 % remove_feature: Removes the feature of index featrue_number from the
3 %                 dataset.
4 %
5 % Output:         modified_dataset: The input dataset without the feature
6 %                                   of index feature_number.
7 %
8 % Input:          dataset: The dataset that is to be modified.
9 %                 feature_number: The index of the feature that shold be
10 %                                removed.
11
12     modified_dataset = dataset;
13     modified_dataset(:,feature_number) = [];
14 end
```

A.3 Plotting functions

```
1     function all_histogram_feature(dataSet1, dataSet2, dataSet3, figNum)
2  %all_histogram_feature: Plots a histogram of all four features in a 2x2
3  %                       matrix. Each histogram contains all three classes.
4  %
5  % Output:           Four histograms in a 2x2 matrix
6  %
7  % Input:            dataSetX: Dataset of each class that should be plotted
8  %                   figNum: The number given to the plotted figure.
9
10 p11 = dataSet1(:, 1);
11 p12 = dataSet2(:, 1);
12 p13 = dataSet3(:, 1);
13
14 p21 = dataSet1(:, 2);
15 p22 = dataSet2(:, 2);
16 p23 = dataSet3(:, 2);
17
18 p31 = dataSet1(:, 3);
19 p32 = dataSet2(:, 3);
20 p33 = dataSet3(:, 3);
21
22 p41 = dataSet1(:, 4);
23 p42 = dataSet2(:, 4);
24 p43 = dataSet3(:, 4);
25
26 figure(figNum)
27 subplot(2,2,1)
28 hold on
29 p_h1 = histogram(p11);
30 p_h1.BinWidth = 0.25;
31 p_h2 = histogram(p12);
32 p_h2.BinWidth = 0.25;
33 p_h3 = histogram(p13);
34 p_h3.BinWidth = 0.25;
35 legend('Setosa', 'Versicolor', 'Virginica'); title('Sepal length')
36 hold off
37
38 subplot(2,2,2)
39 hold on
40 p_h1 = histogram(p21);
41 p_h1.BinWidth = 0.25;
42 p_h2 = histogram(p22);
43 p_h2.BinWidth = 0.25;
44 p_h3 = histogram(p23);
45 p_h3.BinWidth = 0.25;
46 legend('Setosa', 'Versicolor', 'Virginica'); title('Sepal width')
47 hold off
48
49 subplot(2,2,3)
50 hold on
51 p_h1 = histogram(p31);
52 p_h1.BinWidth = 0.25;
53 p_h2 = histogram(p32);
54 p_h2.BinWidth = 0.25;
55 p_h3 = histogram(p33);
56 p_h3.BinWidth = 0.25;
```

```
legend('Setosa', 'Versicolor', 'Virginica'); title('Petal length')
hold off

subplot(2,2,4)
hold on
p_h1 = histogram(p41);
p_h1.BinWidth = 0.25;
p_h2 = histogram(p42);
p_h2.BinWidth = 0.25;
p_h3 = histogram(p43);
p_h3.BinWidth = 0.25;
legend('Setosa', 'Versicolor', 'Virginica'); title('Petal width')
hold off

end
```

```
    function scatter_plot(dataSet1, dataSet2, dataSet3, figNum)
% scatter_plot: Produces a scatter plot of all four features in a 1x2
%               matrix. Each point (x,y) consists of (length, width) of
%               either sepals or petals.
%
% Output:           Two scatter plots in a 1x2 matrix
%
% Input:            dataSetX: Dataset of each class that should be plotted
%                   figNum: The number given to the plotted figure.

%Sepal
p11 = dataSet1(:, 1);
p12 = dataSet1(:, 2);

p21 = dataSet2(:, 1);
p22 = dataSet2(:, 2);

p31 = dataSet3(:, 1);
p32 = dataSet3(:, 2);

%Petal
p13 = dataSet1(:, 3);
p14 = dataSet1(:, 4);

p23 = dataSet2(:, 3);
p24 = dataSet2(:, 4);

p33 = dataSet3(:, 3);
p34 = dataSet3(:, 4);

%Plot
figure(figNum);
hold on
scatter(p11,p12, 'filled');
scatter(p21,p22, 'filled');
scatter(p31,p32, 'filled');
xlabel('Lenght [cm]'); ylabel('Width [cm]'); legend('Setosa', ...
    'Versicolor', 'Virginica'); title('Sepal')
hold off

figure(figNum+1);
hold on
```

```
scatter(p13,p14, 'filled');
scatter(p23,p24, 'filled');
scatter(p33,p34, 'filled');
xlabel('Lenght [cm]'); ylabel('Width [cm]'); legend('Setosa', ...
    'Versicolor', 'Virginica'); title('Petal')
hold off

end
```

B Part II - Classifying handwritten numbers - MATLAB code

B.1 Main program

```
%   Description: KNN-algorithem for classification of the MNIST data set.
%                The classifier is trained using 60 000 training images
%                and 10 000 test images.
%
%   Author: E. T. Fosby, S. Klovning
%
%   Date: 26-04-2024

clear all; clc; close all

tic
%% Initialization
% Constant values
num_classes = 10;    % number of classes
M = 64;              % number of clusters
k = 7;               % number of nearest neighbors

% Initialize data set
load('data/data_all.mat');

%% Clustering
% Sort the training set
[trainlab_sorted, sortIdx] = sort(trainlab, 'descend');
trainv_sorted = trainv(sortIdx, :);

% Generate centroids
centroids_vec = zeros(M*num_classes, vec_size);

for i = 1:num_classes
    reduced_trainv_sorted = split_to_chunks(trainv_sorted, i, ...
        size(trainv_sorted,1)/num_classes);
    [¬, centroids_vec(i*M-M+1:i*M, :)] = kmeans(reduced_trainv_sorted, M);
end


% Generate centroids labels
centroids_lab = zeros(size(centroids_vec,1),1);
class_counter = num_classes - 1;

for j = 1:num_classes
    centroids_lab(j*M-M+1:j*M, :) = class_counter;
    class_counter = class_counter-1;
```

```
43 end
44
45 %% NN-based classifier using the Euclidian distance
46
47 confusion_matrix = zeros(num_classes, num_classes);
48 incorrect = [];
49
50 % Train on the whole training set
51 train_images = centroids_vec;
52 train_labels = centroids_lab;
53
54 % Test on a subset of 100 samples
55 num_test_samples = 10000;
56 test_images = testv(1:num_test_samples, :);
57 test_labels = testlab(1:num_test_samples);
58
59 % Iterate over test images
60 for j = 1:num_test_samples
61     % Compute euclidean distance
62     distances = sum((train_images - test_images(j, :)).^2, 2);
63
64     % Find k nearest neighbors
65     [¬, nn] = mink(distances, k);
66
67     % Find nearest neighbor (NN-algorithem, task 1)
68     % [¬, nn] = min(distances);
69
70     % Predicted label based on nearest neighbor
71     predicted_labels = train_labels(nn);
72     predicted_label = mode(predicted_labels);
73
74     % True label
75     true_label = test_labels(j);
76
77     % Update confusion matrix
78     confusion_matrix(true_label + 1, predicted_label + 1) = ...
79     confusion_matrix(true_label + 1, predicted_label + 1) + 1;
80
81     % Check if misclassified
82     if true_label ≠ predicted_label
83         incorrect = [incorrect; j, true_label, predicted_label];
84     end
85
86 end
87
88 % Compute error rate
89 error_rate = 1 - sum(diag(confusion_matrix)) / sum(sum(confusion_matrix));
90
91 % Display confusion matrix and error rate
92 disp('Confusion Matrix:');
93 disp(confusion_matrix);
94 fprintf('Error Rate: %.2f%%\n', error_rate * 100);
95 toc
```

B.2 Support functions

```
1      function [chunked_vec] = split_to_chunks(dataset, partition_index, ...
2      chunk_size)
3  %splitToChunks: Function that partition the input data set into equal ...
       parts
4  %               of size chunk_size and return the chunk at index
5  %               partition_size.
6  %
7  % Output:       A vector of size chunk_size
8  %
9  % Input:        dataSet: Data set that is partitioned
10 %               partition_index: The index of the element that should be
11 %                                returned
12 %               chunk_size: Size of the elements that should be ...
       partitioned
13 %
14
15 if partition_index == 1
16     i = 1;
17 else
18     i = (partition_index-1)*chunk_size;
19 end
20
21 chunked_vec = dataset(i:i+chunk_size-1, :);
22 end
```

C Alternative distance measuring methods

The Euclidean distance is among the most common distance measures for NN-based classifiers. While it is a suitable metric for classification of handwritten numbers, there are several alternatives to consider based of the type and format of the data set. The following list serves to give a brief overview of some of the other commonly used distance metrics.

- **Manhattan distance**: Also referred to as the taxicab distance, as it is commonly visualized with a grid [9]. It measures the absolute distance between two points by summing the difference between their coordinates. This measure works well for higher dimensional data, and in cases with features of different scales. It is expressed mathematically as:

$$Manhattan = \sum_{i=1}^{m} |x_i - y_i| \tag{6}$$

- **Minkowski distance**: A generalized form of the Euclidean and Manhattan distance metric [9]. The Minkowski distance offers more flexibility as it can be tuned to emphasise some features more than others. This is done by changing the value of p. For $p = 2$ it functions as the Euclidean distance and for $p = 1$ as the Manhattan distance.

$$Minkowski = \sum_{i=1}^{m} (|x_i - y_i|)^{\frac{1}{p}} \tag{7}$$

- **Hamming distance**: It is typically used with two or more boolean or string vectors to calculate the number of instances that do not match. This means that the Hamming distance is only suitable for data where the features can only take a limited number of discrete values. It can be represented mathematically as:

$$Hamming = \sum_{i=1}^{m} (|x_i - y_i|) \tag{8}$$

$$\begin{aligned} x = y \quad & D = 1 \\ x \neq y \quad & D = 0 \end{aligned}$$