Classification project

Group 63
Jacob B L Belsvik
Sif Högnadóttir

April 28, 2020

Department of Engineering Cybernetics

Abstract

This paper is created for the course TTT4275 - Estimation, detection and classification and focuses on the classification part. A linear classifier and several versions of the template-based classifier K Nearest Neighbour (KNN) were designed and tested using real world examples. The examples used were a database consisting of feature information on three Iris flowers and a database with handwritten numbers. The classification of the Iris flowers is close to a linearly separable problem and the designed linear classifier yielded error rates in the region of $0 - 6\%$. Comparing the template-based classifiers KNN with K $= 1$ and K $= 7$, we found that using K $= 7$ yielded slightly better results. The error rates were 3.09% and 3.06%, respectively. Both demanded high processing power and using 64 clusters as templates instead of the whole reference set reduced the processing power greatly. However, the error rate increased to 4.76%.

Contents

1 Introduction

This paper is created for the course TTT4375 - Estimation, detection and classification, and will focus on the classification aspect. In this paper we will present different methods for classification. The goal is to test and analyze the performance of a linear classifier and template-based classifiers. For this we will use databases consisting of information about three Iris flowers and handwritten numbers. These are real world practical examples, and designing classifiers that works well for these examples, can be used on other practical problems as well.

This paper is organized as follows: section 2 contains the necessary theory for the classification problems. Section 3 and section 4 contain a description of the Iris data set, as well as an implementation of a linear classifier with corresponding results. Section 5 contains a description of the MNIST handwritten numbers data set, and section 6 contains an implementation of a template based classifier with corresponding results. The closing remarks are in Section 7. Appendix A and appendix B contain confusion matrices discussed in section 4 and section 6, respectively.

2 Theory

In this section some central classifiers are presented and explained, on the basis of the material presented in the compendium [3]. These classifiers are used in the problems described later on.

2.1 The linear classifier

To use a linear classifier to solve classification problems accurately, the problem needs to be linearly separable. Most of all practical problems are non-separable, and using a linear classifier on these problems would yield a worse performance than with a nonlinear classifier.

Assume we have $n_{classes}$ number of classes and all samples have $n_{features}$ number of features, we define a discriminant function

$$g(x) = Wx + w_0, \tag{1}$$

where g and w_0 are vectors of size $n_{classes}$ and W is a matrix of size $n_{classes} \times n_{features}$. The vector x is a sample of class ω_i and w_0 is an offset constant. For simplicity we define

$$[W \quad w_0] \longrightarrow W \tag{2a}$$
$$[x^T \quad 1]^T \longrightarrow x \tag{2b}$$

such that the discriminant function can be written as

$$g(x) = Wx. \tag{3}$$

A decision rule for classifying x is then given by

$$x \in \omega_j \iff g_j(x) = \max_i g_i(x). \tag{4}$$

This means that if the value in row j has a higher value than the other rows, x is classified as class ω_j.

2.2 Training a linear classifier using Mean Square Error (MSE)

Assuming we have a data set of size n_{data} where each sample x_k belongs to a class ω_i. We define a vector t_k that describes the true class of x_k. This is done by making t_k consist of only zeros except for a single 1 in the index i that relates to the class ω_i. The matrix T that consists of t_ks then describes which class all samples in the data set belongs to.

To use the discriminant function described in section 2.1 for classification, training needs to be done such that W is found. There are several ways of doing this, but the most popular variant is the "Minimum Square Error (MSE)" approach. This approach consists of minimizing the function

$$MSE = \frac{1}{2} \sum_{k=1}^{n_{data}} (g_k - t_k)^T (g_k - t_k) \tag{5}$$

Here the total training set for all classes is used at the same time, making the total matrix W trained as one single entity.

Since the output vector $g_k = Wx_k$ is to be matched toward a vector with binary value, it needs to be scaled to only give out values between 0 and 1. Ideally a Heaviside function should be used, but the smooth Sigmoid function is an acceptable approximation. We define

$$z_k = Wx_k \tag{6}$$

and a new g_k is defined by

$$g_{ik} = sigmoid(x_{ik}) = \frac{1}{1 + e^{-z_{ik}}} \quad i = 1, \ldots, n_{classes} \tag{7}$$

Equation (5) has no explicit solution, so a gradient based technique has to be used to calculate W. This is done by calculating the gradient to the current MSE function with respect to W and update W in the opposite direction of the gradient. This means we need to calculate

$$W(m) = W(m-1) - \alpha \nabla_W MSE \tag{8}$$

until W converges. Here m is the iteration number and α is a step factor that needs to be tuned. It is important to tune the step factor with care. A too large value for α will give large fluctuations in the MSE, and a too small value would increase the number of iterations needed to converge significantly.

Thus, we need to calculate the gradient of MSE:

$$\nabla_W MSE = \sum_{k=1}^{n_{data}} \nabla_{g_k} MSE \nabla_{z_k} g_k \nabla_W z_k, \tag{9}$$

where

$$\nabla_{g_k} MSE = g_k - t_k, \tag{10a}$$

$$\nabla_{z_k} g_k = g_k \circ (1 - g_k), \tag{10b}$$

$$\nabla_{z_k} g_k = x_k^T. \tag{10c}$$

Inserting (10) into (9) we get

$$\nabla_W MSE = \sum_{k=1}^{n_{data}} [(g_k - t_k) \circ g_k \circ (1 - g_k)] x_k^T. \tag{11}$$

This expression is then used to find W at each step until it converges.

2.3 K Nearest Neighbours (KNN) classifier

The K Nearest Neighbours (KNN) classifier is a template based classifier. This classifier matches the sample x towards a set of templates (references) and finds the K closest templates. It then classifies the sample x as the class the majority of the K nearest neighbours belongs to.

Assume we have $k = 1, \ldots, n_{templates}$ templates μ_{ik} for each class ω_i and $i = 1, \ldots, n_{classes}$ number of classes. There are several ways of finding the closest neighbours, but one version is calculating the Euclidian distance,

$$d(x, \mu_{ik}) = (x - \mu_{ik})^T (x - \mu_{ik}), \tag{12}$$

for all templates in every class.

There are several variants of this classifiers. Using a different distance variant, such as the Mahalanobis distance is one way. There are also several ways the templates can be chosen. One way is using similar samples as x. One could also perform clustering and find a smaller set of templates used for the classification. Finally one can change the constant K. the simplest method is to set K equal to 1 and find the closest neighbour, or make K have a larger value. There are ways that will not be described here to find an optimal K.

2.4 Clustering

Using all references as templates require a large amount of processing power, and thus it might be desirable to use a smaller set of templates. One way is to randomly choose a desired amount of templates from the reference set. However, how the templates are chosen have a larger impact on the outcome than desired since the templates chosen might not be representative for the whole class.

There exists other methods where one gets a desired number of templates that are represents the whole class sufficiently. One method is clustering. This method creates templates that represents a given number of clusters of the samples in the reference set. This means one has to create clusters for each class individually. A method to do this is using the Matlab function $[idx_i, C_i] = \textbf{kmeans}(referencev_i, M)$. This function will cluster reference vectors (from the training set) from class ω_i into M templates given by the matrix C_i. More details on how this function works can be found in its documentation.

3 Classification of Iris flowers

In this section the task of classifying Iris flowers based on the length and width of its petal and sepal is discussed. The "Fisher Iris database" is used to do this.

3.1 Description of the database

There exists different variants of the Iris flower. Three of these are called Setosa, Versicolor and Virginica. All variants have both large (Sepal) and small (Petal) leaves, and it is possible to discriminate the variants based on the lengths and widths of said leaves. The length and width of the Sepal and Petal will from now on be referred to as *features*, and the variants of the Iris flower will be referred to as classes. The "Fisher Iris database" is a data set that consists of 50 examples for each of the three classes. More information regarding this database can be found in its Wikipedia-article[1].

For simplicity, the different classes and features will be referred to as shown in table 3.1 and 3.2

Table 3.1: The different classes

Class	Flower
Class 1	Iris-setosa
Class 2	Iris-versicolor
Class 3	Iris-virginica

Table 3.2: The different features of the flower

Feature	Flower
Feature 1	Sepal length in cm
Feature 2	Sepal width in cm
Feature 3	Petal length in cm
Feature 4	Petal width in cm

The histograms in fig. 1 shows the distribution of the different features for the classes. We see that features 3 and 4 are good for discriminating class 1, meaning it is linearly separable from the two other classes. It is also apparent that class 2 and 3 lack complete separability. So when analyzing the whole data set, it appears that the 3 classes are almost, but not completely, linearly separable.

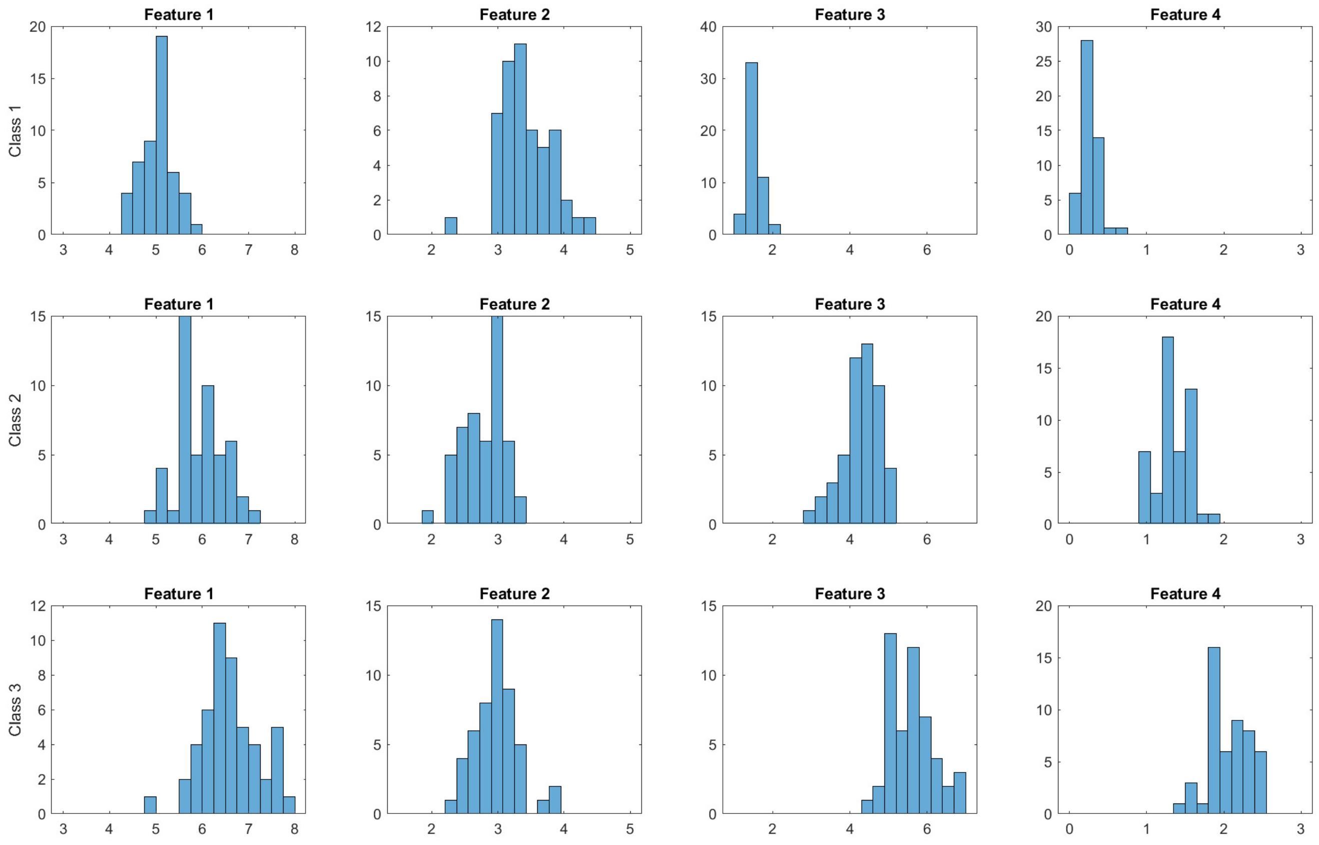

Figure 1: Histogram for each feature and class

3.2 Motivation and description of task

There are few classification problems that are linearly separable but classifying an Iris flower is close to linearly separable. Thus a linear classifier can be designed for the database.

To test how good the linear classifier is, a linear classifier has to be designed and trained. This is done by dividing the data from the database into a training- and test-set. A linear classifier as described in section 2.1 can then be trained using the training set. The test set is then used to find a confusion matrix and error rate that is used to evaluate the linear classifier.

One can also remove certain features from the problem to analyze the importance of the features with respect to linear separability.

4 Implementation and results of Iris task

In this section the implementation of the linear classifier is described. The associated results are also discussed.

4.1 Implementation

The data set is a matrix with $n_{features}$ rows and n_{data} columns. Here $n_{features}$ is the number of features available and n_{data} the number of samples in the data set. First the data set was divided into the two subsets training set and test set. The training set was used to train the linear classifier, while the test set was used to test it.

Using the MSE approach to train a linear classifier the following algorithm was implemented to train the classifier:

Algorithm 1 Training a linear classifier

```
Input: $dataset, T, W_0, \alpha$
Output: $W$

1: function LINEAR_CLASSIFIER _TRAINING($dataset, T, W_0, \alpha$)
2:     $W \leftarrow W_0$
3:     $n_{data} \leftarrow \text{length}(dataset)$
4:     while not termination criteria do
5:         for $k \leftarrow 1$ to $n_{data}$ do
6:             $z_k \leftarrow W \cdot x_k$
7:             $g_k \leftarrow \text{sigmoid}(z_k)$
8:         end for
9:         $\nabla_W MSE \leftarrow \sum_{k=1}^{n_{data}} [(g_k - t_k) \circ g_k \circ (1 - g_k)] x_k^T$
10:        $W(m) \leftarrow W(m-1) - \alpha \nabla_W MSE$
11:    end while
12:    return $W$
13: end function
```

The termination criteria is fulfilled if the classifier (W) converges or the maximum number of iterations is reached. The step factor α was tuned to make the training converge. This algorithm was implemented in Matlab and returns the linear classifier W. By choosing which features to use in the algorithm, it was possible to analyze the importance of using the different features.

This linear classifier is then used to classify the samples in both the training and test set as described in section 2.1, using the discriminant function eq. (3) and the decision rule eq. (4).

4.2 Results

To make the training of the linear classifier converge, the step factor α was tuned to have the value $\alpha = 2.0 \cdot 10^{-3}$. Both a larger and smaller value for α prevented the training from converging with a reasonable number of iterations, and the result would be a classifier with a high error rate. All the results are found using $\alpha = 2.0 \cdot 10^{-3}$.

When using the first 30 samples of each class for training, and the last 20 for testing, we get the confusion matrix in table A.1 for the training set. The confusion matrix for the test set is shown in table A.2. We see that the error rate for the classification of the testing set, 3.33%, is higher than the classification of the training set 2.22%. It is also noteworthy that in both cases there are samples from class 2 that are classified as class 3.

The training and testing of the linear classifier was repeated, but now the last 30 samples of each class was used for training, and the rest for testing. The resulting confusion matrices for the training set and testing set is shown in tables A.3 and A.4 respectively. For this case, the error rate for the training set was 5.56% and 0.0% for the test set. There is a significant difference in the error rate from this part and when the training and test sets where interchanged. This might indicate that how well the linear classifier works on a data set that is almost linear separable, depends on how the data set is split into a training and test set.

Further, from the histograms in fig. 1 we see that feature 2 has the most overlap between the classes. We removed this feature from the data set and the training and classification was performed with the remaining features. This was repeated until we had one feature left, by secondly removing feature 1 and finally feature 3, leaving feature 4 as the only feature used. All results are showed in appendix A.

Overall, there are no significant differences between the results. For all cases there are samples from class 2 that are classified as class 3, and vice versa. It is interesting to note that class 1 is always classified correctly. This means that class 1 is linearly separable from the other two classes. However, class 2 and class 3 are not entirely linearly separated from each other, as we also can see from the histograms in fig. 1. The fact that the results are so similar is expected since the features that are most linearly separable is used. If only the features 1 and 2 was used alone, the results would be significantly worse.

5 Classification of handwritten numbers 0-9

In this section the task of classifying handwritten numbers is discussed. To do this the database MNIST [2] is used.

5.1 Description of the database

The database MNIST is created by NIST and consists of 60000 written training examples of the numbers 0-9 written by 250 people and 10000 test examples written by 250 other persons. All pictures have been preprocessed such that they are centered and scaled similarly. All pictures are 8-bit greyscale and have dimension 28x28 pixels.

5.2 Motivation and description of the task

The classification of handwritten numbers is interesting because of various reasons. It has several practical uses, for example as a way of solving mathematical problems written on paper. The methods used to classify numbers can also be expanded to classifying letters, and then you have a tool to process and analyze big quantities of handwritten papers.

There exists several methods to solve the problem of classifying handwritten numbers. In this paper we will only look at some template-based classifiers. Thus we will not discuss state-of-the-art classifiers such as deep neural networks even though this approach may yield better results.

As discussed in section 2.2 there are several template based methods used to solve the task. Here, the NN and KNN approaches both with and without clustering are used. When using the method without clustering, one could argue that training should be done to reduce "overfitting". This has the important role of reducing the effect outliers has on the classification. However, to do this one needs a lot of computational power, and the procedure would take much more time to finish without sufficient increase in performance. In other words, the improvement of training before classifying is not big enough to outweigh the increase in processing needed. Thus we will not use training to solve this task.

First the whole training set is used as templates for the NN classifier. Then the 6000 training samples for each class is clustered into $M = 64$ clusters. These clusters where then used as templates for the NN classifier. Finally, a KNN classifier with K=7 was designed and tested.

6 Implementation and results of handwritten number task

In this section the implementation of the template based classifiers mentioned in section 5 is discussed. The associated results are thereafter compared and discussed.

6.1 Implementation

The KNN-classifier was implemented in Matlab, as described in the following algorithm

Algorithm 2 KNN classifier

```
Input: $K, template\_set, test\_set$
Output: Classification of samples in test_set

function KNN CLASSIFIER($K, template\_set, test\_set$)
    $n_{data} \leftarrow$ length($test\_set$)
    $Distances \leftarrow$ dist($template\_set, test\_set$)
    for $i \leftarrow 1$ to $n_{data}$ do
        Find the K nearest neighbours for sample $i$ in $Distances$
        Perform a majority vote within the KNN
        Classify sample $i$ based on vote
    end for
    return Classification of samples in test_set
end function
```

When using clustering we used the Matlab function $[idx_i, C_i] = \mathbf{kmeans}(referencev_i, M)$ to make the $M = 64$ clusters for each class ω_i, as described in section 2.4.

6.2 Results

First, we ran the KNN-classifier with K = 1, with the resulting confusion matrix given in table B.1. The numbers shown in fig. 2 are examples of numbers that were classified incorrectly, while the numbers in fig. 3 were classified correctly. From looking at the numbers it is understandable that the misclassified number were classified wrongly since they can be mistaken for other numbers.

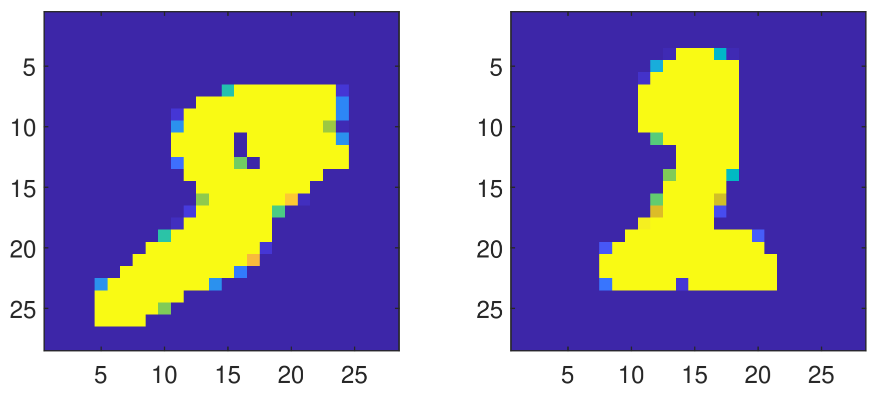

Figure 2: Numbers that were wrongly classified as 7 and 1, while their actual values are 8 and 2, respectively

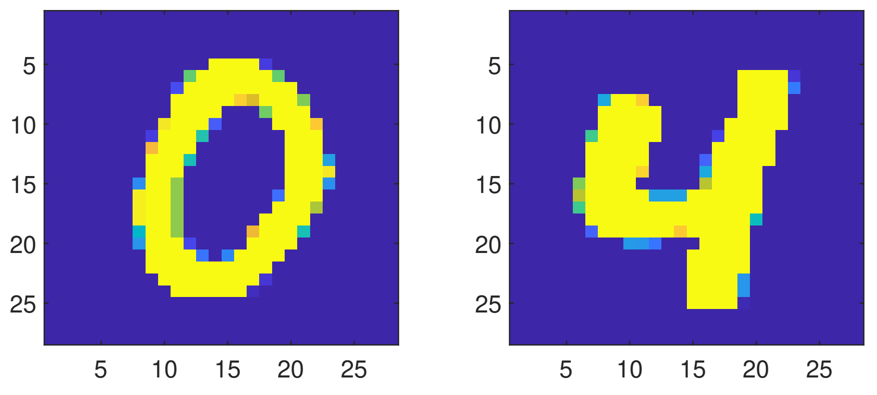

Figure 3: Numbers that were correctly classified as 0 and 4

We then ran the algorithm again, with 64 clusters per class as templates. This took significantly less time, as it now required fewer distance computations. However, the error rate was slightly higher than without clustering. Using clustering we got an error rate of 4.76% against 3.09% for the classification without clustering.

Finally, the algorithm was used with K = 7 and all the 60000 references as templates. The resulting confusion matrix is shown in table B.3. The error rate for this case was 3.06%, which is the best result of the tried methods. When using K > 1, it reduces the chance of outliers in the reference set negatively affecting the classifications and thus yields better results.

7 Conclusion

In this paper we have presented theory on several classifiers. These were then implemented in Matlab and tested with use of real-world examples. We found that a linear classifier yields good results for problems that are close to linearly separable, with error rates in the region of $0 - 6\%$. The variations in the error rates depended on the splitting of the data set and chosen features.

The template-based classifier KNN was tested and implemented for K $=$ 1 and K $=$ 7, where K $=$ 7 gave a slightly better result with an error rate of 3.06%. For this test the whole reference set was used as templates and demanded high processing power. Using 64 clusters per class to make fewer templates greatly reduced the needed processing power. However, it gave a higher error rate of 4.76%.

Throughout this project, we have learned about two useful classifiers and how they can be used on real-world examples.

A Figures and confusion matrices for Iris classification

Table A.1: Confusion matrix for training set, with the first 30 samples for training and the last 20 samples for testing (all features). Error rate: 2.22%

Classified: → True/label: ↓	Class 1	Class 2	Class 3
Class 1	30	0	0
Class 2	0	28	2
Class 3	0	0	30

Table A.2: Confusion matrix for testing set, with the first 30 samples for training and the last 20 samples for testing (all features). Error rate: 3.33%

Classified: → True/label: ↓	Class 1	Class 2	Class 3
Class 1	20	0	0
Class 2	0	18	2
Class 3	0	0	20

Table A.3: Confusion matrix for training set, with the last 30 samples for training and the first 20 samples for testing (all features). Error rate: 5.56%

Classified: → True/label: ↓	Class 1	Class 2	Class 3
Class 1	30	0	0
Class 2	0	27	3
Class 3	0	2	28

Table A.4: Confusion matrix for testing set, with the last 30 samples for training and the first 20 samples for testing (all features). Error rate: 0.0%

Classified: → True/label: ↓	Class 1	Class 2	Class 3
Class 1	20	0	0
Class 2	0	20	0
Class 3	0	0	20

Table A.5: Confusion matrix for training set, with feature 1, 3 and 4. Error rate: 3.33%

Classified: → True/label: ↓	Class 1	Class 2	Class 3
Class 1	30	0	0
Class 2	0	28	2
Class 3	0	1	29

Table A.6: Confusion matrix for testing set, with feature 1, 3 and 4. Error rate: 5.0%

Classified: → True/label: ↓	Class 1	Class 2	Class 3
Class 1	20	0	0
Class 2	0	18	2
Class 3	0	1	19

Table A.7: Confusion matrix for training set, with feature 3 and 4. Error rate: 6.67%

Classified: → True/label: ↓	Class 1	Class 2	Class 3
Class 1	30	0	0
Class 2	0	26	4
Class 3	0	2	28

Table A.8: Confusion matrix for testing set, with feature 3 and 4. Error rate: 5.0%

Classified: → True/label: ↓	Class 1	Class 2	Class 3
Class 1	20	0	0
Class 2	0	19	1
Class 3	0	2	18

Table A.9: Confusion matrix for training set, with feature 4. Error rate: 4.44%

Classified: → True/label: ↓	Class 1	Class 2	Class 3
Class 1	30	0	0
Class 2	0	27	3
Class 3	0	1	29

Table A.10: Confusion matrix for testing set, with feature 4. Error rate: 6.67%

Classified: → True/label: ↓	Class 1	Class 2	Class 3
Class 1	20	0	0
Class 2	0	18	2
Class 3	0	2	18

B Confusion matrices for handwritten number classification

Table B.1: Confusion matrix for the NN-based classifier using the whole training set as templates. Error rate: 3.09%

Classified: → True/label: ↓	0	1	2	3	4	5	6	7	8	9
0	973	1	1	0	0	1	3	1	0	0
1	0	1129	3	0	1	1	1	0	0	0
2	7	6	992	5	1	0	2	16	3	0
3	0	1	2	970	1	19	0	7	7	3
4	0	7	0	0	944	0	3	5	1	22
5	1	1	0	12	2	860	5	1	6	4
6	4	2	0	0	3	5	944	0	0	0
7	0	14	6	2	4	0	0	922	0	10
8	6	1	3	14	5	13	3	4	920	5
9	2	5	1	6	10	5	1	11	1	967

Table B.2: Confusion matrix for the NN-based classifier using 64 clusters of each class as templates. Error rate: 4.76 %

Classified: → True/label: ↓	0	1	2	3	4	5	6	7	8	9
0	964	1	4	0	0	4	5	1	0	1
1	0	1133	1	0	0	0	0	0	0	1
2	10	6	978	6	3	0	3	10	16	0
3	1	0	6	942	1	24	0	8	16	12
4	2	8	3	0	918	0	7	6	1	37
5	3	1	0	17	0	848	6	3	7	7
6	5	4	5	0	1	2	939	1	1	0
7	0	18	8	1	8	1	0	958	0	34
8	5	1	5	14	2	25	2	5	908	7
9	5	4	5	4	25	4	1	18	7	936

Table B.3: Confusion matrix for the KNN-based classifier using the whole training set as templates and K=7 Error rate: 3.06 %

Classified: → True/label: ↓	0	1	2	3	4	5	6	7	8	9
0	974	1	1	0	0	1	2	1	0	0
1	0	1133	2	0	0	0	0	0	0	0
2	11	8	988	2	1	0	2	16	4	0
3	0	3	2	976	1	12	1	7	4	4
4	1	8	0	0	945	0	5	1	1	21
5	5	0	0	8	2	866	4	1	2	4
6	6	3	0	0	3	2	944	0	0	0
7	0	25	3	0	1	0	0	989	0	10
8	6	4	6	11	7	12	1	6	916	5
9	5	6	3	6	8	4	1	11	2	963

References

[1] *Iris flower data set*. https://en.wikipedia.org/wiki/Iris_flower_data_set. Accessed: 2020-04-20.

[2] *THE MNIST DATABASE of handwritten digits*. http://yann.lecun.com/exdb/mnist/. Accessed: 2020-04-21.

[3] Magne H. Johansen Tor A. Myrvoll Stefan Werner. *Estimation, detection and classification theory.*