Report - TFE4275

Magnus Lysø
Håkon Skjetne Kvalnes
Magnar Storflor

April 2018

Summary

This report provides a basis for understanding classifiers, which is a part of a highly relevant and evolving technology: Artificial intelligence. The report is based on the results of a project done in the course Estimation, Detection and Classification (TTT4275) at The Norwegian University of Science and Technology(NTNU), the spring semester of 2018. Even though the course in its total concerns both estimation and detection as well as classification this project is based around a classification task. The project is separated into two parts to show the variety of classifiers: First a linear classifier was used to separate 3 different species of the Iris flower (Setosa, Versicolor and Virginica) based on four features; sepal width and length, and petal width and length. Secondly, a classifier based on the Gaussian Mixture Model (GMM) was used to classify a set of 11 vowels where each vowel had three features. The results of the classifiers can be said to be very good for the linear and sufficient for the GMM. The error rates for the linear classifier drops as low as to 1.67% for the test set in which the classifier is trained with 10000 iterations. The best results for the GMM comes at an error rate of 38.47% for the test set with 3 mixing proportion. This project gives a better understanding of the linear and Gaussian classifier as a whole, as they are explained and portrayed throughout the report. The classifiers worked well and the whole process of making them has given a much broader understanding of the subject and been a great way of learning.

Contents

Chapter 1

Introduction

The main objective of this project is to be able to separate different data into classes based on distinct features. In the first part the objective is to making a linear classifier that separates the three different Iris species, with a total of 150 samples (50 of each species), in to three classes. The Iris problem is such that one can get good results by using a simple linear classifier (separates the classes linearly). The second part is based around classifying 139 vowel recordings done by men, women and kids into 11 classes based on the three strongest frequency peaks corresponding to each recorded vowel. To do so a Gaussian Mixture Model is used, and this gives origin to a more complex classifier than the linear one. Different problems often requires different types of classifiers, and the complexity of the task is also an important factor when choosing which classifier type to use. To be able to classify data correctly without having to look through every single data-point manually has a huge practical value. Classification is used in a variety of ways to separate data such as in face recognition, recognizing numbers and digits, sorting spam mail and categorization of different types. This is only a small list of application for classification. The report is organized with a theory part in chapter 2, a short task description in chapter 3 and the implementation of the classifier in chapter 4. The results can be seen in chapter 5 and the conclusion in chapter 6. At the end references and code for the classifiers can be found.

Chapter 2

Theory

Classifiers are an important tool when faced with problems of identifying where a set of new observation belongs in an already established grouping of sets (classes)[1]. By using a set of data which contains observations whose class is already known, a classifier can be trained. One can later use the trained classifier on a test set. A typical example is placing an email into "spam" or "non-spam". Classification is a form of machine learning where the performance of the classifier is improved by the amount and quality of the training data that is used. How a classifier is trained varies from how the specific algorithm is constructed and what kind of classifier that is used. Classifiers may be statistically based, which means it categorizes samples by probabilities such as a Gaussian classifier or it may rely on other factors such as MSE (Minimum Square Error) which is a typical case for a linear classifier.

2.1 Linear classifier

The simplest problems in classification is often those who are linearly separable. This means that the distinct classes that are to be separated can be so by simply putting a line between them. Linear classifiers will also give good results for problems that are not a 100% linearly separable. The general form of a linear classifier is given by:

$$g = Wx + w_0 \tag{2.1}$$

where g and w_0 are vectors of the class dimension C and the matrix W is of dimension $C \times D + 1$, were D is the numbers of distinct features that are to separate the classes. The w_0 is the offset class and equation (2.1) can be simplified to $g = Wx$. The matrix W is what will separate the samples into the correct classes. The linear classifier is not statistically based, but the MSE can be used to train and optimize the classifier. The MSE can be found from:

$$MSE = \frac{1}{2}\sum_{k=1}^{N}(g_k - t_k)^T(g_k - t_k) \tag{2.2}$$

Here t is the label matrix for the correct class (a reference matrix). In order to match the output vector g and the input x with binary values, ideally a heavyside function should be used. To use the MSE in the training process, it is required that the function has a derivative. A sigmoid function gives a good approximation for this, and is given by:

$$g_{ik} = sigmoid(x_{ik}) = \frac{1}{1 + e^{(-z_{ik})}}, i = 1, ..., C \tag{2.3}$$

Here z is equal to Wx. To be able to solve equation (2.2) one have to use the chain rule to obtain the derivatives of the MSE[2]:

$$\nabla_W MSE = \sum_{k=1}^{N} \nabla_{g_k} MSE \nabla_{z_k} g_k \nabla_W z_k \tag{2.4}$$

Equation (2.4) can also be written as:

$$\nabla_W MSE = \sum_{k=1}^{N} [(g_k - t_k) \circ g_k \circ (1 - g_k)] x_k^T \tag{2.5}$$

When the gradient of the MSE is calculated it can be used to update the W matrix to suite the classification better. By performing the process of updating W repeatedly the error will converge and the classifier will "learn" and become better. The process of updating W is given by:

$$W(m) = W(m-1) - \alpha \nabla_W MSE \tag{2.6}$$

The α in equation (2.6) is a tuning factor that can be decided by trial and error to make sure the MSE converges properly. A too large α will make the MSE vary a lot for each iteration and a too small α will make the classification training unnecessary slow due to small variations in MSE from iteration to iteration.

2.2 Gaussian Classifier

A form of the Plug-in Maximum A Posteriori classifier is the Gaussian Mixture Model (GMM). This classifier type is based on the Gaussian density form. The parameters for the probability function is estimated by a set of training data. The real density of the training set may not fit the Gaussian that well, but by applying a weighted mixture of Gaussians the classifier tends to give sufficient results for a lot of practical problems. The GMM is given by:

$$p(x/w_i) = \sum_{k=1}^{M_i} c_{ik} N(\mu_{ik}, \Sigma_{ik}) = \sum_{k=1}^{M_i} \frac{c_{ik}}{\sqrt{2\pi}^D |\Sigma_{ik}|} \exp{(-\frac{1}{2}(x - \mu_{ik})^T \Sigma_{ik}^{-1} (x - \mu_{ik}))}, i = 1, ..., C \tag{2.7}$$

The c_{ik} is the weighted part and has M components that sum up to 1. What number of weighted mixtures that gives the best result may variate a lot, and a high number is not always the best. M_i can be decided from trial and error.

The sample mean and covariance for the Gaussians can be found by:

$$\hat{\mu}_i = \frac{1}{N_i} \sum_{k=1}^{N_i} x_{ik} \tag{2.8}$$

$$\hat{\Sigma}_i = \frac{1}{N_i} \sum_{k=1}^{N_i} (x_{ik} - \hat{\mu}_i)(x_{ik} - \hat{\mu}_i)^T \tag{2.9}$$

Chapter 3

Task description

The two main tasks are to create a linear classifier that separates the 3 species from the Fisher Iris data set and to separate 11 vowels by using a Gaussian Model. The theory needed to understand this is provided in chapter 2.

First, a linear classifier has to be trained by using the first 30 samples of each species as a training set and then tested against the last 20 samples of each species. The step factor is then to be decided until the training converge with a minimum of training iterations. By the trained classifier the confusion matrices and error rated for both training and test has to be found. A new linear classifier is then going to be trained by using the 30 last samples of each species as a training set and the 20 first as a test set. From the Iris data, histograms will be produced for each feature and class and used decide which features has the most overlap. The features with the most overlaps will be removed and the remaining features will be tested

The second part will be realized by using Gaussian class models. The first 70 samples are going to be used for training and the last 69 for testing. First, the mean and covariance for each class has to be calculated. Then, for different amounts of mixing proportions and covariance type, the confusion matrices and error rates for the different cases will be provided and compared.

Chapter 4

Implementation

For the implementation of the classifiers the computing program matlab was used. The codes can be seen in Appendix A, B, C and D.

4.1 Linear classifier

In appendix A, the code for the linear classifier is listed. After importing the complete Iris data (150 samples, 50 of each species) the code separates it into a training set of 90 (30 of each species) and a test set of 60 (20 of each species). The matrix W is the training matrix and is initially filled with random numbers. Z_{ik} is introduced as the multiplication of the training matrix W and the training set. To calculate the sigmoids g_{ik}, Z_{ik} is inserted in equation (2.3). A target matrix t_k is then implemented as an optimal outcome of g_{ik}, and the gradient of MSE is then calculated with equation (2.4). A new W is found by using equation (2.6). This process is repeated in a desirable amount of iterations. The data is classified by which sigmoid has the highest value for each sample in the given set. To calculate the optimal α value for this particular case a separate matlab code in Appendix B was used. This code tests amount of correct classified data for α values between 0.001 and 1, and for training iterations between 10 and 100.

4.2 Gaussian Classifier

The Gaussian classifier was also implemented by separating the vowel data into a training and a test set. Here the 70 first samples are used for training and the 69 last are used for testing, from a total of 139 samples. From the training set the mean and covariance matrices are found for each feature in every class. This can be seen in Appendix C. By using the means, covariance matrices and the training data as inputs for the matlab function *mvnpdf* it outputs the normal distributed probability density functions based on these values. The data from the training and test set is then classified by how well they fit into the different probability densities.

The code was also implemented as seen in Appendix D. Here a matlab function called *gmdistribution.fit* is used. This function creates an object containing parameters such as the mean, the covariance matrices and the mixing proportions for every class. These proportions are vectors with the length of the number of proportions and always sums to 1. By multiplying these proportions with the *mvnpdf* used in the first part of the task one can implement the weighted proportions into the classifier. This results in a Gaussian Model where it is possible to variate the mixture.

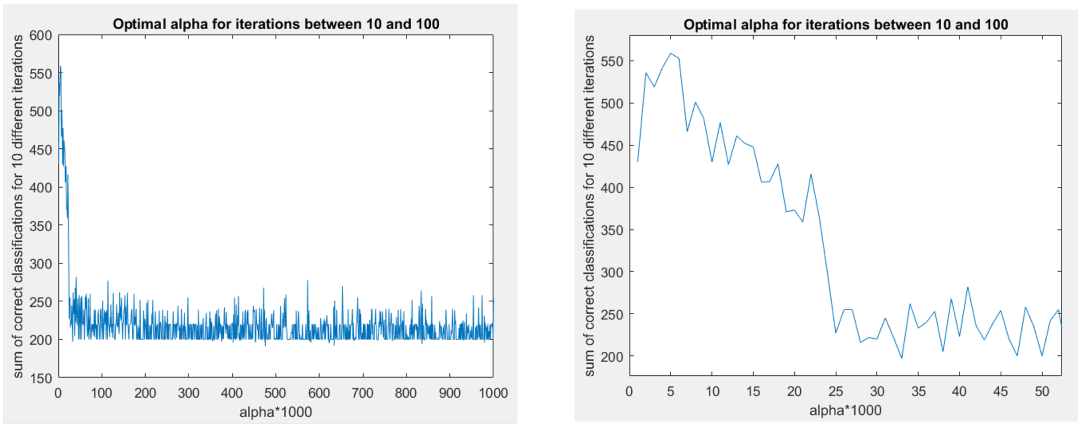
Chapter 5

Results

In this chapter all the results for both the Iris classification and the Vowels classification are presented and discussed.

5.1 Linear Classifier

First, the data was separated into a training and a test set consisting of respectively the 30 first and the 20 last samples of each class. To train the classifier the step factor alpha had to be decided. The optimal value of alpha for a small number of training iterations (10 to 100), was found by using the matlab code in appendix B. The results are shown in figure 5.1 (a) and (b). In figure 5.1 (a), alpha is plotted from 0.001 to 1, and one can see that the majority of correct classified data are located at low values of alpha. Figure 5.1 (b), therefore gives a better view of the data, zoomed in at the lower end of the plot. The maximum of the graph is located at alpha equal to 0.005 and is the optimal value for this particular case.



(a) Original plot

(b) Zoomed in plot

Figure 5.1: Plot for locating the optimal alpha value for training iterations between 10 and 100.

An alpha value of 0.005 was used in the matlab code in appendix A with 100 iterations of training. A plot of the convergence of MSE shown in figure 5.2 is outputted by running the matlab code. By studying the plot, one can notice that there is a convergence at about 40 iterations. Because a low number of iterations will make a more effective code, 40 iterations is a great compromise between efficiency and performance. In table 5.1 the confusion matrices for 40, 100 and 1000 iterations are shown, both for training and testing. The error rate for the matrices are also in the table. As one can see, there are little difference between 40 iterations of training and 100 iterations - only 1.66% bigger error rate in testing. When comparing 100 and 1000 iterations there are no differences at all in performance.

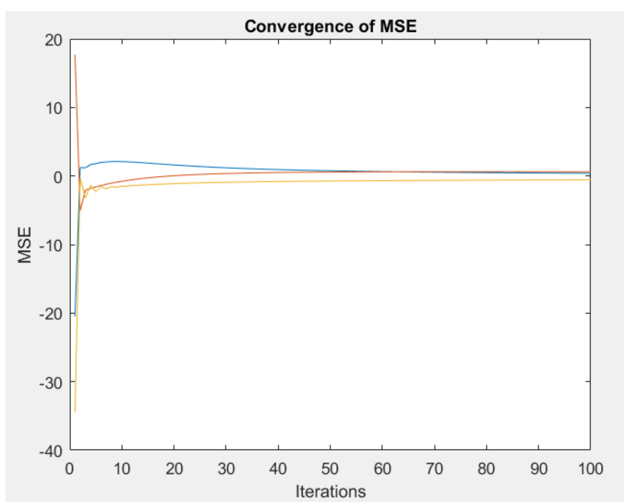


Figure 5.2: Plot of the convergence of MSE for the three classes with 100 iterations.

	40 iterations			100 iterations			1000 iterations		
Confusion matrix training	30	0	0	30	0	0	30	0	0
	0	29	1	0	30	0	0	30	0
	0	2	28	0	3	27	0	3	27
Error rate training	3.33%			3.33%			3.33%		
Confusion matrix testing	20	0	0	20	0	0	20	0	0
	0	19	1	0	20	0	0	20	0
	0	2	18	0	2	18	0	2	18
Error rate testing	5.00%			3.33%			3.33%		

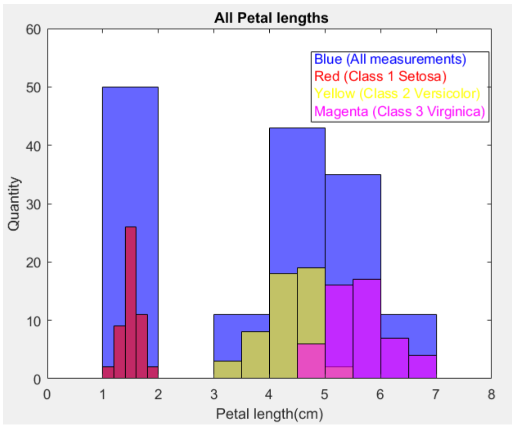
Table 5.1: Confusion matrices and error rates for different training iterations and alpha equal to 0,005. The training set consists of the first 30 samples and the test set consists of the last 20 samples.

Next, the training set was set to be the last 30 samples and the test set to be the first 20. The optimal alpha was again tested with the code in appendix B and was kept to 0.005. The new training set and test set generated the confusion matrices and error rates shown in table 5.2. When comparing matrix 5.1 and 5.2 one may see that the error rate in training is larger for the latter, but the error rate in testing is lower. This may indicate that there are more difficult data to classify in the last 30 samples than the first 30, and that therefore the last 30 are less linearly separable.

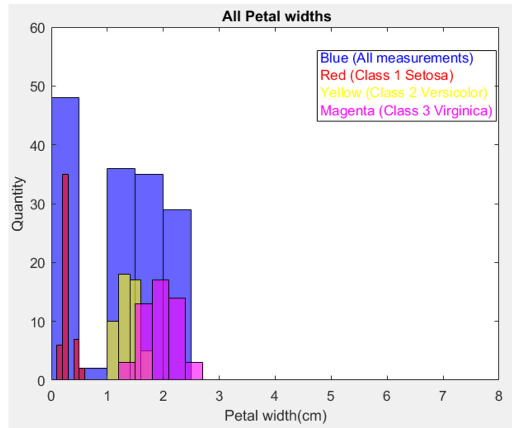
	40 iterations			100 iterations			1000 iterations		
Confusion matrix training	30	0	0	30	0	0	30	0	0
	0	29	1	0	29	1	0	30	0
	0	4	26	0	5	25	0	5	25
Error rate training	5.56%			6.67%			5.56%		
Confusion matrix testing	20	0	0	20	0	0	20	0	0
	0	19	1	0	20	0	0	20	0
	0	1	19	0	1	19	0	1	19
Error rate testing	3.33%			1.67%			1.67%		

Table 5.2: Confusion matrices and error rates for different training iterations and alpha equal to 0,005. The training set consists of the last 30 samples and the test set consists of the first 20 samples.

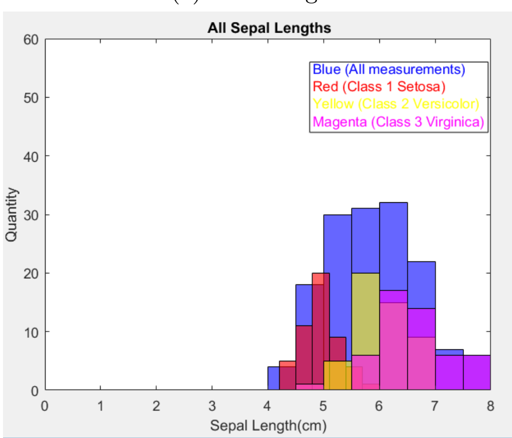
Another output from the matlab code in appendix A is the four histogram plots in figure 5.3. These plots show how the size of the different features are distributed. This information is used to exclude one or more of the most overlapping features so that the code run more efficient, and ideally, without affecting the performance. The most overlapping feature is (d), the Sepal widths - and it is therefore the worst feature. The second worst feature is (c), the Sepal lengths. Both Petal length and Petal width are good features when looking at their linear separability, but it looks like the Petal length (a) is the best feature. Table 5.3 shows how the error rates for training and testing gets affected by decreasing the number of features. What is interesting with this data is that the lowest error rate is when there is only one feature, the Petal length (a). When using only one feature, it is necessary to use about 10000 training iterations. This makes the code run twice as slow as with 40 iterations and 4 features.



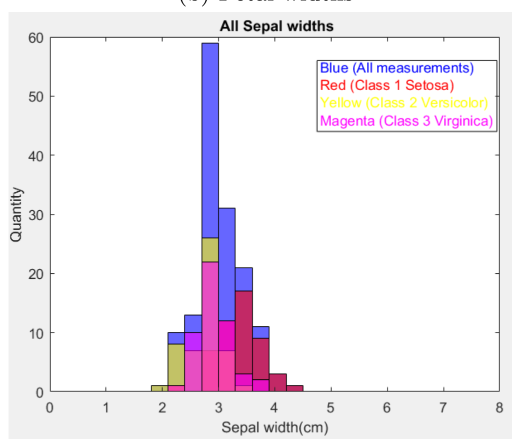
(a) Petal lengths



(b) Petal widths



(c) Sepal lengths



(d) Sepal widths

Figure 5.3: Histogram plots of all the features.

	Iterations	4 features	3 features	2 features	1 feature
Training	40	3.33%	28,89%	33.33%	33.33%
	100	3.33%	6.67%	33.33%	33.33%
	1000	3.33%	5.56%	13.33%	21.11%
	10000	3.33%	3.33%	5.56%	6.67%
Testing	40	5.00%	30.00%	33.33%	33.33%
	100	3.33%	8.83%	33.33%	33.33%
	1000	3.33%	6.67%	10.00%	15.00%
	10000	3.33%	5.00%	5.00%	1.67%

Table 5.3: Error rates for different iterations and number of features

5.2 Gaussian Classifier

When designing a Gaussian classifier one have to decide the number of mixing proportion and whether the covariance matrix is going to be diagonal or not. Here a single mixing proportion with standard and diagonal covariance matrix was tested with the matlab code in appendix C, and the results are shown in respectively table 5.4 and 5.5. Comparing these two figures, it can be seen that there are more wrongly classified data while using the diagonal covariance matrix than the full matrix since the error rates are higher. This is as expected since the reason for using a diagonal covariance matrix is to increase the efficiency, and this may affect the performance.

Confusion matrix for training set	53	0	0	14	2	0	1	0	0	0	0
	0	53	7	0	0	0	0	0	0	10	0
	0	6	59	0	0	0	0	1	0	4	0
	15	1	0	54	0	0	0	0	0	0	0
	9	0	0	3	23	0	35	0	0	0	0
	2	0	0	0	0	68	0	0	0	0	0
	1	0	0	0	3	0	66	0	0	0	0
	0	0	3	0	0	0	0	55	5	4	3
	0	0	0	1	0	1	0	2	57	2	7
	0	5	6	0	0	0	0	2	1	56	0
	0	0	0	0	1	1	0	7	1	0	51
Error rate	22.73%										
Confusion matrix for test set	48	0	0	9	12	0	0	0	0	0	0
	0	58	1	2	0	0	0	0	0	8	0
	0	24	33	0	0	1	0	1	0	10	0
	20	0	0	45	4	0	0	0	0	0	0
	13	0	0	0	39	0	17	0	0	0	0
	26	0	0	2	4	37	0	0	0	0	0
	0	0	0	0	35	0	34	0	0	0	0
	0	1	3	0	0	0	0	46	5	10	4
	0	0	0	3	5	15	0	5	32	7	2
	0	4	2	3	0	0	0	0	3	56	1
	0	0	0	1	10	10	1	16	17	0	14
Error rate	41.77%										

Table 5.4: Confusion matrix and error rates for single mixture Gaussian with full covariance matrix. The green cells indicate the correct classified data.

	36	1	0	29	3	0	1	0	0	0	0
	0	46	7	1	0	0	0	0	0	16	0
	0	10	55	0	0	0	0	3	0	2	0
	12	1	0	56	0	1	0	0	0	0	0
Confusion matrix for training set	6	0	0	4	20	0	40	0	0	0	0
	0	0	0	1	1	68	0	0	0	0	0
	1	0	0	0	11	0	58	0	0	0	0
	0	0	5	0	0	0	0	45	5	6	9
	0	0	0	2	0	0	0	4	52	3	9
	0	13	13	2	0	0	0	4	0	38	0
	0	0	0	0	0	1	1	4	13	0	51
Error rate	31.82%										
	47	0	0	7	15	0	0	0	0	0	0
	0	59	0	8	0	0	0	0	0	2	0
	0	37	22	0	0	1	0	0	0	9	0
	38	0	0	29	2	0	0	0	0	0	0
Confusion matrix for test set	11	0	0	0	52	0	6	0	0	0	0
	6	1	0	20	5	36	0	0	0	1	0
	0	0	0	0	57	0	12	0	0	0	0
	0	1	10	0	0	0	0	23	2	29	4
	0	0	0	20	3	8	0	3	20	15	0
	0	22	1	21	0	0	0	1	0	24	0
	0	0	0	5	4	9	5	14	24	3	6
Error rate	56.52%										

Table 5.5: Confusion matrix and error rates for single mixture Gaussian with diagonal covariance matrix. The green cells indicate the correct classified data.

The testing of the Gaussian classifier with two and three mixing proportions is shown in respectively table 5.6 and 5.7. There is a small improvement in the error rate for the three mixing proportions compared to two mixing proportions.

In table 5.8 the performance and efficiency of all the Gaussian classifier cases are tested. The single Gaussian (one mixing proportion) with diagonal covariance matrix has the best runtime, but the worst error rate. On the other hand, the Gaussian Mixture Model with three mixing proportions and diagonal covariance matrix has the lowest error rate, but the worst runtime. It is also worth mentioning that a single Gaussian with full covariance matrix has the second best runtime and the best training error rate. Thus the latter may be the best option for compromising between performance and efficiency.

Confusion matrix for training set	59	1	0	5	5	0	0	0	0	0	0
	0	54	7	0	0	0	0	0	0	9	0
	0	6	60	0	0	0	0	0	0	4	0
	31	1	0	38	0	0	0	0	0	0	0
	8	0	0	1	21	1	39	0	0	0	0
	2	0	0	0	0	68	0	0	0	0	0
	1	0	0	0	6	0	63	0	0	0	0
	0	0	3	0	0	0	0	53	3	4	7
	0	0	0	1	0	0	0	7	39	2	21
	0	8	9	0	0	0	0	1	1	51	0
	0	0	0	0	0	0	0	6	5	0	59
Error rate	26.62%										

Confusion matrix for test set	37	0	0	9	22	0	1	0	0	0	0
	0	60	2	2	0	0	0	0	0	5	0
	0	24	38	0	0	1	0	2	0	4	0
	25	0	0	41	3	0	0	0	0	0	0
	5	0	0	0	56	0	8	0	0	0	0
	23	2	0	4	1	39	0	0	0	0	0
	0	0	0	0	39	0	30	0	0	0	0
	0	2	1	0	0	0	0	36	8	10	12
	3	0	0	5	3	0	0	3	38	5	12
	0	12	2	7	0	0	0	0	2	45	1
	2	0	0	3	4	1	4	7	20	0	28
Error rate	40.97%										

Table 5.6: Confusion matrix and error rates for double mixture Gaussian with diagonal covariance matrix. The green cells indicate the correct classified data.

Confusion matrix for training set	51	1	0	13	5	0	0	0	0	0	0
	0	48	10	0	0	0	0	0	0	12	0
	0	3	64	0	0	0	0	1	0	2	0
	15	1	0	53	0	0	0	0	1	0	0
	8	0	0	1	31	1	29	0	0	0	0
	2	0	0	0	0	68	0	0	0	0	0
	0	0	0	0	8	0	62	0	0	0	0
	0	0	2	0	0	0	0	55	4	3	6
	0	0	0	0	0	0	0	6	51	1	12
	0	2	10	0	0	0	0	3	2	52	0
	0	0	0	0	0	0	0	7	6	0	57
Error rate	23.12%										

Confusion matrix for test set	42	0	0	9	18	0	0	0	0	0	0
	0	54	5	2	0	0	0	0	0	8	0
	0	19	40	0	0	1	0	2	0	7	0
	29	0	0	37	3	0	0	0	0	0	0
	7	0	0	0	56	0	6	0	0	0	0
	26	2	0	6	0	35	0	0	0	0	0
	0	0	0	0	36	0	33	0	0	0	0
	0	0	2	0	0	0	0	42	8	7	10
	0	0	0	4	0	0	0	6	42	5	12
	0	3	2	4	0	0	0	0	5	54	1
	1	0	0	2	0	1	4	8	21	0	32
Error rate	38.47%										

Table 5.7: Confusion matrix and error rates for triple mixture Gaussian with diagonal covariance matrix. The green cells indicate the correct classified data.

Mixing proportions	Diagonal covariance matrix	Error rate training	Error rate testing	Code runtime
1	No	22.73%	41.77%	0.053s
1	Yes	31.82%	56.52%	0.044s
2	Yes	26.62%	40.97%	0.271s
3	Yes	23.12%	38.47%	0.330s

Table 5.8: Performance and efficiency of different Gaussian classifiers.

Chapter 6

Conclusion

During this project a lot of new information and a better in depth understanding of classifiers has been achieved. A practical task such as this is a great way to test how theoretical knowledge can be used to solve real life problems.

The linear classifier for the Iris variant performed very well and had very small error rates for a relatively small amount of training iterations. The classes Versicolor and Virginica were possible to separate almost perfectly, while the Setosa class was a 100% separable from the other two. By looking at the error rates for different training iterations while reducing the number of features, one can see that the error rate increases as there are fewer features and less iterations. By using many iterations (10000), the error rate stays close to constant for the different number of features used.

For classifying different vowels, a Gaussian model was used. The data was classified using full and diagonal covariance matrix, as well as one, two and three mixing proportions. By testing the different models, it was found that the error rate goes down by increasing the number of mixing proportions and by using a full covariance matrix rather than a diagonal one. The error rates for the Gaussian model resulted in being a lot worse than for the linear classifier, but might be caused by the data sets and overlap between the vowels.

The project have been extremely giving, in the form of learning the basics of a highly relevant technology; artificial intelligence. The process with programming and understanding the different tasks was at times very difficult, but the student assistants were very helpful both during and after class.

Bibliography

[1] Wikipedia: The free encyclopedia, "Statistical classification" https://en.wikipedia.org/wiki/Statistical_classification

[2] "Classification", TTT4275 handout, ch. 3.2 (MSE based training of a linear classifier), NTNU, 2018.

[3] "Classification", TTT4275 handout, ch. 3.1 (Plug-in-Map using Maximum Likelihood (ML) based training), NTNU, 2018.

Appendix A

Code for linear classifier

```
%% Set up

x1all = load('class_1','-ascii');
x2all = load('class_2','-ascii');
x3all = load('class_3','-ascii');

%x1= [x1all(:,4) x1all(:,1) x1all(:,2)];
%x2= [x2all(:,4) x2all(:,1) x2all(:,2)];
%x3= [x3all(:,4) x3all(:,1) x3all(:,2)];

%x1= [x1all(:,3) x1all(:,4)];
%x2= [x2all(:,3) x2all(:,4)];
%x3= [x3all(:,3) x3all(:,4)];

%x1= [x1all(:,4)];
%x2= [x2all(:,4)];
%x3= [x3all(:,4)];

x1= x1all;
x2= x2all;
x3= x3all;

%% Training
x1training = [];
x2training = [];
x3training = [];

x1training = [x1training; x1(1:30,1) x1(1:30,2) x1(1:30,3) x1(1:30,4)];
x2training = [x2training; x2(1:30,1) x2(1:30,2) x2(1:30,3) x2(1:30,4)];
x3training = [x3training; x3(1:30,1) x3(1:30,2) x2(1:30,3) x3(1:30,4)];
```

```
All_training_data = [x1training; x2training; x3training];
All_training_data = [All_training_data ones(90,1)];

W = randn(3,5)/10; % Training matrix
alpha = 0.005; % Tuning constant

iterations = 40; % Iterations of training
MSE_max = zeros(iterations,3); % Maxvalues of MSE,

for k = 1:iterations

    z = W*All_training_data';

    % Calculate g (sigmoid)
    g = zeros(3,90);
    for i = 1:3
        g(i,:) = 1./(1+exp(-z(i,:)));
    end

    % Creating the target matrix for the training set
    t = zeros(3,90);
    for i = 0:2
        t(i+1,30*i+1:30*(i+1)) = 1;
    end

    % MSE for training set
    MSE = ((g-t).*g.*(1-g))*All_training_data;
    MSE_max(k,:) = sum(MSE');

    % Calculate new W based on MSE and alpha:
    W = W-alpha*MSE;
end

% Finding the confusion matrix for the training set
a = ones(1,90);
for i = 31:60
    a(i) = 2;
end
for i = 61:90
    a(i) = 3;
end

b = zeros(1,90);
for i = 1:90
   [maxvalue,index] = max(g(:,i));
```

```
    b(i) = index;
end
C = confusionmat(a,b);

% Calculating the error rate
errors = 0;
for i = 1:3
    for j = 1:3
        if (C(i,j) ~= 0) && (i~=j)
            errors = errors + C(i,j);
        end
    end
end
error_rate = errors/90;

disp('Confusion matrix for training set:');
disp(C);
disp('Error rate for training set:');
disp(error_rate);
figure(1);
plot(MSE_max);
title('Convergence of MSE')
xlabel('Iterations');
ylabel('MSE');

%% Testing

x1testing = [];
x2testing = [];
x3testing = [];

x1testing = [x1testing; x1(31:50,1) x1(31:50,2) x1(31:50,3) x1(31:50,4)];
x2testing = [x2testing; x2(31:50,1) x2(31:50,2) x2(31:50,3) x2(31:50,4)];
x3testing = [x3testing; x3(31:50,1) x3(31:50,2) x2(31:50,3) x3(31:50,4)];

All_testing_data = [x1testing; x2testing; x3testing];
All_testing_data = [All_testing_data ones(60,1)];

z = W*All_testing_data';

% Calculate g (sigmoid)
g = zeros(3,60);
for i = 1:3
```

```
    g(i,:) = 1./(1+exp(-z(i,:)));
end

% Plotting the three sigmoids
figure(2);
plot(g(1,:));
hold on;
plot(g(2,:));
hold on;
plot(g(3,:));
hold off;
title('Classification');
xlabel('x(k)');
ylabel('g(x(k))');

% Creating the target matrix for the test set
t = zeros(3,60);
for i = 0:2
    t(i+1,20*i+1:20*(i+1)) = 1;
end

% MSE for testset:
MSE = ((g-t).*g.*(1-g))*All_testing_data;

% Finding the confusion matrix for the test set
a = ones(1,60);
for i = 21:40
    a(i) = 2;
end
for i = 41:60
    a(i) = 3;
end
b = zeros(1,60);
for i = 1:60
    [maxvalue,index] = max(g(:,i));
    b(i) = index;
end
C = confusionmat(a,b);

% Finding the error rate
errors = 0;
for i = 1:3
    for j = 1:3
        if (C(i,j) ~= 0) && (i~=j)
            errors = errors + C(i,j);
        end
```

```
    end
end
error_rate = errors/60;

disp('Confusion matrix for test set:');
disp(C);
disp('Error rate for test set:')
disp(error_rate);

%%%% HISTOGRAM %%%%
All_data = [x1;x2;x3];

figure(4)
h1 = histogram (All_data(:,1),'facecolor','blue'); hold on
axis([0 8 0 60])
xlabel('Sepal Length(cm)')
ylabel('Quantity')
title('{\bf All Sepal Lengths}')
text(4.8,55,'Blue (All measurements)', 'color', 'blue');
text(4.8,52,'Red (Class 1 Setosa)','color', 'red');
text(4.8,49,'Yellow (Class 2 Versicolor)','color', 'yellow');
text(4.8,46,'Magenta (Class 3 Virginica)','color', 'magenta');
rectangle('position',[4.75 44 3.2 12])
h_1c1 = histogram(x1all(:,1),'facecolor','red');
h_1c2 = histogram(x2all(:,1), 'facecolor','yellow');
h_1c3 = histogram(x3all(:,1), 'facecolor','magenta'); hold off

figure(5)
h2 = histogram (All_data(:,2),'facecolor','blue'); hold on
axis([0 8 0 60])
xlabel('Sepal width(cm)')
ylabel('Quantity')
title('{\bf All Sepal widths}')
h_2c1 = histogram(x1all(:,2),'facecolor','red');
h_2c2 = histogram(x2all(:,2),'facecolor','yellow');
h_2c3 = histogram(x3all(:,2),'facecolor','magenta'); hold off
text(4.8,55,'Blue (All measurements)', 'color', 'blue');
text(4.8,52,'Red (Class 1 Setosa)','color', 'red');
text(4.8,49,'Yellow (Class 2 Versicolor)','color', 'yellow');
text(4.8,46,'Magenta (Class 3 Virginica)','color', 'magenta');
rectangle('position',[4.75 44 3.2 12])


figure(6)
h3 = histogram (All_data(:,3),'facecolor','blue'); hold on
axis([0 8 0 60])
```

```
xlabel('Petal length(cm)')
ylabel('Quantity')
title('{\bf All Petal lengths}')
h_3c1 = histogram(x1all(:,3),'facecolor','red');
h_3c2 = histogram(x2all(:,3),'facecolor','yellow');
h_3c3 = histogram(x3all(:,3), 'facecolor','magenta'); hold off
text(4.8,55,'Blue (All measurements)', 'color', 'blue');
text(4.8,52,'Red (Class 1 Setosa)','color', 'red');
text(4.8,49,'Yellow (Class 2 Versicolor)','color', 'yellow');
text(4.8,46,'Magenta (Class 3 Virginica)','color', 'magenta');
rectangle('position',[4.75 44 3.2 12])


figure(7)
h4 = histogram (All_data(:,4),'facecolor','blue'); hold on
axis([0 8 0 60])
xlabel('Petal width(cm)')
ylabel('Quantity')
title('{\bf All Petal widths}')
h_4c1 = histogram(x1all(:,4),'facecolor','red');
h_4c2 = histogram(x2all(:,4),'facecolor','yellow');
h_4c3 = histogram(x3all(:,4),'facecolor','magenta'); hold off
text(4.8,55,'Blue (All measurements)', 'color', 'blue');
text(4.8,52,'Red (Class 1 Setosa)','color', 'red');
text(4.8,49,'Yellow (Class 2 Versicolor)','color', 'yellow');
text(4.8,46,'Magenta (Class 3 Virginica)','color', 'magenta');
rectangle('position',[4.75 44 3.2 12])
```

Appendix B

Code for optimizing alpha

```
%% Set up

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

%% Training
x1training = [];
x2training = [];
x3training = [];

x1training = [x1training; x1(1:30,1) x1(1:30,2) x1(1:30,3) x1(1:30,4)];
x2training = [x2training; x2(1:30,1) x2(1:30,2) x2(1:30,3) x2(1:30,4)];
x3training = [x3training; x3(1:30,1) x3(1:30,2) x2(1:30,3) x3(1:30,4)];

All_training_data = [x1training; x2training; x3training];
All_training_data = [All_training_data ones(90,1)];

correct = zeros(1000,10);
for alpha_num = 1:1:1000 % Iterating from alpha=0.001 to alpha=1
    alpha = alpha_num/1000;
for iterations = 10:10:100 % Trying for different training iteration
    lengths
MSE_vec = zeros(iterations,3);
W = randn(3,5)/10; % Training matrix
for k = 1:iterations

    z = W*All_training_data';

    % Calculate g (sigmoide)
```

```
    g = zeros(3,90);
    for i = 1:3
        g(i,:) = 1./(1+exp(-z(i,:)));
    end

    % Creating the target matrix for the training set
    t = zeros(3,90);
    for i = 0:2
        t(i+1,30*i+1:30*(i+1)) = 1;
    end

    % MSE for training set
    MSE = ((g-t).*g.*(1-g))*All_training_data;
    MSE_vec(k,:) = sum(MSE');

    % Calculate new W based on MSE:
    W = W-alpha*MSE;
end



%% Testing

x1testing = [];
x2testing = [];
x3testing = [];

x1testing = [x1testing; x1(31:50,1) x1(31:50,2) x1(31:50,3) x1(31:50,4)];
x2testing = [x2testing; x2(31:50,1) x2(31:50,2) x2(31:50,3) x2(31:50,4)];
x3testing = [x3testing; x3(31:50,1) x3(31:50,2) x2(31:50,3) x3(31:50,4)];

All_testing_data = [x1testing; x2testing; x3testing];
All_testing_data = [All_testing_data ones(60,1)];

z = W*All_testing_data';

% Calculate g (sigmoid)
g = zeros(3,60);
for i = 1:3
    g(i,:) = 1./(1+exp(-z(i,:)));
end

% Creating the target matrix for the test set
```

```
t = zeros(3,60);
for i = 0:2
    t(i+1,20*i+1:20*(i+1)) = 1;
end

% MSE for testset
MSE = ((g-t).*g.*(1-g))*All_testing_data;

% Calculating the number of correct classified data for every alpha and
% Training iteration
a = ones(1,60);
b = zeros(1,60);
for i = 21:40
    a(i) = 2;
end
for i = 41:60
    a(i) = 3;
end
for i = 1:60
    [maxvalue,index] = max(g(:,i));
    b(i) = index;
    if b(i) == a(i)
        correct(alpha_num,iterations/10) = correct(alpha_num,iterations
            /10)+1;
    end
end
end
end
% Finding the best alpha:
sum_correct = sum(correct');
disp('optimal alpha:');
[max_value, alpha_1000] = max(sum_correct);
disp(alpha_1000/1000);

% Plotting number of correct classified data for all alpha
plot(sum_correct);
title('Optimal alpha for iterations between 10 and 100');
xlabel('alpha*1000');
ylabel('sum of correct classifications for 10 different iterations');
xlim = 40;
```

Appendix C

Code for single Gaussian Mixture

```
wovels = load('Wovels_12class.mat');
training_set = zeros(70,3,11);
test_set = zeros(69,3,11);

size_test = 70;

training_set(1:70,:,1) = wovels.xae(1:70,:);
training_set(1:70,:,2) = wovels.xah(1:70,:);
training_set(1:70,:,3) = wovels.xaw(1:70,:);
training_set(1:70,:,4) = wovels.xeh(1:70,:);
training_set(1:70,:,5) = wovels.xei(1:70,:);
training_set(1:70,:,6) = wovels.xer(1:70,:);
training_set(1:70,:,7) = wovels.xih(1:70,:);
training_set(1:70,:,8) = wovels.xoa(1:70,:);
training_set(1:70,:,9) = wovels.xoo(1:70,:);
training_set(1:70,:,10) = wovels.xuh(1:70,:);
training_set(1:70,:,11) = wovels.xuw(1:70,:);

% Putting the training sets into one matrix
training = zeros(size_test*11,3);
for i = 1:11
    training((i-1)*size_test+1:i*size_test,:) = training_set(:,:,i);
end

% Calcluating sample mean
sample_mean = zeros(11,3);
for i = 1:11
```

```
sample_mean(i,:) = sum(training_set(:,:,i))/70;
end

% Calculating covariance matrix
cov_matrix = zeros(3,3,11);
for i = 1:11
    cov_matrix(:,:,i) = (training_set(:,:,i)-repmat(sample_mean(i,:)
        ,70,1))'*(training_set(:,:,i)-repmat(sample_mean(i,:),70,1))/70;
end
for i = 1:11
    for j = 1:3
        for k = 1:3
            if j ~= k
                cov_matrix(k,j,i) = 0;
            end
        end
    end
end

% Creating pdfs for every class
X = zeros(11,size_test*11);
for i = 1:11
        X(i,:) = mvnpdf(training, sample_mean(i,:), cov_matrix(:,:,i))
            ';
end

% Plotting the pdfs
figure(1);
for i = 1:11
    plot(X(i,:));
    hold on;
    title('Distribution of classes for training set')
end

% Classifying X
[~, index_train] = max(X);
indd = zeros(size_test,11);
for i=1:11
    indd(:,i) = index_train((i-1)*size_test+1:i*size_test);
end

% Creating confusion matrix
confd = zeros(11,11);
for i = 1:11
        for j = 1:11
                confd(i,j)= length(find(indd(:,i) == j));
```

```
        end
end
disp('Confusion matrix for training set:');
disp(confd);

% Finding the error rate
errors = 0;
for i = 1:11
    for j = 1:11
        if (confd(i,j) ~= 0) && (i~=j)
            errors = errors + confd(i,j);
        end
    end
end
error_rate = errors/(size_test*11);
disp('Error rate for training set:');
disp(error_rate);

%% Testing
size_test = 69;

test_set(1:69,:,1) = wovels.xae(71:139,:);
test_set(1:69,:,2) = wovels.xah(71:139,:);
test_set(1:69,:,3) = wovels.xaw(71:139,:);
test_set(1:69,:,4) = wovels.xeh(71:139,:);
test_set(1:69,:,5) = wovels.xei(71:139,:);
test_set(1:69,:,6) = wovels.xer(71:139,:);
test_set(1:69,:,7) = wovels.xih(71:139,:);
test_set(1:69,:,8) = wovels.xoa(71:139,:);
test_set(1:69,:,9) = wovels.xoo(71:139,:);
test_set(1:69,:,10) = wovels.xuh(71:139,:);
test_set(1:69,:,11) = wovels.xuw(71:139,:);

% Putting the test sets into one matrix
testing = zeros(size_test*11,3);
for i = 1:11
    testing((i-1)*size_test+1:i*size_test,:) = test_set(:,:,i);
end

% Creating pdfs
Y = zeros(11,size_test*11);
for i = 1:11
        Y(i,:) = mvnpdf(testing, sample_mean(i,:) , cov_matrix(:,:,i));
end

% Plotting pdfs
```

```
figure(2);
for i = 1:11
    plot(Y(i,:));
    hold on;
    title('Distribution of classes for test set')
end

% Classifying Y
[~, index_test] = max(Y);
indt = zeros(size_test,11);
for i=1:11
    indt(:,i) = index_test((i-1)*size_test+1:i*size_test);
end

% Finding confusion matrix
conft = zeros(11,11);
for i = 1:11
        for j = 1:11
                conft(i,j)= length(find(indt(:,i) == j));
        end
end
disp('Confusion matrix for test set:');
disp(conft);

% Finding the error rate
errors = 0;
for i = 1:11
    for j = 1:11
        if (conft(i,j) ~= 0) && (i~=j)
            errors = errors + conft(i,j);
        end
    end
end
error_rate = errors/(size_test*11);
disp('Error rate for test set:');
disp(error_rate);
```

Appendix D

Code for multi-weighted Gaussian Mixture

```
wovels = load('Wovels_12class.mat');
training_set = zeros(70,3,11);
test_set = zeros(69,3,11);

size_test = 70;

training_set(1:70,:,1) = wovels.xae(1:70,:);
training_set(1:70,:,2) = wovels.xah(1:70,:);
training_set(1:70,:,3) = wovels.xaw(1:70,:);
training_set(1:70,:,4) = wovels.xeh(1:70,:);
training_set(1:70,:,5) = wovels.xei(1:70,:);
training_set(1:70,:,6) = wovels.xer(1:70,:);
training_set(1:70,:,7) = wovels.xih(1:70,:);
training_set(1:70,:,8) = wovels.xoa(1:70,:);
training_set(1:70,:,9) = wovels.xoo(1:70,:);
training_set(1:70,:,10) = wovels.xuh(1:70,:);
training_set(1:70,:,11) = wovels.xuw(1:70,:);

% Putting the training sets into one matrix
training = zeros(size_test*11,3);
for i = 1:11
    training((i-1)*size_test+1:i*size_test,:) = training_set(:,:,i);
end

% Using GMM for finding Cov. matrix and mean
Gmm = cell(11,1);
for i = 1:11
```

```
    Gmm{i} = gmdistribution.fit(training_set(:,:,i),2,'SharedCov', true, 'CovType', 'Diagonal', 'regularization', 10^-4);
end

% Putting the covariance values to the diagonal
Cov_matrix = zeros(3, 3, 11);
for i = 1:11
    for j = 1:3
        Cov_matrix(j,j,i) = Gmm{i}.Sigma(j);
    end
end

% Creating pdfs for every class
l = length(Gmm{1}.ComponentProportion);
X = zeros(11,size_test*11);
for i = 1:11
    for j = 1:l
        X(i,:) = X(i,:) + Gmm{j}.ComponentProportion(j)*mvnpdf(training,Gmm{i}.mu(j,:),Cov_matrix(:,:,i))';
    end
end

% Plotting the pdfs
figure(1);
for i = 1:11
    plot(X(i,:));
    hold on;
    title('Distribution of classes for training set')
end

% Classifying X
[~, index_train] = max(X);
indd = zeros(size_test,11);
for i=1:11
    indd(:,i) = index_train((i-1)*size_test+1:i*size_test);
end

% Creating confusion matrix
confd = zeros(11,11);
for i = 1:11
        for j = 1:11
                confd(i,j)= length(find(indd(:,i) == j));
        end
end
disp('Confusion matrix for training set:');
disp(confd);
```

```
% Finding the error rate
errors = 0;
for i = 1:11
    for j = 1:11
        if (confd(i,j) ~= 0) && (i~=j)
            errors = errors + confd(i,j);
        end
    end
end
error_rate = errors/(size_test*11);
disp('Error rate for training set:');
disp(error_rate);

%% Testing
size_test = 69;

test_set(1:69,:,1) = wovels.xae(71:139,:);
test_set(1:69,:,2) = wovels.xah(71:139,:);
test_set(1:69,:,3) = wovels.xaw(71:139,:);
test_set(1:69,:,4) = wovels.xeh(71:139,:);
test_set(1:69,:,5) = wovels.xei(71:139,:);
test_set(1:69,:,6) = wovels.xer(71:139,:);
test_set(1:69,:,7) = wovels.xih(71:139,:);
test_set(1:69,:,8) = wovels.xoa(71:139,:);
test_set(1:69,:,9) = wovels.xoo(71:139,:);
test_set(1:69,:,10) = wovels.xuh(71:139,:);
test_set(1:69,:,11) = wovels.xuw(71:139,:);

% Putting the test sets into one matrix
testing = zeros(size_test*11,3);
for i = 1:11
    testing((i-1)*size_test+1:i*size_test,:) = test_set(:,:,i);
end

% Creating pdfs
Y = zeros(11,size_test*11);
for i = 1:11
    for j = 1:1
        Y(i,:) = Y(i,:) + Gmm{j}.ComponentProportion(j)*mvnpdf(testing,
            Gmm{i}.mu(j,:),Cov_matrix(:,:,i))';
    end
end

% Plotting pdfs
figure(2);
```

```
for i = 1:11
    plot(Y(i,:));
    hold on;
    title('Distribution of classes for test set')
end

% Classifying Y
[~, index_test] = max(Y);
indt = zeros(size_test,11);
for i=1:11
    indt(:,i) = index_test((i-1)*size_test+1:i*size_test);
end

% Finding confusion matrix
conft = zeros(11,11);
for i = 1:11
        for j = 1:11
                conft(i,j)= length(find(indt(:,i) == j));
        end
end
disp('Confusion matrix for test set:');
disp(conft);

% Finding the error rate
errors = 0;
for i = 1:11
    for j = 1:11
        if (conft(i,j) ~= 0) && (i~=j)
            errors = errors + conft(i,j);
        end
    end
end

error_rate = errors/(size_test*11);
disp('Error rate for test set:');
disp(error_rate);
```