TTT4275 Classification Project

Group 18
Even Anton Thamdrup Eliassen
Hanna Waage Hjelmeland

March 26, 2020

Summary

This is a project report in the subject of Estimation, Detection and Classification, TTT4275 (2020 SPRING) at the Norwegian University of Science and Technology. This subject combines the disciplines of estimation, detection and classification to form a solid basis for data analysis [3]. The theme for this project is the latter discipline, namely classification. While estimation and detection respectively focuses on finding the value of an input and to determine if some value is present in the input or not, classification takes on the task of assigning a label to the input and thereby categorizing it into one or more classes.

As I am sitting in my dorm writing this report, I look towards two of my plants sitting peacefully in my windowsill. The plants differ in both their trunk and leaf, which makes it a trivial task for me to look at them and immediately being able to tell you that these are two different plants in addition to labeling them by two different names. Although these small observations which takes place on a daily basis may seem trivial, in reality they are the result of years of life experience and cognitive training. In this project, different methods of building this intuition in computer code has been implemented and tested.

In the first task a classifier was trained and tested on a data set called Fisher's Iris data set. The data consists of 50 samples from each of three types of iris flower. Each of these samples contain four measurements: length and width of the petals and sepals. The goal was to make the computer label the flower samples correctly, accordingly to the three different types.

In the second part of the project, the objective was to design and train a template based classifier so that it would be able to classify handwritten numbers between 0 and 9 with good precision.

In both cases, impressive results was observed. By combining statistic approaches and programming, our own computers had enabled "their own instinct", being able to tell us what type of Iris flower we are looking at, and successfully distinguish handwritten numbers from each other and label the correctly. Even though the tasks are profoundly different, both can be solved by implementing algorithms based on a relatively basic statistical groundwork. This leaves us with a deep respect for the power of classification and how well a computer is at carrying out such tasks.

Contents

0 Introduction

This is a report of the classification project in the course Estimation, Detection and Classification, TTT4275, at the Norwegian University of Science and Technology. Classification is an important topic that finds a lot of use today. Examples of commonly used applications are computer vision, speech recognition, handwriting recognition, biometric identification and internet search engines. The goal in this project is to learn and become familiar with some basic techniques in the field of classification.

The report initiates in Part 1 with a somewhat detailed description of the theory that lay the groundwork for the classification methods used in the project. Following this, the result of using two different methods of classification is presented in Part 2 and Part 3. At last, a conclusion (4) based on the project results can be found.

All code is written in Matlab.

1 Part 1 - Theory

In this project, different parts of classification theory has been applied to two practical tasks, namely the Iris task and the Handwritten Numbers task. The theory applied is presented in this chapter.

1.1 A general introduction to classification

The concept of classification comprises the action of mapping a set of data to a set of classes. It is a widely used approach and it is performed in numerous ways, using different methods. In the subject TTT4275, the subcategory "feature based classification" is elaborated on. This category includes the problems where "[...] a sensor is measuring some values or signals which can be transformed into appropriate features suited for classification. Further, the classes are logically and scientifically defined, i.e. not subject to different humans interpretations (like 'right' and 'wrong')." [2]

In this subcategory, in order to be able to do the aforementioned mapping, the classifiers must be trained. This training is done in different ways, much dependent on the chosen classifier, but common for all methods is that a set of ***training data*** is required. That is, a set of data relevant to the problem that are pre-classified. In the Handwritten Numbers task, i. e. Part 3, this training set consists of 60 000 images of handwritten numbers, all labeled with the associated number. When the training is done, the classifier is ready for implementation. That is, the classes are predefined in order to assign the ***test data*** a class label. This way of training is called supervised learning. The set of test data is data that is not used in the training, which is used to determine different qualities of the classifier.

A classifier makes decisions on the given data, and these decisions can be right or wrong. Therefore, an important feature of classifiers is the ***error rate***, namely the percentage of cases that it wrongly classified for a given test set. This is an important indication of the robustness of the classifier.

Another key feature related to classification problems is the ***confusion matrix***, a matrix which displays how the test data have been correctly or incorrectly classified. This matrix illustrates in what way and to what extent the classes overlap. As an example, one can think of a classification experiment where images of animals are supposed to be mapped to the classes cats, dogs and birds. Say the test set were to consist of 90 images, 30 of each species. Then the associated confusion matrix could look something like

True \ Classified	Cat	Dog	Bird
Cat	25	3	0
Dog	5	27	0
Bird	0	0	30

where different observations can be made of this results, among others that birds are in no cases mistaken for cats or dogs, and cats are to some extent more frequently mistaken as dogs than the oposite.

An observation can also be made regarding the relation between the confusion matrix and the error rate: it is equal to the number of off-diagonal elements in the matrix divided by the number of objects in the test data. Thus, the error rate can be found by the following expression

$$er = \frac{n - tr(C)}{n}$$

where er is the error rate, n is the number of objects in the test data and $tr(C)$ is the trace of the confusion matrix. In the example above, the error rate equals to $\frac{8}{90} = 0.0889 = 8.89\%$.

1.2 Linear classifiers

The linear classifier is a discriminant classifier. This means that it consists of a discriminant function, $g_i(x)$, for each class. A sample is assigned to the class with the largest discriminant function for that sample. I.e. the decision rule is given by

$$x \in \omega_j \Leftrightarrow g_j(x) = \max_i g_i(x) \ , \ i = 1, ..., C \tag{1.1}$$

where ω_j is the class assigned to x, and C is the number of classes.

In the case of the linear classifier, the discriminant functions are given by

$$g_i(x) = \mathbf{w_i}^\top \mathbf{x} + w_{i0} \tag{1.2}$$

By augmenting the feature vector to be on the form $\mathbf{x} = \begin{bmatrix} x_1 & \cdots & x_n & 1 \end{bmatrix}^\top$ one can combine all the weights and the offsets, w_{io}, into a matrix, $\mathbf{W}$, as shown in eq. (1.3).

$$\mathbf{z(x)} = \begin{bmatrix} w_{1,1} & \cdots & w_{1,n} & w_{1,0} \\ \vdots & \ddots & \vdots & \vdots \\ w_{C,1} & \cdots & w_{C,n} & w_{C,0} \end{bmatrix} \mathbf{x} = \begin{bmatrix} z_1(x) \\ \vdots \\ z_C(x) \end{bmatrix} \tag{1.3}$$

To train the classifier one has to define an objective function that is to be minimized. It is common to use the mean square error, MSE:

$$MSE = \frac{1}{2} \sum_{k=1}^{N} (\mathbf{g}_k - \mathbf{t}_k)^\top (\mathbf{g}_k - \mathbf{t}_k) \tag{1.4}$$

where N is the number of training samples, $\mathbf{t}_k$ is the target value that represents the true class and each element in $\mathbf{g}_k$ is given by

$$g_{ik} = sigmoid(z_{ik}) = \frac{1}{1 + e^{-z_i k}} \quad , \quad i = 1, ..., C \tag{1.5}$$

An example of a target value would be $\mathbf{t}_k = \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}^\top$, if $\mathbf{x}_k$ belongs to the class ω_1. The sigmoid function is used to squash the elements in vector $\mathbf{z}$ so that values above zero is mapped to 1 and values below to 0. It is plotted in fig. 1.1.

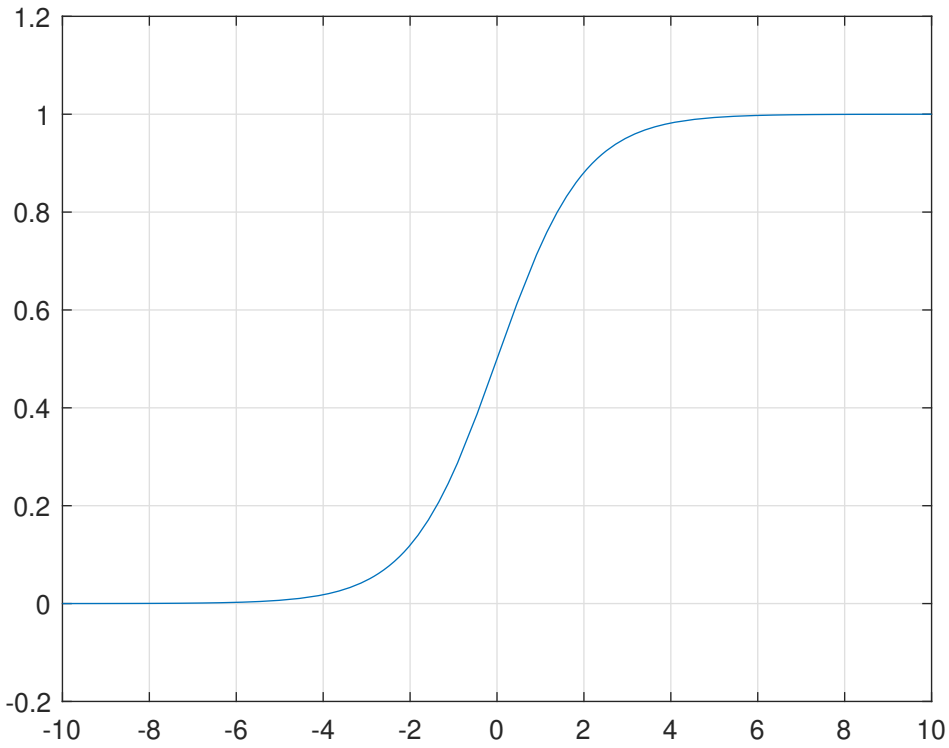

Figure 1.1: Plot of the sigmoid function which is used in the discriminant.

The structure of the discriminant functions are summarized in fig. 1.2

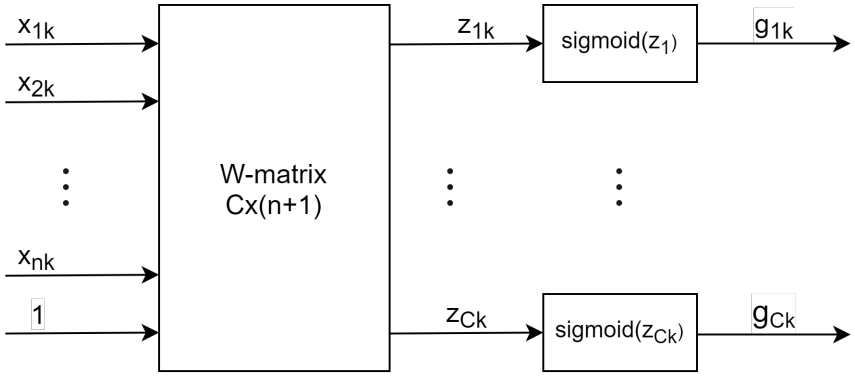

Figure 1.2: Discriminant function

In order to find an optimal set of weights, the objective function can be minimized using

gradient descent. In this method one starts with an initial guess of the weights, and take iterative steps in the steepest direction given by the negative gradient. The iterations are given by

$$W_m = W_{m-1} - \alpha \nabla_W MSE \ , \ \ \alpha \in (0,1) \tag{1.6}$$

where α decides the step length. The gradient is given by eq. (1.7), where $\circ$ means elementwise multiplication.

$$\nabla_W MSE = \sum_{k=1}^{N} [(\mathbf{g}_k - \mathbf{t}_k) \circ \mathbf{g}_k \circ (1 - \mathbf{g}_k)] \mathbf{x}_k^\top \tag{1.7}$$

1.3 Template based classifiers

The template based classifier is a classifier which uses a set of templates on the same form as the input data, and matches the input data towards these templates. This matching can be performed in several manners, but two popular choices of techniques are the Nearest Neighbour (NN) method and the K Nearest Neighbours (KNN) method. These decision methods uses a variant of a distance measure to decide which template is closest to the input, proceding to label the input with the same class label as the closest template. An example of a distance measure is the Euclidian distance [2], namely

$$d(x, \mu_{ik}) = (x - \mu_{ik})^\top (x - \mu_{ik}) \tag{1.8}$$

where x is the input and μ_{ik} is the kth template for each class i. In the NN case, the label of template closest is the one chosen, while in the KNN case, the most frequent label of the K nearest templates is the one chosen.

Another variying feature of the template based classifiers is how the templates are chosen. The simplest one of these is to directly use all the data in the training set as templates. Thus, every object that is subject to classification is compared to every object in the training set. An alternative to this is the method of ***clustering***. This is a technique which, based on the training data, constructs a cluster of a number of representative templates for each class.

A template based classifier, along with these decision methods (both NN and KNN), is applied both with and without the use of clustering in the Handwritten Numbers task, i. e. Part 3.

2 Part 2 - The Iris Task

2.1 Description

In this task a linear classifier is trained and tested on a data set called Fisher's Iris data set. The data consists of 50 samples from each of three types of iris flower. Each of these samples contain four measurements: length and width of the petals and sepals.

The goal in this task was to design and evaluate the linear classifier, before analyzing the relative importance of each of the four features with respect to linear separability.

2.2 Design and training

This subsection describes how the linear classifier was designed and trained.

First the 30 first samples from each class was set aside for training, with the last 20 for testing. A linear classifier was implemented in Matlab as described in section 1. The samples were put on the augmented form

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_4 \\ 1 \end{bmatrix} \tag{2.1}$$

and a gradient function for the MSE was made. The training then consisted of finding a set of weights that decreased the error. This was done using a self implemented gradient descent method, as described in section 1. The step length α was chosen experimentally to be $\alpha = 0.01$ for all steps. This made the algorithm converge, although slowly. It always exceeded the iteration threshold before the gradient became small enough to terminate the algorithm. Training was much quicker with the built in matlab function `fminunc`, using a quasi-newton algorithm, but this was not used in this report as it was not part of the task.

After training with the 30 first samples, the classifier was trained on the 30 last samples. Then a histogram was made for each feature and class. The features with the most overlap between classes was removed and the classifier was trained again with the 30 first samples. This was repeated until only one feature remained.

2.3 Results and discussion

A histogram showing the classes and their features is shown in fig. 2.1. The linear classifier was trained and tested using the following setup:

$\mathbf{W_0}$	α	maxiterations
0.01*`ones(3,5)`	0.01	25000

Table 1: Training setup of the classifier. 3 is the number of classes and 5 is the number of features + 1.

This resulted in the confusion matrices and error rates shown in table 2 and 3, where the first table contains results from when the classifier was trained on the 30 first samples and tested on the 20 last, and the second table from when it was trained on the 30 last and tested on the 20 first.

	Training		Testing	
Features	Confusion matrix	Error Rate	Confusion matrix	Error Rate
1, 2, 3, 4	$\begin{pmatrix} 30 & 0 & 0 \\ 0 & 29 & 1 \\ 0 & 0 & 30 \end{pmatrix}$	0.0111	$\begin{pmatrix} 20 & 0 & 0 \\ 0 & 18 & 2 \\ 0 & 1 & 19 \end{pmatrix}$	0.05
1, 3, 4	$\begin{pmatrix} 30 & 0 & 0 \\ 0 & 29 & 1 \\ 0 & 0 & 30 \end{pmatrix}$	0.0111	$\begin{pmatrix} 20 & 0 & 0 \\ 0 & 18 & 2 \\ 0 & 1 & 19 \end{pmatrix}$	0.05
3, 4	$\begin{pmatrix} 30 & 0 & 0 \\ 0 & 28 & 2 \\ 0 & 1 & 29 \end{pmatrix}$	0.0333	$\begin{pmatrix} 20 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 2 & 18 \end{pmatrix}$	0.0333
3	$\begin{pmatrix} 30 & 0 & 0 \\ 0 & 25 & 5 \\ 0 & 1 & 29 \end{pmatrix}$	0.0667	$\begin{pmatrix} 20 & 0 & 0 \\ 0 & 19 & 1 \\ 0 & 0 & 20 \end{pmatrix}$	0.0167

Table 2: Confusion matrices and error rates when training set consists of the 30 first samples and testing the 20 last.

	Confusion matrix	Error Rate
Training	$\begin{pmatrix} 30 & 0 & 0 \\ 0 & 28 & 2 \\ 0 & 1 & 29 \end{pmatrix}$	0.0333
Testing	$\begin{pmatrix} 20 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 20 \end{pmatrix}$	0

Table 3: Confusion matrices and error rates when training set consists of the 30 last samples and testing the 20 first. All four features are used.

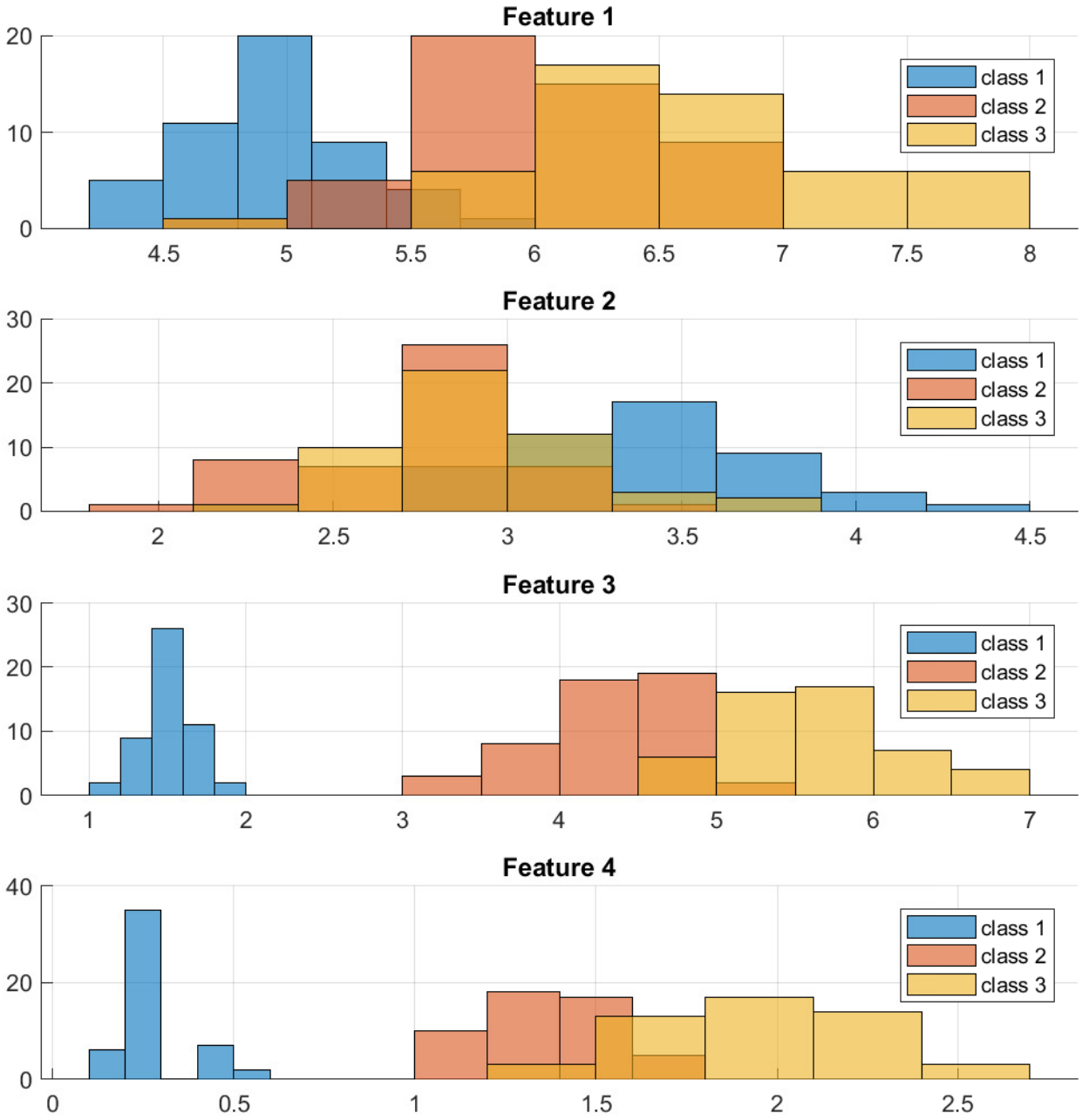

Figure 2.1: Histogram showing how each feature is distributed for each class.

Comparing the two cases where all four features are used in table 2 and 3, one can see a slight difference in the result. This is expected, as the classifier is dependent on the training data, and the test on the test data. In the first case, where training was done on the first 30 samples, only one sample was miss-classified during training. A class 2 sample was assigned to class 3. In the second case there were three miss-classified samples during training. Here the training set consisted of the 30 last samples. If one now looks at the corresponding error rates during testing, one can see that the error rate during training does not correctly predict the error rate during

testing. In these results the first case produced a higher error rate than the second, even though it had a lower error rate during training.

From the histogram in fig. 2.1 one can see that there is most overlap between classes in feature 2. It was therefore decided to train the classifier without this feature. After removing feature 2, feature 1 had the most overlap, and the classifier was trained with only feature 3 and 4. Lastly feature 4 was removed and the classifier trained again. The result of this process can be seen in table 2.

Removing feature 2 gave identical results to including all features. This makes sense, as feature 2 has quite a bit more overlap than the rest. Removing feature 1 gave worse training error rate, but better test error rate. The same can be seen when removing feature 4. The increasing error rate when training with fewer features is as expected. The test error rate decreasing is probably because the test data coincidentally fit better with these decision rules.

If a problem is linearly separable, it should be possible to correctly classify all samples with a linear classifier. The Iris data set is almost linearly separable. As the histogram shows, there is not much overlap between class 1 and the other classes. This suggests that a linear classifier should be able to correctly classify all samples of this class. In fig. 2.2 feature 1 is plotted against feature 2 for each class. Here one can see that it is possible to separate class one from the others with a straight line, while the other two classes are quite overlapping. This is reflected in the confusion matrices, where all class 1 samples are correctly classified and the only confusion is between class 2 and 3. In fig. 2.3 feature 3 is plotted against feature 4 for each class. Comparing this to fig. 2.2 and fig. 2.1 one can clearly see that if one has to choose only two features, the pair (3,4) gives a much more linearly separable problem than (1,2). A benefit of using fewer features is faster training.

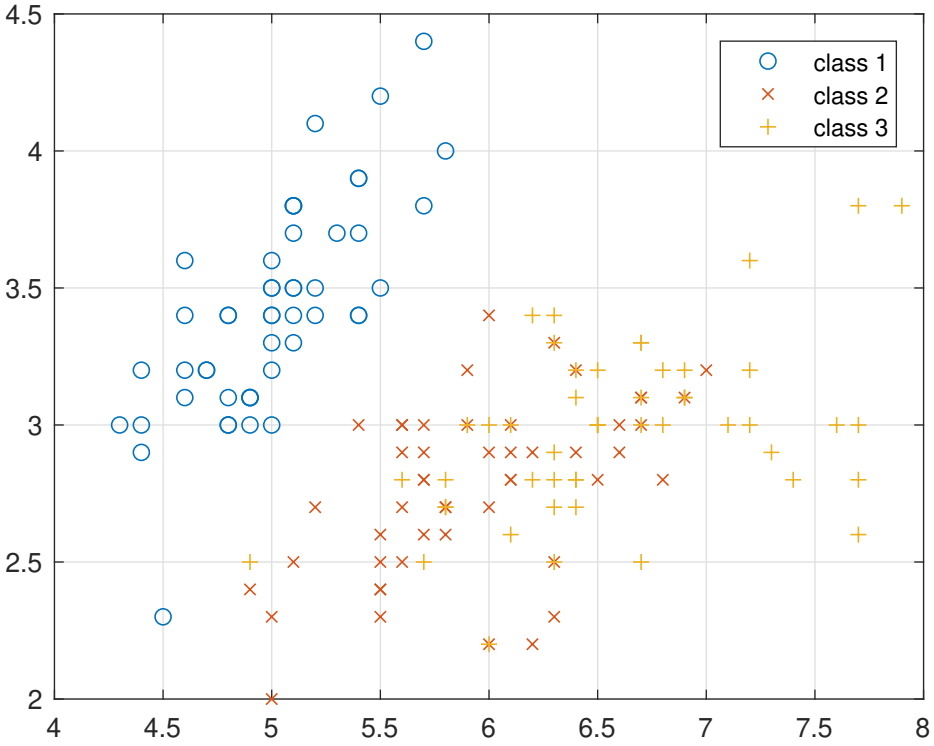

Figure 2.2: Feature 1 and 2 plotted against each other for each class.

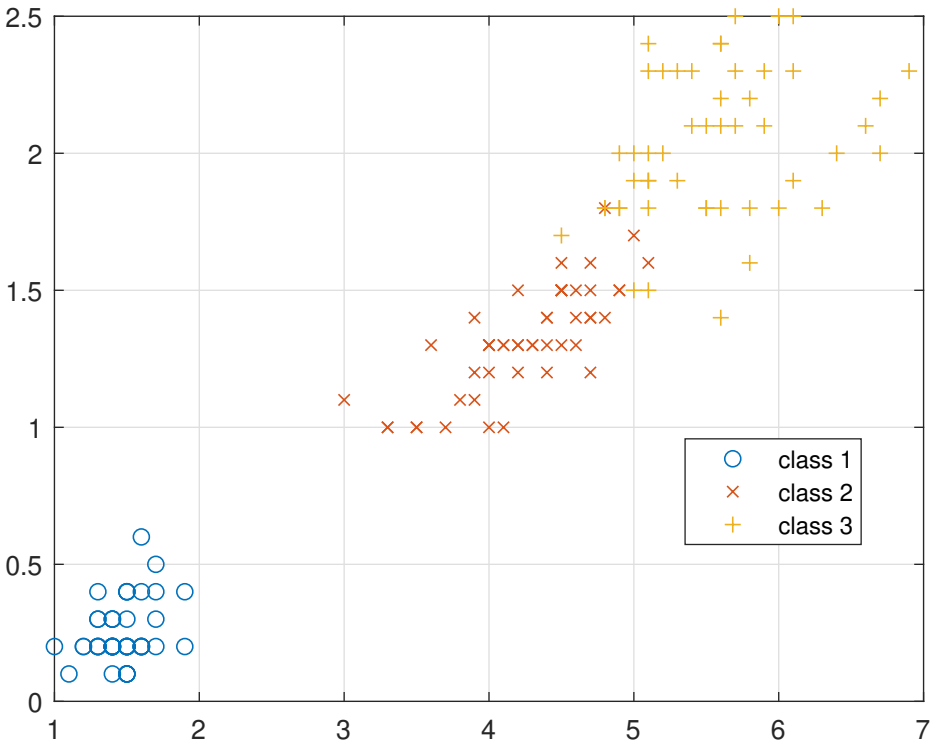

Figure 2.3: Feature 3 and 4 plotted against each other for each class.

3 Part 3 - Classification of handwritten numbers 0-9

3.1 Description

In this part of the project, the objective was to design and train a template based classifier so that it would be able to classify handwritten numbers between 0 and 9 with good precision. To do so, a given set of training data that consisted of 60000 labeled images of handwritten numbers was used, and a set of 10000 other labeled images was used for testing. All the data was obtained from the MNIST database, [1]. A general algorithm for the template based classifier is depicted in fig. 3.1. Note that the method of choosing templates is not included, as this will differ between the two following subtasks. In both cases, the Euclidian distance, eq. (1.8), will be used as distance measure.

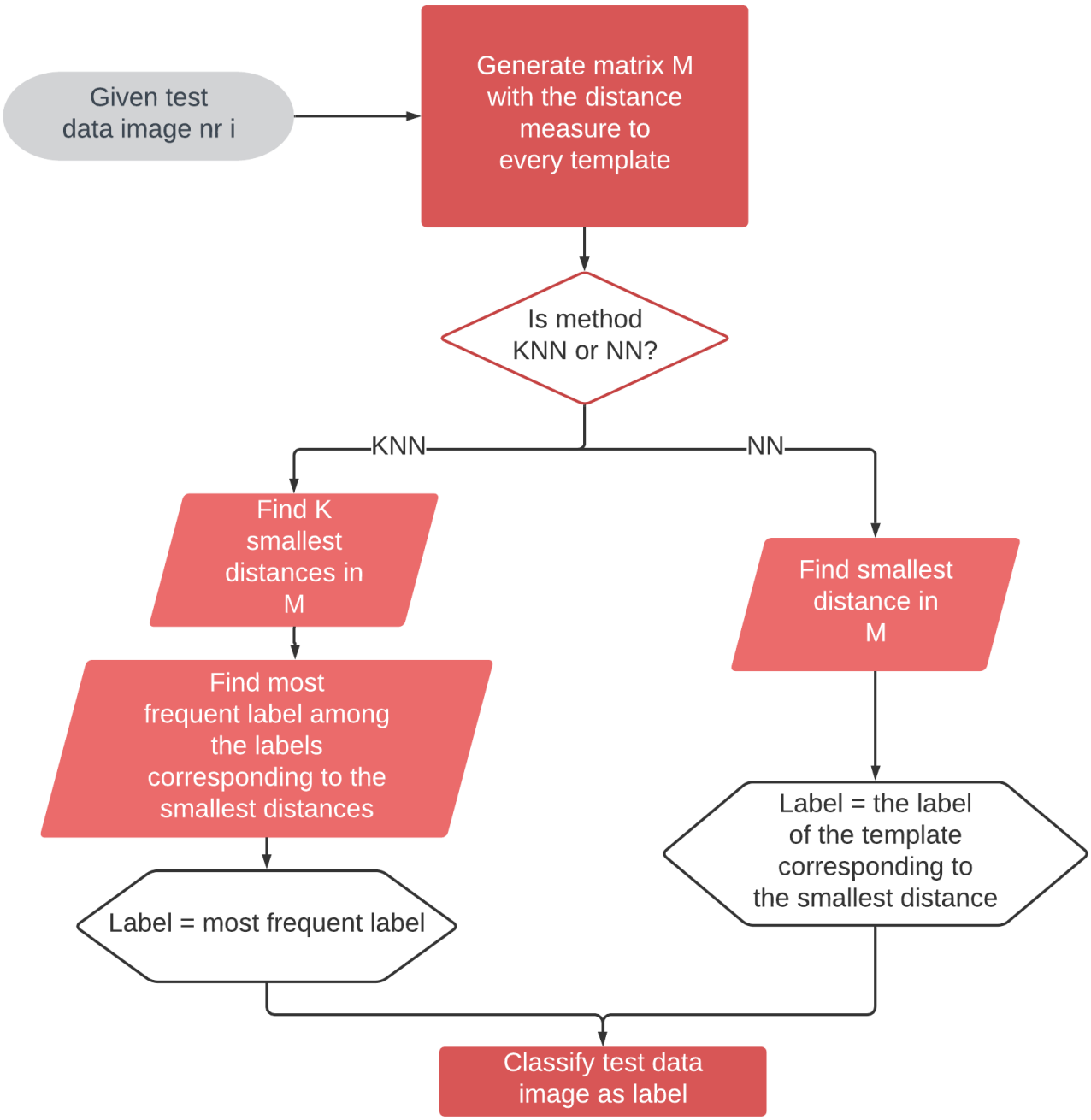

Figure 3.1: A general algorithm flowchart for the template based classifier

3.2 Using the whole training set as templates

As claimed in section section 1, the simplest way to design the templates for a template based classifier is to use all the training data as templates. In the first attempt at this problem, this approach was implemented. Based on the Eucledian distance, eq. (1.8), generated by using the MATLAB funtion `pdist2`, the nearest template to the input test image was chosen, and the corresponding label was chosen as the designated class. The code was implemented and run in MATLAB, and the associated script can be found in appendix A.5.

At first, the NN method was applied (see section 1 for more information on the different methods). This approach showed good results with an error rate of 0.0309, and the resulting

confusion matrix showed in eq. (3.1) verify that most of the numbers are classified correctly, as the numbers on the diagonal are dramatically higher than the rest.

$$C = \begin{pmatrix} 973 & 0 & 7 & 0 & 0 & 1 & 4 & 0 & 6 & 2 \\ 1 & 1129 & 6 & 1 & 7 & 1 & 2 & 14 & 1 & 5 \\ 1 & 3 & 992 & 2 & 0 & 0 & 0 & 6 & 3 & 1 \\ 0 & 0 & 5 & 970 & 0 & 12 & 0 & 2 & 14 & 6 \\ 0 & 1 & 1 & 1 & 944 & 2 & 3 & 4 & 5 & 10 \\ 1 & 1 & 0 & 19 & 0 & 860 & 5 & 0 & 13 & 5 \\ 3 & 1 & 2 & 0 & 3 & 5 & 944 & 0 & 3 & 1 \\ 1 & 0 & 16 & 7 & 5 & 1 & 0 & 992 & 4 & 11 \\ 0 & 0 & 3 & 7 & 1 & 6 & 0 & 0 & 920 & 1 \\ 0 & 0 & 0 & 3 & 22 & 4 & 0 & 10 & 5 & 967 \end{pmatrix} \tag{3.1}$$

Other observations one can make from the confusion matrix is which numbers that more often than others get mistaken for one another. The most frequent miss-classifications are mistaking 4 for 9, 7 for 2, 7 for 1, 8 for 3 and 5 for 3 and vice versa. These results resonate with our intuition, as it is known that these numbers are similar in shape. By plotting some of the instances of the most often miss-classified numbers, as done in fig. 3.2, it becomes understandable as to why the miss-classifications occur - it is likely that some of these instances would generate debate also among humans.

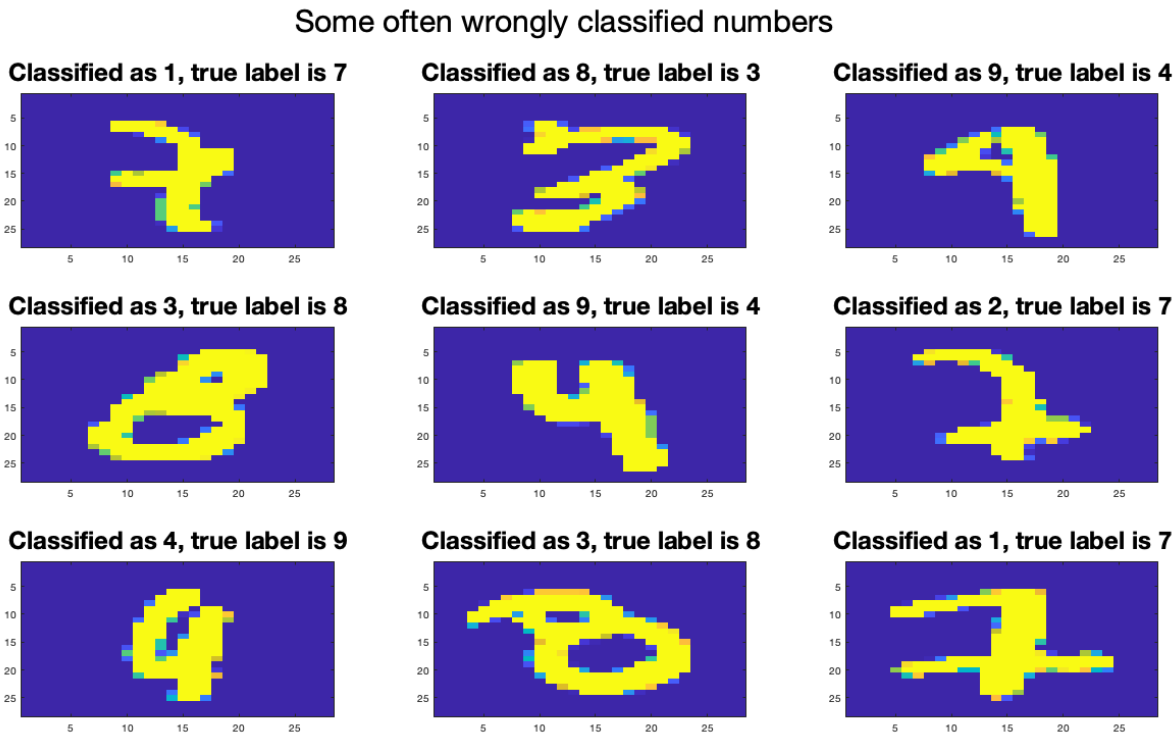

Figure 3.2: Some instances of the most occurring miss-classifications, plotted.

Some of the correctly classified numbers are plotted in fig. 3.3. This verifies the robustness of the classifier to some extent, as some of the numbers plotted are fairly different from the "textbook example" of the number. In spite of this, the classifier still manages to map it to the correct label.

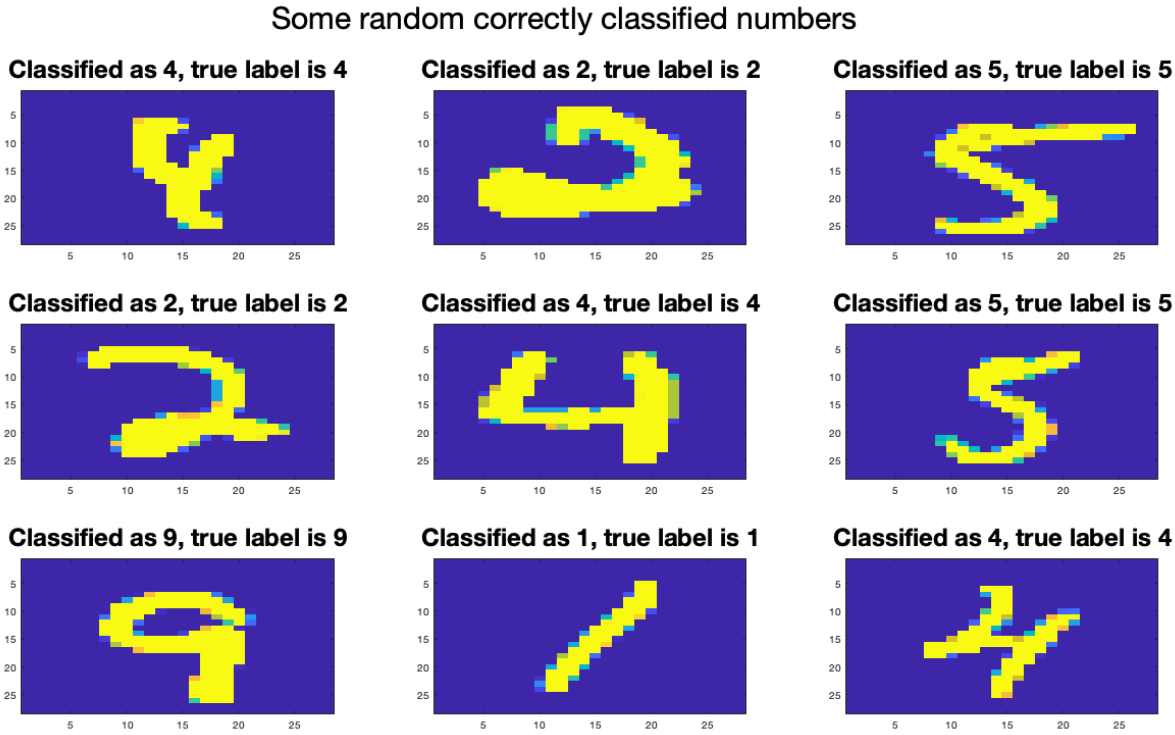

Figure 3.3: Some random correctly classified numbers, plotted.

In spite of good theoretical results, the method of comparing the test data to every single training example is generally impractical due to the amount of time it takes to run. This should come as no surprise, considering the amount of data it has to compare in order to reach a result. By making MATLAB produce a time report when running the script, it becomes clear that the majority of the time spent is used by `pdist2`, i. e. generating the matrix of Euclidian distances, see fig. 3.4.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
classify	1	365.503 s	21.283 s	
pdist2	20	344.110 s	344.099 s	

Figure 3.4: Snapshot of parts of the time report generated by MATLAB when run with 500 test images at a time. `classify` is the name of the full script. Note: most of the functions in the script are left out, due to their insignificant addition to the running time compared to pdist2.

Experimenting with the code showed that partitionning the test set into 500 images to classify at a time was the most efficient implemetation, but with this implemented the script still took a rough six minutes to reach a conclusion on the whole test set.

Implementing the KNN method with $K = 7$ instead gave an almost identical result, except that the error rate decreased to a value of 0.0306, i. e. by 0.3 thousandths. This also somewhat increased the runtime, see fig. 3.5.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
classify	1	383.980 s	22.793 s	
pdist2	20	359.507 s	359.460 s	

Figure 3.5: Snapshot of parts of the time report generated by MATLAB when run with 500 test images at a time and KNN is implemented. `classify` is the name of the full script. Note: most of the functions in the script are left out, due to their insignificant addition to the running time compared to pdist2.

3.3 Using clustering

To decrease the runtime, the method of clustering was applied before running the classification algorithm. This means that the classification-part applied is almost identical as to the previous subtask, but the templates used differ. As explained in section 1, clustering will generate templates based on the training data. In this new script, which can be found in appendix A.6, the MATLAB function `kmeans` was used to produced $M = 64$ templates for each class. Due to the nature of the `kmeans` function, the training data belonging to each individual class had to be gathered before applying it. This resulted in 64 representative vectors for each class, which was then passed on to the algorithm in fig. 3.1.

As expected, the cluster script had a significant improve in running time compared to the script in the previous subtask, see fig. 3.6.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
cluster	1	58.743 s	3.813 s	
kmeans	10	51.565 s	0.537 s	
smartForReduce	10	50.959 s	0.037 s	
kmeans>loopBody	10	50.916 s	0.111 s	
kmeans>distfun	1088	46.461 s	46.461 s	
kmeans>loopBody/batchUpdate	10	44.241 s	0.623 s	
pdist2	20	3.318 s	3.315 s	

Figure 3.6: Snapshot of parts of the time report generated by MATLAB when cluster script was run with 500 test images at a time. `cluster` is the name of the full script. Note: only the functions that contribute most to the running time are shown.

It was observed from the time report that the `kmeans` function had effectively replaced `pdist2` in the time spending hierarchy. This contributes even more to the cluster script's running time dominance when compared to the solution in the previous subtask, as the `kmeans` part of the script only has to run once in order to produce the clusters. After the clusters are generated, the rest of the script can run very effectively (in less than 10 seconds) as many times one would like, without `kmeans` harming the efficiency.

But at what cost had the running time improved? Not a very high one, it seemed. The resulting error rate was 0.0458, which is still an acceptable error rate. The confusion matrix became eq. (3.2)

$$C = \begin{pmatrix} 967 & 0 & 9 & 0 & 1 & 3 & 5 & 0 & 4 & 6 \\ 1 & 1130 & 7 & 0 & 7 & 1 & 3 & 14 & 2 & 3 \\ 3 & 1 & 979 & 6 & 3 & 1 & 0 & 11 & 4 & 5 \\ 1 & 0 & 6 & 945 & 0 & 14 & 0 & 0 & 14 & 5 \\ 0 & 0 & 4 & 0 & 923 & 3 & 5 & 5 & 3 & 32 \\ 3 & 0 & 0 & 18 & 0 & 849 & 3 & 1 & 17 & 3 \\ 3 & 2 & 4 & 0 & 6 & 7 & 940 & 0 & 2 & 0 \\ 1 & 0 & 11 & 6 & 3 & 2 & 0 & 962 & 5 & 19 \\ 0 & 2 & 12 & 21 & 1 & 8 & 2 & 1 & 916 & 5 \\ 1 & 0 & 0 & 14 & 38 & 4 & 0 & 34 & 7 & 931 \end{pmatrix} \tag{3.2}$$

which is quite similar to that of the previous approach. Comparing the two confusion matrices, it is clear that the pattern with the most frequent missclassified numbers is the same for both, only with some higher values for the cluster-script produced. This is logical, as clustering produces a generalized representation of the classes. Thus, the more far-out exmples, such as the 4-signs that look a lot like 9, may disappear when clustering is done as the generalization will tend more towards the typical representation.

When switching to the KNN method with K = 7, the time report turned out as in fig. 3.7. An increase in the running time of only one second again verifies that the classification-part is no longer the time-demanding part of the algorithm.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
cluster	1	59.396 s	5.309 s	
kmeans	10	50.251 s	0.590 s	
smartForReduce	10	49.478 s	0.050 s	
kmeans>loopBody	10	49.417 s	0.125 s	
kmeans>distfun	1047	45.092 s	45.092 s	
kmeans>loopBody/batchUpdate	10	42.660 s	0.585 s	
pdist2	20	3.331 s	3.328 s	

Figure 3.7: Snapshot of parts of the time report generated by MATLAB when cluster script was run with 500 test images at a time, using the KNN method. `cluster` is the name of the full script. Note: only the functions that contribute most to the running time are shown.

A remarkable result of implementing the KNN method, was that the error rate increased fairly dramatically to 0.0647. This is not an expected result from our perspective; our intuition said that, as in the previous sub-task, the classifier would become more exact when run with KNN. It is considered likely that it is the aforementioned effect of the generalization that causes the increase. To illustrate by example: when the cluster for the class 4 is generated, it is likely that only some of the templates generated represent the images of 4 that looks like images of 9. Thus, there may be more templates in the 9-cluster that resembles the current 4, and therefore the 9-label will dominate among the nearest neighbours. When inspecting the confusion matrix eq. (3.3),

$$C = \begin{pmatrix} 950 & 0 & 12 & 0 & 0 & 3 & 11 & 1 & 6 & 7 \\ 1 & 1129 & 18 & 4 & 16 & 5 & 4 & 38 & 2 & 10 \\ 3 & 1 & 948 & 8 & 2 & 1 & 3 & 13 & 6 & 4 \\ 2 & 1 & 11 & 957 & 1 & 34 & 0 & 1 & 22 & 11 \\ 0 & 0 & 2 & 2 & 898 & 5 & 7 & 10 & 6 & 32 \\ 10 & 1 & 2 & 19 & 0 & 825 & 8 & 0 & 32 & 5 \\ 11 & 2 & 4 & 0 & 11 & 10 & 922 & 0 & 2 & 1 \\ 2 & 0 & 10 & 6 & 2 & 1 & 0 & 927 & 5 & 23 \\ 1 & 1 & 25 & 11 & 2 & 6 & 3 & 1 & 886 & 5 \\ 0 & 0 & 0 & 3 & 50 & 2 & 0 & 37 & 7 & 911 \end{pmatrix} \tag{3.3}$$

it is clear that the number of miss-classifications in these typical cases have increased dramatically (for example from 38 to 50 in the case of 4 being classified as 9), which supports the suspicion of this effect.

4 Conclusion

When it comes to the linear classifier, we conclude that it works. The error rates were below 5% in all tests. The result depends on training data. A classifier will create different decision rules from different training data. Also, a linear classifier works better with linearly separable classes. Removing features with overlap between classes did not affect the error rate much, but did decrease training time, which is good.

When it comes to the template based classifier, the performance was also good. Error rates were found to bee between 3% and 6%. Using all the training data as templates gave the best error rates. However it took a long time to classify images. Using clustering resulted in higher error rates but dramatically faster classification once trained. Using KNN increased classification time in both cases, but the effect on the error rates varied - when using all the training data as templates, the error rate was improved slightly with KNN, while it was worsened when using clustering. The increased error rates could be caused by the classifier generalizing too much, meaning that outliers are mistaken for another class.

All in all this project have given good insight into some of the basic principles of classification.

A Matlab Code

A.1 Iris Task - Linear classifier

```
%% The Iris Task 1
clear all

%% Setup
x1 = load('Iris_TTT4275/class_1','-ascii');
x2 = load('Iris_TTT4275/class_2','-ascii');
x3 = load('Iris_TTT4275/class_3','-ascii');

Ntest   = 30;           % Number of samples for testing
i       = [3,4];        % Features to include
maxiter = 1000;         % max iterations in gradient descent method
alpha   = 0.01;         % Fixed step length in gradient descent method

% Use first 30 samples to train:
xtrain = [x1(1:Ntest,i); x2(1:Ntest,i); x3(1:Ntest,i)];
xtest  = [x1(Ntest+1:end,i); x2(Ntest+1:end,i); x3(Ntest+1:end,i)];

% Use last 30 samples to train:
% xtrain = [x1(end-Ntest+1:end,i);x2(end-Ntest+1:end,i);x3(end-Ntest+1:end,i)]
% xtest  = [x1(1:end-Ntest,i); x2(1:end-Ntest,i); x3(1:end-Ntest,i)];

C            = 3;                                          % Number of classes
D            = size(xtrain,2);                             % Dimension of x
W0           = 0.01*ones(C,D+1);                           % Initial weights
sigmoid      = @(x) 1./(1+exp(-x));
discriminant = @(xk,W) sigmoid(W*xk);
X            = [xtrain(1:Ntest*C,:)'; ones(1, Ntest*C)]; % Feature matrix
T            = [kron(ones(1,Ntest), [1 0 0]') ...         % Target values
                kron(ones(1,Ntest), [0 1 0]')  ...
                kron(ones(1,Ntest), [0 0 1]')];

%% Training
W = gradient_descent(@(W)MSE_grad(X,T,W,discriminant), W0, alpha, maxiter);

%% Testing
Xtest       = [xtest'; ones(1, size(xtest,1))];              % Test cases
Ttest       = [repelem(1,20), repelem(2,20), repelem(3,20)]; % Target
[~,classes] = max(W*Xtest);                                  % Prediction

error_rate  = sum(classes~=Ttest)/length(classes)
confusion   = confusionmat(Ttest,classes)

% Finding error rate and confusion matrix for training data
Ttrain              = [repelem(1,30), repelem(2,30), repelem(3,30)];
[~,trained_classes] = max(W*X);
error_rate_train    = sum(trained_classes~=Ttrain)/length(trained_classes)
confusion_train     = confusionmat(Ttrain, trained_classes)
```

A.2 Iris Task - MSE Gradient

```
function grad = MSE_grad(X,T,W, discriminant)
    grad = zeros(size(W));
    for k = 1:size(X,2)
        xk = X(:,k);
        tk = T(:,k);
        gk = discriminant(X(:,k),W);
        grad = grad + ((gk-tk) .* gk .* (1-gk))*xk.';
    end
end
```

A.3 Iris Task - Gradient Descent

```
function [W,n] = gradient_descent(gradient, W0, alpha, maxiter)
    W = W0;
    iterate = true;
    n = 1;
    while iterate && n < maxiter
       grad = gradient(W);
       W = W - alpha * grad;
       if mod(n,1000) == 0
           display(n)
       end
       n = n+1;
       iterate = norm(grad) > 1e-4;
    end
end
```

A.4 Iris Task - Histogram and feature plots

```
%% The Iris Task 2 - Histogram
clear all

x1 = load('Iris_TTT4275/class_1','-ascii');
x2 = load('Iris_TTT4275/class_2','-ascii');
x3 = load('Iris_TTT4275/class_3','-ascii');

close all
figure
for i = 1:4
    subplot(4,1,i)
    hold on
    histogram(x1(:,i))
    histogram(x2(:,i))
    histogram(x3(:,i))
    hold off
    grid
    title("Feature " + i)
    legend("class 1", "class 2", "class 3")
end

%% Plotting feature 1 and 2 against each other
figure
plot(x1(:,1),x1(:,2), 'o')
hold on
plot(x2(:,1),x2(:,2), 'x')
plot(x3(:,1),x3(:,2), '+')
grid
legend('class 1', 'class 2', 'class 3')

%% Plotting feature 3 and 4 against each other
figure
plot(x1(:,3),x1(:,4), 'o')
hold on
plot(x2(:,3),x2(:,4), 'x')
plot(x3(:,3),x3(:,4), '+')
grid
legend('class 1', 'class 2', 'class 3')
```

A.5 Handwritten numbers - Classification using all training data as templates

```
%% Classification of the 10K test images using all 60K
% training images as templates.
% Remember to load 'data_all.mat' before running.

classified = NaN(num_test, 1);

% Choose between NN and KNN
knn = false;
K = 7;

% Number of partitions of the test data
num_partitions = 20;

% This for loop partitions the classification into a job of 500 images
% at once to avoid too large matrices. Also speeds up the process.
for i = 1:num_partitions
    start = 1 + (i-1)*num_test/num_partitions;
    endin = i*num_test/num_partitions;
    x = testv(start:endin, :);
    % M becomes a matrix of all the 500 images' distance to the 60K
    % training images
    M = pdist2(x, trainv);
    k = 1;
    % Find the template(s) with the smallest distance and classify
    % the test image as the same as this
    if knn
       for j = start:endin
            [~, I] = mink(M(k, :), K);
            for n = 1:length(I)
                I(n) = trainlab(I(n));
            end
            right_class = mode(I);
            classified(j) = right_class;
            k = k + 1;
        end
    else
        for j = start:endin
            [~, I] = min(M(k, :));
            classified(j) = trainlab(I);
            k = k + 1;
        end
    end
end

% Calculate confusion matrix and the error rate
% for the classifier on the test set
C = confusionmat(classified, testlab);

error_rate = (num_test-trace(C))/num_test;
```

A.6 Handwritten numbers - Classification using clustering

```
%% Classification of the 10K test images using generated
% clusters as templates.
% Remember to load 'data_all.mat' before running.

% Number of templates in each cluster
M = 64;

% Choose between NN and KNN
knn = false;
K = 7;

% Number of partitions of the test data
num_partitions = 20;

%% Find indeces of the vectors of the individual classes
class_length = ones(1, 10);
max_num = 6743;
class_indeces = NaN(10, max_num);
for i = 1:num_train
    cl = trainlab(i) + 1;
    class_indeces(cl, class_length(1, cl)) = i;
    class_length(1, cl) = class_length(1, cl) + 1;
end

%% Find class specific clusters - Cis is a 3D Matrix of
% cluster matrices, each w/ 64 templates
Cis = NaN(M, 10, vec_size);
for j = 1:10
    n = class_length(j) - 1;
    classVectors = NaN(n, vec_size);
    k = 1;
    while(~isnan(class_indeces(j, k)))
        ind = class_indeces(j, k);
        classVectors(k, :) = trainv(ind, :);
        k = k + 1;
    end
    [~, Cis(:, j, :)] = kmeans(classVectors, M);
end
C = reshape(Cis, [10*M vec_size]);
%% Find smallest distance
classified = NaN(num_test, 1);
for i = 1:num_partitions
    start = 1 + (i-1)*num_test/num_partitions;
    endin = i*num_test/num_partitions;
    x = testv(start:endin, :);
    M_dist = pdist2(x, C);
    k = 1;
    if knn
        for j = start:endin
```

```
            [~, I] = mink(M_dist(k, :), K);
            for l = 1:length(I)
                I(l) = floor((I(l)-1)/M);
            end
            right_class = mode(I);
            classified(j) = right_class;
            k = k + 1;
        end
    else
        for j = start:endin
            [~, I] = min(M_dist(k, :));
            I = floor((I-1)/M);
            classified(j) = I;
            k = k + 1;
        end
    end

end

% Calculate confusion matrix and the error rate
% for the classifier on the test set
Conf = confusionmat(classified, testlab);

error_rate = (num_test-trace(Conf))/num_test;
```

References

[1] Cortes Corinna Burges Christopher J.C. LeCun Yann. *THE MNIST DATABASE of handwritten digits*. `http://yann.lecun.com/exdb/mnist`. [Accessed: 2020-03-24]. 2020.

[2] Norwegian University of Science and Department of Electronic Systems Technology. *Classification Compendium from the course TTT4275*. 2020.

[3] Norwegian University of Science and Department of Electronic Systems Technology. *TTT4275 - Estimering, deteksjon og klassifisering*. `https://www.ntnu.no/studier/emner/TTT4275#tab=omEmnet`. [Online; Accessed: 2020-04-28]. 2020.