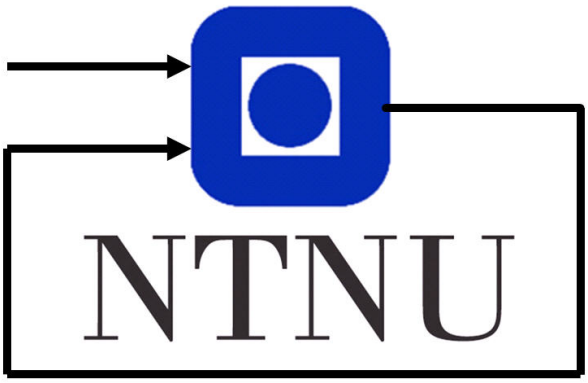
Classification Project

Group 66

Student numbers:
490950
490907

Spring, 2020



Summary

This project was a part of the course *TTT4275 Estimation, detection and classification*, given at NTNU Trondheim. There were in total five different topics to choose from, regarding estimation, detection or classification. Of these five we chose one of the classification exercises, which consists of classification of three different types of Irises based on the length and width of their two leaves, as well as a template-based classification of handwritten numbers 0-9 using nearest-neighbors and k-nearest-neighbors.

We implemented a multi-class perceptron algorithm for the Iris flowers, since this was linearly separable. Here, we attained an error rate at only 1.11% measuring all the four features. Removing features proved to increase the error rate a little, which makes sense due to the rule: "the more data, the better". The confusion matrices showed that class 1 did not get any errors, which was expected since this was linearly separable.

For solving the classification of handwritten numbers, we implemented the nearest-neighbor algorithm. Having 60000 training/template vectors, 10000 test vectors and 784 features (the pixtures of handwritten numbers was 8-bit greyscaled in 28x28 pixels), we achieved an error rate of 3.09%, however a large computational time of 35 minutes and 12 seconds. By clustering the data set into 64 clusters per class, we got a much lower computational time of only 69 seconds, without punishing the error rate too much (4.64%). Lastly, we extended the nearest-neighbor algorithm to the more advanced k-nearest-neighbors algorithm with $k = 7$. Surprisingly, the error rate got a bit larger (6.54%), and $k = 1$ seemed to be reasonable for this case.

Contents

1 Introduction **1**

2 Theory **2**
2.1 Short intro to classification and clustering 2
2.2 Terminology 3
2.3 Some useful classifiers 4

3 Task **6**
3.1 Description of project topics 6
3.2 Choice of topic 6
3.3 Project 4. Classification - Iris and handwritten numbers 0-9 7

4 Implementation and results **8**
4.1 The Iris task 8
4.2 Classification of handwritten numbers 0-9 14

5 Conclusion **18**

References **19**

1 Introduction

This report covers our semester project in *TTT4275 - Estimation, detection and classification.* We chose the classification exercise, where our task was to classify three different variants of the Iris flower using a linear classifier and classify handwritten numbers based on the nearest-neighbor and k-nearest-neighbor algorithm. In other words, we were to classify two sets of data; one using a linear classifier and one using a template-based classifier. The test results are mainly presented in different confusion matrices and error rates, combined with some run-time results in the latter exercise where this was a relevant topic. The Iris flower classification algorithm is relevant for making determination of species more efficient, having some measurable features to classify the correct variant faster. The classification of handwritten numbers have a lot of use cases, classifying a large amount of numbers in short time in for example a mobile phone or a tablet. It can also be extended to interpret handwritten letters if one finds an accurate and fast algorithm to recognize digits.

In section 2 the most important theory of classification is provided to be able to follow and understand the results and conclusions later on in the report. In section 3 a bigger presentation of the different project exercises is given, as well as an explanation of why we chose to work on one of the classification exercises. In section 4 we present our results with Matlab code and figures, and discuss our findings. Finally, section 5 gives a conclusion based on our observations and discussion.

2 Theory

This section provides the most important theory to better understand the classification and clustering tasks. That includes a brief introduction to what classification and clustering is all about, in addition to explanation of the most important terminologies and classifiers.

2.1 Short intro to classification and clustering

Classification and clustering is all about the ability to recognize and differentiate a given set of data instances. The classification task is to group every instance of the data set to its correct class by looking at the different feature variables of the instances. The method uses a supervised learning technique, which means that a labeled data set, called the training set, construct a basis of how the classifier interpret each instance. For clustering, the main target is to find similarities in the different feature variables of each instance, and group the instances based on the feature. Thus, the clustering is an unsupervised learning technique, meaning that it does not base its decisions on a training set of already labeled instances. A classification problem may appear in several kinds of structures, and a very general way of looking at different kinds of these structures are well illustrated in fig. 1, stolen from [5]. It shows two different classes, ω_1 and ω_2, consisting of features $[x_1 x_2]^T$, which exists together in three different ways: linearly separable, non-linear separable and non-separable. The task is to correctly classify the instances of the data set to either ω_1 or ω_2.

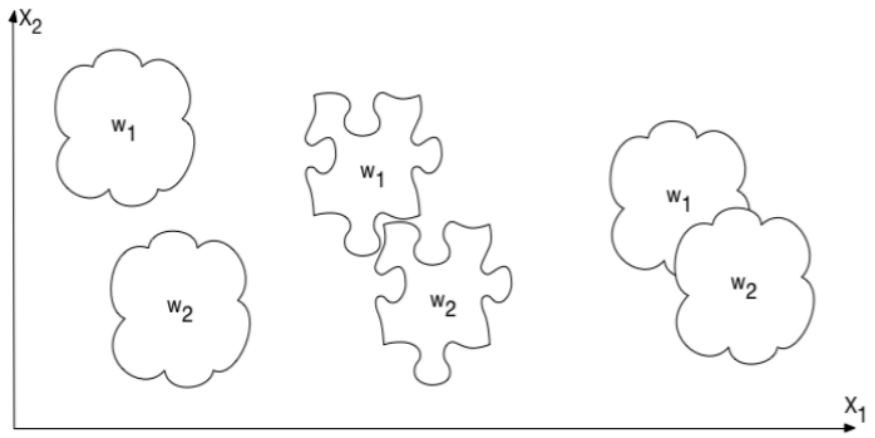


Figure 1: The two different classes ω_1 and ω_2 are, from left to right, illustrated as linearly separable, non-linearly separable and non-separable

2.2 Terminology

Class - A group/category of instances which are closely related by the similarities of their feature(s). A class has specified value boundaries on its features, defining whether or not an instance is a part of the class.

Feature - A variable of an instance in a data set. For example weight and height of a person.

Training/design set - A labeled data set used to teach the classifier which class instances with different combinations of feature values belongs to.

Test/evaluation set - A data set used for testing the classifier. The test set is independent of the training set, but follows the same probability distribution.

Validation set - A data set that is evaluated with respect to performance. It indicates whether or not to continue the training of the classifier.

Error rate - The error rate is defined for both the training- and test set as the estimated error rate, respectively denoted EER_D and EER_T. $EER_D = \frac{E_D}{N_D}$ and $EER_T = \frac{E_T}{N_T}$ is often used as the estimators, where E_D and E_T are the numbers of classification errors, while N_D and N_T are the total samples in each data set. The true (unknown) error rate is not possible to estimate, but $|EER_T - EER_D|$ gives a good indication of the performance difference between the test set and the real data.

Performance - Basically how well the classifier evaluates the different instances of a data set. It is directly connected to the error rate of the classifier. Low error rate $\implies$ good performance.

Discriminant - A distinguishing feature or characteristic.

Confusion matrix - A matrix that monitors both performance for each individual class and the misclassification between different classes. It consists of rows, indicating the true class, and columns, indicating how a data set was classified. From the confusion matrix one can calculate the total estimated error rate of the test set as well as the individual error rate for each class. table 1 shows an example of a confusion matrix.

True class \ Classified	Class 1	Class 2	Class 3
Class 1	27	3	0
Class 2	6	24	0
Class 3	0	0	30

Table 1: Confusion matrix showing three classes with 30 samples per class. Class 1 and 2 are seemingly confusable

2.3 Some useful classifiers

Linear classifier:

A linear classifier evaluates each samples of a data set based on a linear combination of the features. There are two basic models to use for linear classifiers; Generative and discriminative. We will use a discriminative model.

The discriminative model, as explained in [5], uses a discriminant function $g_i(x)$ and the decision rule

$$x \in w_j \Leftrightarrow g_j(x) = \max_i g_i(x) \tag{1}$$

The linear discriminant classifier is defined by the function

$$g_i(x) = w_i^T x + w_{io} \quad i = 1, ..., C \tag{2}$$

where w_{io} is offset for class w_i.

For $C > 2$ one can define a matrix form

$$g = Wx + w_o \tag{3}$$

where g and w_o are vectors of dimension C (number of classes) and W is a CxD matrix.

Perceptron is an example of a discriminative linear binary classifier, which classifies an instance of a data set based on which side of a given linear hyperplane the instance lies. By training the perceptron coefficients, the hypeplane will divide the two classes more and more correctly. Such a hyperplane is illustrated in fig. 2. To classify more than 2 classes at once, a multi-class perceptron might be used, where plural decision hyperplanes are used.

Figure 2: The left graph shows two classes divided by a decision border, while the right graph shows multi-class decision borders dividing three classes. Both graphs are examples of a trained perceptron classifier

The training of the perceptron and linear classifiers in general utilize different kinds of optimization algorithms, such as gradient descent and Newton methods. We will use MSE (minimum square error) based training, where the error equals the difference between the output of the discriminant functions and the corresponding labels, t_k. The discriminant $g = Wx + w_o$ is rewritten as $g = [Ww_o][x^T 1]^T$, and the input and matrix is redefined as $[x^T 1]^T \rightarrow x$ and $[Ww_o] \rightarrow W$, resulting in $g = Wx$, which is used in MSE

$$MSE = \frac{1}{2}\sum_{k=1}^{N}(g_k - t_k)^T(g_k - t_k) \tag{4}$$

The resulting optimal weights in W during training is attained by finding $\nabla_W MSE$ (gradient of MSE w.r.t W), scale it with a chosen step factor α and subtract from the old weights, as shown in eq. (5)

$$W(m) = W(m-1) - \alpha \nabla_W MSE \tag{5}$$

where m is iteration number

Template based classifier:

A template based classifier uses a template/reference to decide which class a data set sample belongs to. There exists different methods to choose the templates, where two of them will be used in the project

- Use the whole training set as templates
- Use clustering to divide the training set into a number of templates for each class, as shown fig. 3

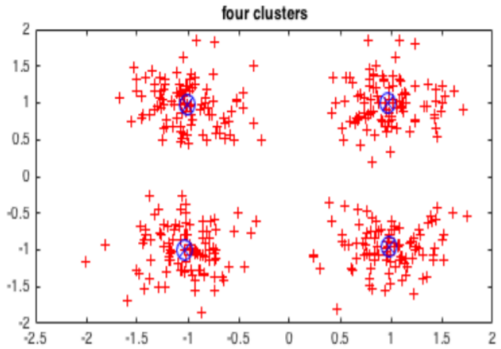


Figure 3: An example of 4 templates made by clustering of a training set. One template assigned to each class

There exists several decision methods/classifiers to evaluate a data sample using the templates, where we will use "nearest neighbour - NN". This method builds on assigning the sample x to the same class as the nearest template, as explained in [5]. We will also use the more advanced method KNN, which is based on a majority vote within the $K > 1$ nearest neighbours. If several classes end up with the same number, the closest class among them is chosen, i.e one revert to 1-NN to decide.

As stated in [5] , one can assume $k = 1, ..., M_i$ references ref_{ik} for class ω_i, i.e a total of $M = \sum_i M_i$ references. The most general reference set is given by $ref_{ik} = (\mu_{ik}, \Sigma_{ik}) \quad k = 1, ..., M_i$; for each class $\omega_i \quad i = 1, ...C$. We will use the Euclidian distance as our decision rule, which is

$$d(x, ref_{ik}) = (x - \mu_{ik})^T \Sigma_{ik} (x - \mu_{ik}) \tag{6}$$

where $ref_{ik} = \mu_{ik}$ is chosen directly from the training set, and $\Sigma_{ik} \to I$ (identity matrix)

3 Task

At the beginning of this section we will give a short description of the different topics we could choose from for our project. Then we will explain our choice of topic, and give a deeper description of the tasks of that topic.

3.1 Description of project topics

For the project we got to choose from five different topics

1. *Estimation - Maximum likelihood estimator*
2. *Estimation - BLUE for high SNRs*
3. *Detection - Spectrum sensing in OFDM Cognitive Radios*
4. *Classification - Iris and handwritten numbers 0-9*
5. *Classification - Iris and pronounced vowels*

Both the estimation projects (1 and 2) address the problem of estimating a complex exponential embedded in white, complex Gaussian noise

$$x(t) = Ae^{i(w_0 t+\phi)} + w(t) \tag{7}$$

where the projects use different estimation methods, respectively MLE and BLUE, to attain an estimate of the signal in eq. (7).

The detection project (3) addresses the problem of making secondary users (SU) not interfering with primary users (PU) in a cognitive radio system. Hence the task is to develop suitable models that will be subsequently used for spectrum sensing, and then use a detector so that the SU know when the spectrum is idle.

The classification projects (4 and 5) address the problem of assigning a data set to its correct class. Both projects share the task of classifying different Iris flowers, focusing on

- design/training and generalization
- feature and linear separability

and then they split up in two different tasks

- Task 2 of project 4 address a problem of classifying handwritten numbers, focusing on using variants of a nearest neighbourhood classifier
- Task 2 of project 5 address a problem of classifying pronounced vowels, fousing on using Gaussians as models

3.2 Choice of topic

We chose to work on project 4. *Classification - Iris and handwritten numbers 0-9.* The reason is that we wanted to learn more about classification through a rather more practical approach than just reading about methods and explanations of how it works. Classification is also closely connected to interesting fields as machine learning and AI, which is a huge motivation for learning more about the topic. Moreover, since we felt that we had a visual interpretation of the handwritten numbers exercise, we found it more intuitive to work with. It is easy to compare the performance of this classification exercise to our human intelligence and we therefore hope that doing this exercise will provide an understandable basis for the machine learning branch.

3.3 Project 4. Classification - Iris and handwritten numbers 0-9

The first task of our project concerns the classification of three different Iris flowers; Setosa, Versicolor and Virginica. We are provided with a database, consisting of 50 samples of each of the three classes. Each sample consists of 4 features, which is the length and width of the large(sepal) and small(petal) leaves of the flowers. These data samples will be used in two ways, as mentioned in the short description

1. The first part has focus on design/training and generalization
   (a) Choose the first 30 samples for training and the last 20 samples for testing
   (b) Train a linear classifier. Tune step factor until the training converge
   (c) Find the confusion matrix and the error rate for both the training and the test set
   (d) Use the last 30 samples for training and the first 20 samples for test. Repeat the training and test phases for this case
   (e) Compare the results for the two cases and comment
2. The second part has focus on features and linear separability. In this part the first 30 samples are used for training and the last 20 samples for testing
   (a) Produce histograms for each feature and class. Take away the feature which shows most overlap between the classes. Train and test a classifier with the remaining three features
   (b) Repeat the experiment above with respectively two and one features
   (c) Compare the confusion matrices and the error rates for the four experiments. Comment on the property of the features with respect to linear separability both as a whole and for the three separate classes

The second task concerns the classification of handwritten numbers 0-9. We are provided with a database of 60000 training examples written by 250 different persons and 10000 test examples written by 250 other persons. As mentioned in the short description, this task is also split into two parts, both using variants of a nearest neighbourhood classifier

1. In the first part the whole training set shall be used as templates
   (a) Design a NN-based classifier using the Euclidian distance. Find the confusion matrix and the error rate for the test set.
   (b) Plot some of the misclassified pixtures
   (c) Also plot some correctly classified pixtures. Do you as a human disagree with the classifier for some of the correct/incorrect plots?
2. In the second part you shall use clustering to produce a small(er) set of templates for each class
   (a) Perform clustering of the 6000 training vectors for each class into M = 64 clusters
   (b) Find the confusion matrix and the error rate for the NN classifier using these M = 64 templates pr class. Comment on the processing time and the performance relatively to using all training vectors as templates.
   (c) Now design a KNN classifier with K=7. Find the confusion matrix and the error rate and compare to the two other systems.

4 Implementation and results

In this section we will present the matlab implementations and discuss its corresponding results.

4.1 The Iris task

In this task, we started by designing a linear classifier. We decided to implement the perceptron algorithm. This is originally a binary classifier, differing the two classes by a hyperplane where its dimension is equal to the number of features. As described in the theory part, we scaled this to a multiclass classifier by making more hyperplanes. Since we have three classes, we trained three hyperplanes, where each hyperplane separated one class from the rest. A test sample would then be classified by the decision rule in (1), where the weights in the linear discriminant function are indeed equal to the trained weights from three hyperplanes.

We start the algorithm by assuming a zero weight matrix in our hyperplane. We multiplied the weight matrix with the feature points for each training sample and inserted these elements into the sigmoid function, giving us a decimal number between 0 and 1. To train our weight matrices, we aimed to minimize the minimum square error (MSE), where the error is the difference between the output from the sigmoid function and the correct class labels, which is either of value 0 or 1. Thereafter, we calculated the gradient of the MSE with respect to the weight matrix each iteration. We subtracted this gradient times the learning rate α for each iteration until the MSE converged or the number of iterations got significantly large (we terminated after 10000 iterations). After experimenting with different learning rates, we found that a value of 0.01 was suitable for our case. A larger learning rate would cause too large oscillations around the correct values, making it hard to reach some satisfactory coefficients precisely. On the other hand, a smaller learning rate did not provide a sufficiently large improvement for each iteration, such that we would need too many iterations to reach our goal. In order to detect convergence, we compared the MSE to a small constant $\epsilon = 0.01$, and terminated once the MSE was smaller than this threshold. ϵ can not be made as small as you want because this could cause a "false accuracy". The classifier learns to adapt to the training set more than to the general case, preventing the classifier from generalizing. In practice this would be of less significance if we had a huge, representative selection of training set. However, this is not the case with only 30 training sets for each class.

With trained coefficients, we were ready to test our classifier, which was done by simply finding the maximum value for each sample inserted in the discriminant function (2) with the three different weights. The class that are separated in the weight matrix that holds this maximum value will be the label of this test sample, which provides a fairly simple decision rule.

In subtask 1 we used the 30 first samples of the data sets for each class as a training set, and the remaining 20 samples as a test set. Then, we chose to train a multi-class perceptron as our linear classifier, tested it, and attained the confusion matrices and estimated error rates, shown in tables 2 and 3

Classified / True class	Class 1	Class 2	Class 3	EER_T	Hit-ratio
Class 1	30	0	0	0.00%	100.00%
Class 2	0	29	1	3.33%	96.67%
Class 3	0	0	30	0.00%	100.00%
				1.11%	98.89%

Table 2: Confusion matrix, EER_T and hit-ratio of the training set. The total EER_T and hit-ratio of the classifier are shown in the grey cells

Classified / True class	Class 1	Class 2	Class 3	EER_T	Hit-ratio
Class 1	20	0	0	0.00%	100.00%
Class 2	0	18	2	10.00%	90.00%
Class 3	0	0	20	0.00%	100.00%
				3.33%	96.67%

Table 3: Confusion matrix, EER_T and hit-ratio of the test set. The total EER_T and hit-ratio of the classifier are shown in the grey cells

Subsequently we used the last 30 samples for training and the 20 first for testing, and attained the confusion matrices and estimated error rates, shown in tables 4 and 5

Classified / True class	Class 1	Class 2	Class 3	EER_T	Hit-ratio
Class 1	30	0	0	0.00%	100.00%
Class 2	0	28	2	6.67%	93.33%
Class 3	0	3	27	10.00%	90.00%
				5.56%	94.44%

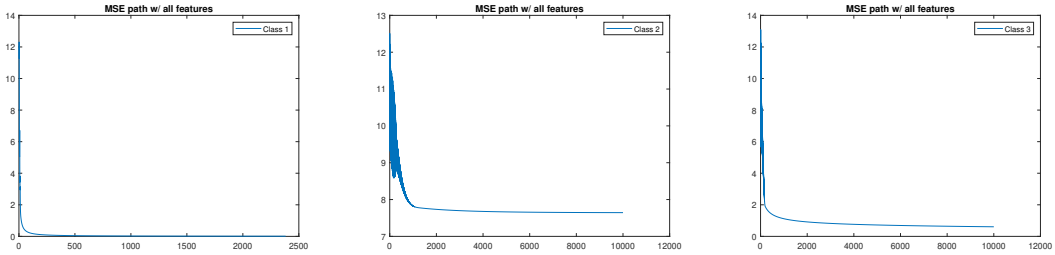
Table 4: Confusion matrix, EER_T and hit-ratio of the training set. The total EER_T and hit-ratio of the classifier are shown in the grey cells

Classified / True class	Class 1	Class 2	Class 3	EER_T	Hit-ratio
Class 1	20	0	0	0.00%	100.00%
Class 2	0	20	0	0.00%	100.00%
Class 3	0	0	20	0.00%	100.00%
				0.00%	100.00%

Table 5: Confusion matrix, EER_T and hit-ratio of the test set. The total EER_T and hit-ratio of the classifier are shown in the grey cells

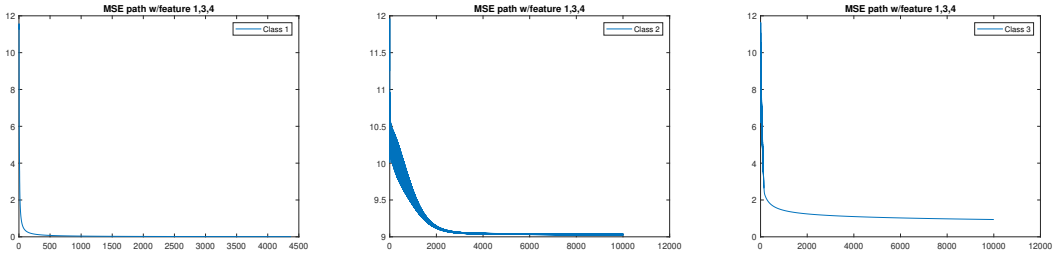
We see that the confusion matrices and error rates of the two differently trained classifiers have small differences. This makes sense due to the fact that a classifier depends on its training. If two classifiers are structured the same but trained with different samples they will acquire a different performance, though the difference should not be significant. Furthermore, by inspection of the confusion matrices we discover that class 1 is classified correctly every time, and that it is some minor confusion between class 2 and 3. This may indicate that class 1 is linearly separable from the other two in one or more features, which is tested for in subtask 2. This theory is additionally strengthened by the fact that class 1 was the only class which MSE seemed

to converge with our choice of $\epsilon = 0.01$. The two other classes did not provide a convergence in MSE, and the final MSE of class 2 varied between $7.8 - 9.7$, while the final MSE of class 3 varied between $1 - 2.5$. However, the classification between class 2 and 3 are quite robust as well.



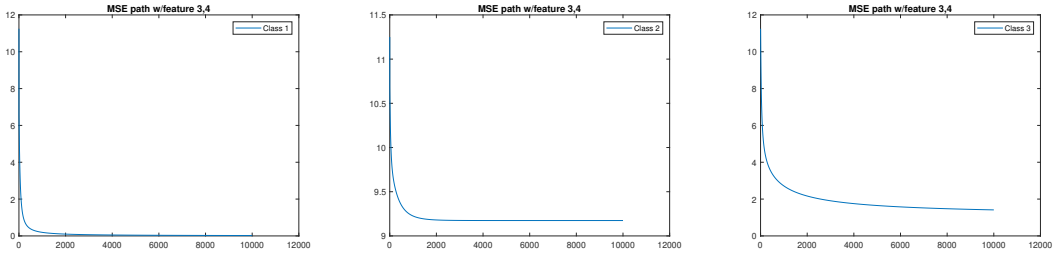
a) b) c)

Figure 4: MSE path for each class w/all features used. a) converges in 2371 iterations, b) doesn't converge and ends up with approx. $MSE = 7.8$, c) doesn't converge and ends up with approx. $MSE = 1$



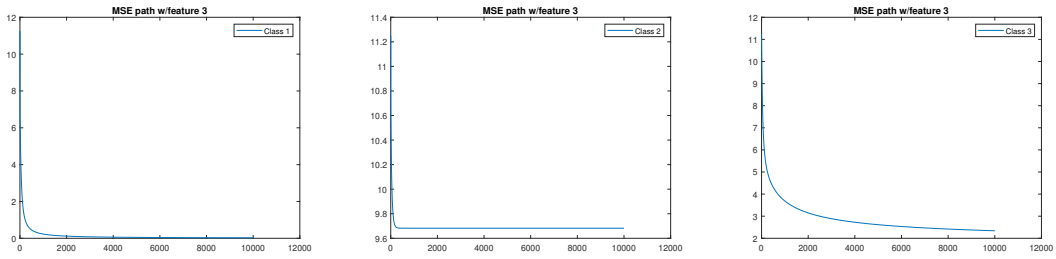
a) b) c)

Figure 5: MSE path for each class w/feature 1, 3 and 4 used. a) converges in 2371 iterations,b) doesn't converge and ends up with approx. $MSE = 9$, c) doesn't converge and ends up with approx. $MSE = 1.75$



a) b) c)

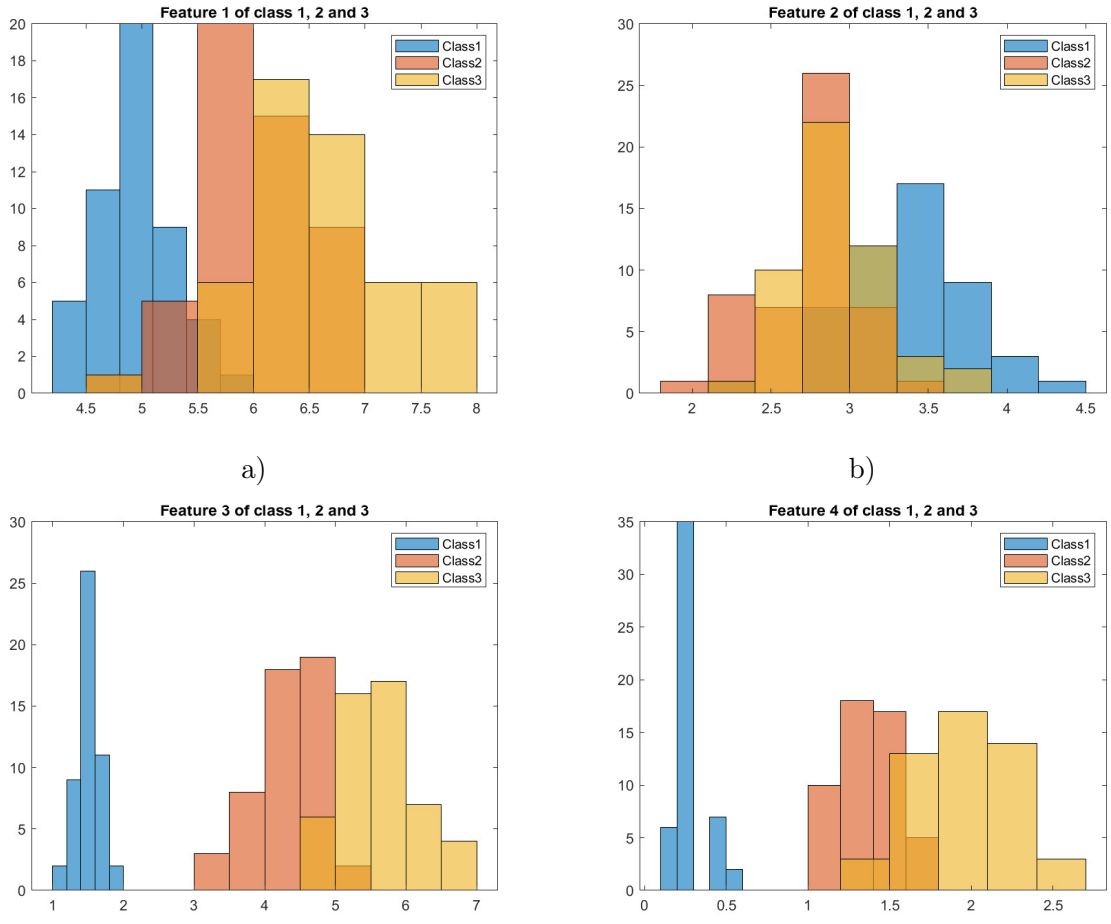
Figure 6: MSE path for each class w/feature 3 and 4 used. a) doesn't converge and ends up with approx. $MSE = 0.02$, b) doesn't converge and ends up with approx. $MSE = 9.25$, c) doesn't converge and ends up with approx. $MSE = 1.9$



a) b) c)

Figure 7: MSE path for each class w/feature 3 used. a) doesn't converge and ends up with approx. $MSE = 0.025$, b) doesn't converge and ends up with approx. $MSE = 9.7$, c) doesn't converge and ends up with approx. $MSE = 2.5$

In subtask 2 we used the first 30 samples for training and the last 20 for testing. We made histograms illustrating the degree of linear separability occuring in the 4 different features, shown in fig. 8



a) b) c) d)

Figure 8: Histograms of the 4 different features of the 3 classes

We see that the linearly separability of the different features, rated from the most separable to least separable, is $3 \rightarrow 4 \rightarrow 1 \rightarrow 2$. In subtask 1 we trained the classifier by using all the features. When removing feature 2 and train the classifier again, using only feature 1,3 and 4, we get the confusion matrices and estimated error rates shown in respectively tables 6 and 7

Classified / True class	Class 1	Class 2	Class 3	EER_T	Hit-ratio
Class 1	30	0	0	0.00%	100.00%
Class 2	0	28	2	6.67%	93.33%
Class 3	0	0	30	0.00%	100.00%
				2.22%	97.78%

Table 6: Confusion matrix, EER_T and hit-ratio of the training set. The total EER_T and hit-ratio of the classifier are shown in the grey cells

Classified / True class	Class 1	Class 2	Class 3	EER_T	Hit-ratio
Class 1	20	0	0	0.00%	100.00%
Class 2	0	18	2	10.00%	90.00%
Class 3	0	0	20	0.00%	100.00%
				3.33%	96.67%

Table 7: Confusion matrix, EER_T and hit-ratio of the test set. The total EER_T and hit-ratio of the classifier are shown in the grey cells

Subsequently also removing feature 1 and train the classifier, using only feature 3 and 4, we get the confusion matrices and estimated error rates shown in respectively tables 8 and 9

Classified / True class	Class 1	Class 2	Class 3	EER_T	Hit-ratio
Class 1	30	0	0	0.00%	100.00%
Class 2	0	28	2	6.67%	93.33%
Class 3	0	2	28	6.67%	93.33%
				4.44%	95.56%

Table 8: Confusion matrix, EER_T and hit-ratio of the training set. The total EER_T and hit-ratio of the classifier are shown in the grey cells

Classified / True class	Class 1	Class 2	Class 3	EER_T	Hit-ratio
Class 1	20	0	0	0.00%	100.00%
Class 2	0	20	0	0.00%	100.00%
Class 3	0	2	18	10.00%	90.00%
				3.33%	96.67%

Table 9: Confusion matrix, EER_T and hit-ratio of the test set. The total EER_T and hit-ratio of the classifier are shown in the grey cells

Finally, removing feature 4 and train the classifier with only feature 3, we get the confusion matrices and estimated error rates shown in respectively tables 10 and 11

True class \ Classified	Class 1	Class 2	Class 3	EER_T	Hit-ratio
Class 1	30	0	0	0.00%	100.00%
Class 2	0	25	5	16.67%	83.33%
Class 3	0	1	29	3.33%	96.67%
				6.67%	93.33%

Table 10: Confusion matrix, EER_T and hit-ratio of the training set. The total EER_T and hit-ratio of the classifier are shown in the grey cells

True class \ Classified	Class 1	Class 2	Class 3	EER_T	Hit-ratio
Class 1	20	0	0	0.00%	100.00%
Class 2	0	19	1	5.00%	95.00%
Class 3	0	0	20	0.00%	100.00%
				1.67%	98.33%

Table 11: Confusion matrix, EER_T and hit-ratio of the test set. The total EER_T and hit-ratio of the classifier are shown in the grey cells

Comparison of the four different experiments' confusion matrices and estimated error rates w.r.t training shows that the more features evaluated results in a better performance. It also shows that the classifier has a 0% error rate on class 1 every time. This makes sense due to the fact that class 1 is linearly separable in both feature 3 and 4. However, the classifier trained with only feature 3 has the worst total design performance, p_D. The reason is that feature 3 is not linearly separable in class 2 and 3. The other training sets, consisting of several features, make the classifier attain a better total design performance due to giving it more parameters to distinguish between each class.

Looking at the confusion matrices and estimated error rates w.r.t testing we see that it still has 0% estimated error rate on class 1 every time, which one could expect due to the 0% estimated error rate in the training set. The test set differ from the training set in that the testing with only feature 3 has the lowest total $EER_T = \frac{1}{60}$, while the other experiments has a total $EER_T = \frac{2}{60}$. Similarly to the training set, the test set's confusion matrices and EER_T's shows that the classifier has minor distinguishing problems related to class 2 and 3. Again, they are not linearly separable in any of the features, which makes it difficult for a linear classifier, as the perceptron, to evaluate them flawlessly.

4.2 Classification of handwritten numbers 0-9

In subtask 1 of the classification of handwritten numbers 0-9 we used all the training samples as templates. Furthermore we designed an k-nearest-neighbour classifier to evaluate the test samples. This algorithm calculates the euclidean distance (from (6)) from every single test sample to every template sample, and finds the k shortest distances and their corresponding label. The majority of labels within these k shortest distances will be the label for each test sample. With 60000 templates, 10000 test samples and 784 features between 0 and 255, this will obviously be a lot of computations to deal with for the computer. Therefore, the test set got divided into 1000 chunks with 10 pixtures in each chunk to be computed and classified, such that the matrix dimensions in the euclidean distances got smaller. Still, this algorithm used some minutes to complete on such large data sets. On the other hand, one would expect large accuracy due to all this available data.

As a result, using $k = 1$ and hence having the nearest-neighbour algorithm (since this is a special case of k-nearest-neighbours algorithm), we attained the confusion matrix and estimated error rates as shown in table 12. From the individual error rates one observes that class 2 (digit 1) is recognized the most. This makes sense due to the distinct form of 1 w.r.t the other digits. As expected, we achieved a very low error rate at only 3.09%. The running time was 35 minutes and 12 seconds.

True class \ Classified	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	EER_T	Hit-ratio
Class 1	973	1	1	0	0	1	3	1	0	0	0.71%	99.29%
Class 2	0	1129	3	0	1	1	1	0	0	0	0.53%	99.47%
Class 3	7	6	992	5	1	0	2	16	3	0	3.88%	96.12%
Class 4	0	1	2	970	1	19	0	7	7	3	3.96%	96.04%
Class 5	0	7	0	0	944	0	3	5	1	22	3.87%	96.13%
Class 6	1	1	0	12	2	860	5	1	6	4	3.59%	96.41%
Class 7	4	2	0	0	3	5	944	0	0	0	1.46%	98.54%
Class 8	0	14	6	2	4	0	0	992	0	10	3.50%	96.50%
Class 9	6	1	3	14	5	13	3	4	920	5	5.54%	94.46%
Class 10	2	5	1	6	10	5	1	11	1	967	4.16%	95.84%
											3.09%	96.91%

Table 12: Confusion matrix, EER_T and hit-ratio of the test set. The total EER_T and hit-ratio of the classifier are shown in the grey cells. Class 1-10 map to digit 0-9

figs. 9 and 10 shows one sample of each digit which is correctly classified and misclassified, respectively.

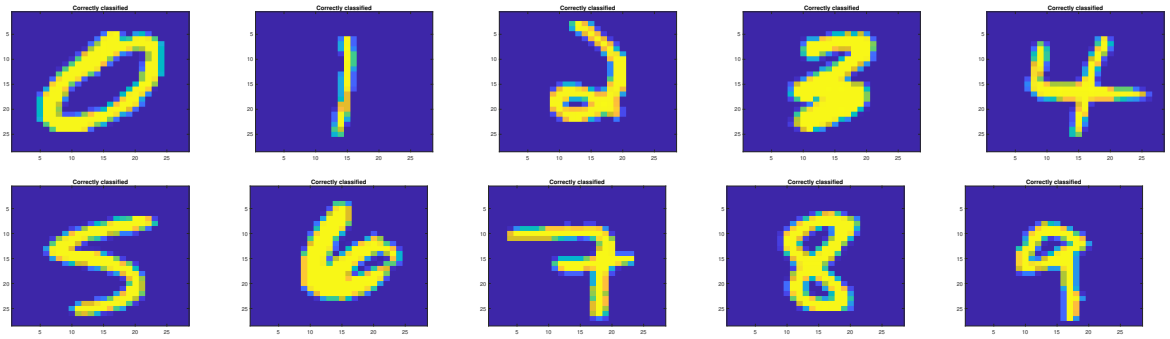
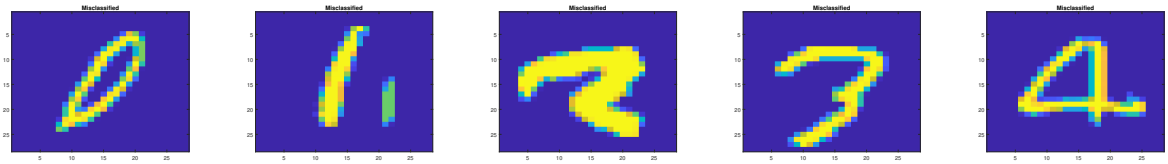
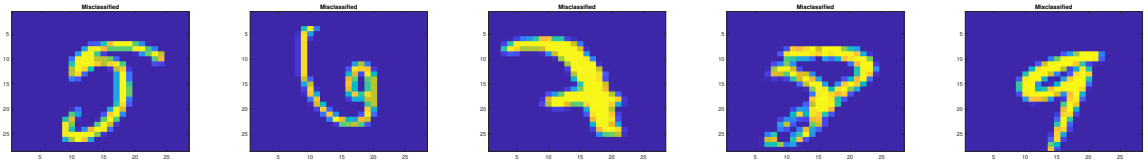


Figure 9: Correctly classified pixtures



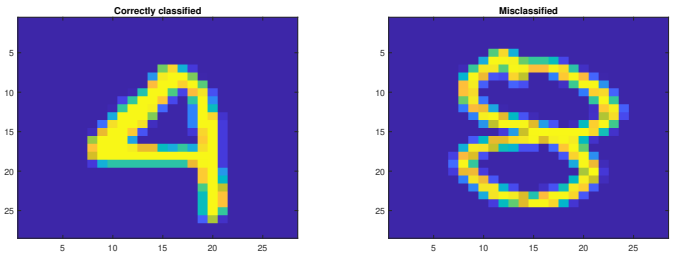
a) 0 as 6 b) 1 as 6 c) 2 as 7 d) 3 as 7 e) 4 as 9



f) 5 as 3 g) 6 as 4 h) 7 as 3 i) 8 as 9 j) 9 as 4

Figure 10: Misclassified pixtures

Some of the correctly classified and misclassified samples are classified differently than a human potentially would do, shown in fig. 11. From our point of view, pixture a) in fig. 11 could easily been seen as a 9 as well as 4. In pixture b) one clearly see that the number should have been classified as 8, indicating the difference between the human mind and the classifier.



a) b)

Figure 11: These classified samples potentially differ from a human point of view. a) is correctly classified as 4, while b) is misclassified as 5

In subtask 2 we created $M = 64$ templates for each class, based on clustering of the approximately 6000 training vectors per class. To achieve this we used the Matlab command `kmeans(trainv_i, M)`, which clusters training vectors of a class ω_i into M templates. This resulted in a faster classifier than the classifier from subtask 1, since we now only deal with a total of 640 templates, which is significantly less than 60000. This would drastically reduce our computational time, and by running this program we reduced the running time to 69 seconds. The confusion matrix and estimated error rates of the test set are now as shown in table 13. We observe that the accuracy got a bit lower, with an error rate of 4.64%. We would argue that this is not a very significant accuracy decrease, in particular when considered the drastically lower computational time. The low accuracy decrease can partly be caused by that we still make use of every 60000 training data, but use them in a more efficient manner to get an acceptable computational time. Therefore this trade off seems to be reasonable for many purposes. An example of usage where this simplification could come advantageous is on several calculator apps on mobiles that allows the user to write in math expressions "by hand", such that the program can interpret the expression and return the answer. In such cases, low computational time is obviously important for the user experience.

Classified / True class	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	EER_T	Hit-ratio
Class 1	962	1	4	1	1	3	5	1	1	1	1.84%	98.16%
Class 2	0	1131	1	0	0	0	1	0	1	1	0.35%	99.65%
Class 3	9	5	980	6	2	0	2	14	14	0	5.04%	94.96%
Class 4	0	1	5	949	1	25	0	8	14	7	6.04%	93.96%
Class 5	1	9	1	0	927	0	6	6	2	30	5.60%	94.40%
Class 6	6	1	0	17	2	841	14	1	8	2	5.72%	94.28%
Class 7	5	4	2	0	5	2	937	0	2	1	2.19%	97.81%
Class 8	1	14	8	0	7	2	0	969	0	27	5.74%	94.26%
Class 9	4	0	3	13	3	23	4	5	913	6	6.26%	93.74%
Class 10	4	5	5	7	29	5	2	19	6	927	8.13%	91.87%
											4.64%	95.36%

Table 13: Confusion matrix, EER_T and hit-ratio of the test set. The total EER_T and hit-ratio of the classifier are shown in the grey cells. Class 1-10 map to digit 0-9

We also designed a KNN-classifier with $k = 7$, which we tested the same way as the 1-NN classifier with clustering. It resulted in the confusion matrix and estimated error rates shown in table 14. We experienced a running time of 69 seconds.

True class \ Classified	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10	EER_T	Hit-ratio
Class 1	955	1	3	1	1	6	10	1	2	0	2.55%	97.45%
Class 2	0	1129	2	1	0	0	2	0	1	0	0.53%	99.47%
Class 3	12	15	947	10	4	1	4	13	26	0	8.24%	91.76%
Class 4	0	5	8	946	1	22	0	10	14	4	6.34%	93.66%
Class 5	0	13	3	0	900	0	12	2	3	49	8.35%	91.65%
Class 6	4	4	2	27	7	828	7	2	7	4	7.17%	92.83%
Class 7	8	4	4	0	11	10	918	0	3	0	4.18%	95.82%
Class 8	0	38	13	1	14	0	0	925	3	34	10.02%	89.98%
Class 9	5	2	4	23	10	29	4	7	885	5	9.14%	90.86%
Class 10	6	11	4	8	27	5	1	26	8	913	9.51%	90.49%
											6.54%	93.46%

Table 14: Confusion matrix, EER_T and hit-ratio of the test set. The total EER_T and hit-ratio of the classifier are shown in the grey cells. Class 1-10 map to digit 0-9

The computational time was the same as earlier, because of the clustering algorithm that ensures that the number of templates is only 640. Therefore, the k-nearest-neighbors algorithm uses short time to evaluate and classify all test vectors, such that the rather more advanced complexity does not affect the running time in a significant degree. The reduced accuracy was a bit surprising, since we expected the accuracy to be higher. This can be caused by the fact that our algorithm, for cases with equal amount of occurrences of two or more of majority frequent classes, picks one of these most frequent numbers arbitrary using the Matlab function `mode()`. Instead, one could for such cases go back to finding the class label for the shortest distance, i.e. use nearest neighbour. Yet, it is quite surprising that this did not increase the accuracy, and we conclude that $k = 1$ is a good choice of k for this problem.

5 Conclusion

From the Iris task we observed that our linear perceptron classifier worked really well, although the data sets were not very large. However, linear classifiers are normally not that efficient simply because most classification problems are not linearly separable. Therefore, due to this somewhat rare occurrence of a linearly separable problem, we would expect most linear classifiers to be sufficient for achieving high accuracy. As we observed, our classifier never made any mistake on class 1, which indeed was expected since this was linearly separable from the others. Class 2 and 3 were not linearly separable, and we experienced some, yet a quite small error rate between these classes. Still, the error rate was acceptable and a bit higher than expected due to our visual observations of the features in two dimensions, and this is probably caused by the fact that the classes had four features each (the more data, the better classification). Removing features also caused a decrease in the classifier performance, which supports this theory.

The classification of handwritten numbers was somewhat more comprehensive, working with larger dimensions and data sets. Moreover, the usage of a template based classifier seemed more demanding than simply separating sets by a hyperplane. Still, the nearest-neighbour classifier proved high accuracy, though with a rather poor computational time due to the large data sets. By clustering the data sets, we decreased the computational time drastically without affecting the accuracy too much. We agreed that this approach had many practical benefits, such as in applications on mobile phones and tablets, where handwritten numbers or letters can be interpreted directly by the computer. Lastly, we extended the algorithm to consider the k nearest neighbours instead of only the one nearest neighbour. Choosing $k = 7$, the algorithm proved to be a bit more poor than nearest neighbour both with respect to accuracy and performance.

References

[1] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. second. John Wiley Sons, INC., 2001.

[2] Steven M. Kay. *Fundamentals Of Statistical Signal Processing - Detection theory*. Prentice Hall, PTR, 1993.

[3] Steven M. Kay. *Fundamentals Of Statistical Signal Processing - Estimation theory*. Prentice Hall, PTR, 1993.

[4] *Lecture notes from TTK4275 Estimation, detection and classification*. `https://ntnu.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_15236_1&content_id=_881475_1`. Accessed: 2020-04-20.

[5] Tor A. Myrvoll, Stefan Werner, and Magne H. Johnsen. *Estimation, detection and classification*. Institutt for elektroniske systemer, NTNU, Trondheim.

[3] [2] [1] [5] [4]