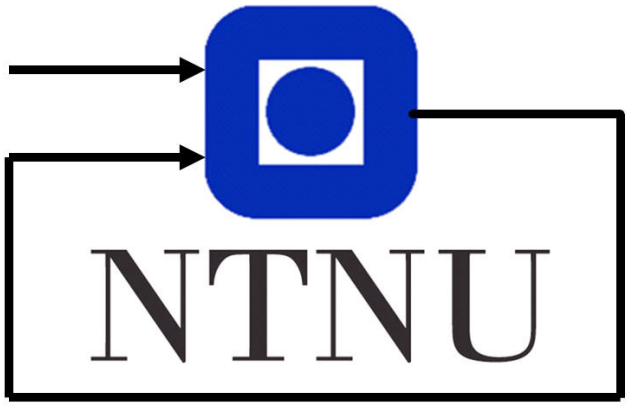
Classification of iris flowers and vowels

Eivind Heldal Stray
Magnus Dyre-Moe

April 22, 2020



Department of Engineering Cybernetics

This report consists of two tasks. The first task will consist the implementation - and testing of a Linear Classifier on Iris flowers. This is an example of a nearly linearly separable problem, and tests will be done on 4,3,2 and 1 features. Not surprisingly, evaluating more features result in better performance than evaluating few.

The second task is to classify vowels by means of a Maximum Likelihood approach, assuming different kinds of Gaussian distributions. The first part of this task will assume single Gaussian, while the second part will assume a mixed Gaussian with 2-3 components.

Contents

1 Introduction

Classification is the problem of identifying which of a set of categories new observations belong. This is often done by dividing a large variety of observations into a training set and a test set. The training set is used to determine certain parameters in a classifier to which create a selection criteria for new observations. The remaining observations, the test set, is then used to test the classifier for performance.

Classifiers are widely used in society from facial recognition to communication and economy. Hence the use for classifiers are abundant. Furthermore classifiers are an important feature of machine learning and data mining, a field with endless possibilities that are waiting to be discovered.

This report will showcase our work on classifiers applied on classifying iris flowers and pronounced vowels. The theory is found in [1] and [2] and referred to throughout this report. Note that when discussing error rates in this report, these refer to $e = 1 - \frac{correct}{total}$ for the test sets in question. Hence, when referring to performance or accuracy we refer to $p = 1 - e = \frac{correct}{total}$.

2 Theory

As humans we are incredibly capable of classifying. We can easily distinguish between an Asian female and an African female. We are even able to classify Asian females into sub classes of lets say Indian females and Chinese females. There are certain features we can see with our eyes in order to come to a conclusion. However if we were to see a Chinese and an Indian woman for the first time we would not be able to properly classify them. Hence, we have learned distinguishing traits that tell them apart over a certain amount of sightings (samples). In the same way we can train a computer to learn and classify by specific traits. When evaluating data for say, radio waves or a barometer, a classifier becomes very useful as a means of separating data.

2.1 Different classification problems and classifiers

There are many different classification problems. However it is common to distinguish classification problems into three domains; linear problems, nonlinear problems and non seperable problems.

A linear problem is a classification problem where the different classes can be separated by a linear decision border. For the 2D problem it is a line, for the 3D problem it is a plane and beyond three dimension the linear border is called a hyperplane [1]. For the linear problems one utilizes a linear classifier. The linear classifier creates the line/plane/hyperplane that is the decision border between different classes. Because the classes are separated it is possible to make an error free classifier. The iris flower data set is close to being a linearly separated classification problem.

A nonlinear classification problem is a problem were the classes are separated, but it is not possible to make an error free classifier by creating a linear decision border (hyperplane). However it is possible to make an error free classifier by making a non linear classifier. The non linear classifier creates a non linear decision border, hence the name of the classifier.

A non separated problem is classification problem where the classes overlap. Hence there is no distinct decision border that can create an error free classifier. Unfortunately most practical problems are non separable. Whether to use a linear or non linear classifier depends on the problem. The choice of classifier often comes down to complexity and performance (error rate). The vowel classification problem is an example of a non separated problem.

2.2 Training a classifier

For the iris problem a linear classifier, also known as perceptron, will be used and a non linear classifier will be used for the vowels problem. Even though there exists two classifiers, there are many different ways to train and test a classifier, resulting in vast differences in performance.

For the Iris problem, an MSE (mean-squared-error) approach will train the linear classifier. This classifier, also known as perceptron, uses a weight matrix rather than probability density functions of the classes (which is common) . For the iris task we get[2]

$$g = Wx + W_0 \tag{1}$$

Where W is the weight matrix, x is a vector containing information about the flowers and w_0 is an offset. The g is called a discriminant and the decision rule is

$$g_j(x) = \max_i g_i(x) \tag{2}$$

The crucial part of the linear classifier is to have a correct weight matrix W. W is calculated numerically by

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

[1] Classification, Johnsen, 2017, page 6

[2] Classification, page 9-10

Where m is the number of iterations used to calculate the weight matrix, α is a constant and $\nabla_W MSE$ is the gradient of the mean-squared-error with respect to the weight matrix.

The gradient of the MSE is the steepest descent. Hence, in order to reduce the error we move W in the opposite direction of the steepest descent.

Observing the equation for W, number of iterations and α are important in order to obtain a good weight matrix. Number of iterations is important for the run time of the algorithm and α is important as a scalar on our line-search. An α to large or too small may contribute to never reaching our desired weight matrix.

For the vowel problem a ML (maximum likelihood) classifier will be used. The ML classifier uses a statistical approach, to the contrary of the MSE classifier. ML estimates necessary parameters in order to create a probability density function. For $\omega_i, i = 1, ..., C$ classes a given training subset $X_{N_i} = \{x_{i1}, ..., x_{iN_i}\}$ has unknown parameters $\Lambda_i = \{\mu_i, \varepsilon_i\}$ that we wish to estimate. μ is the expected vector and ε is the covariance matrix. The likelihood is then

$$L[X_{N_i}, \Lambda_i] = \prod_{k=1}^{N_i} p(x_{ik}/\Lambda_i) \tag{4}$$

Solving the likelihood function for each class $\omega_i, i = 1, ..., C$ returns the unknown parameters which in turn creates probability density functions for every class.

The size of the training subset are important in order to obtain reliable parameters Λ_i. Larger training sets will give more accurate parameters which will turn useful when testing the classifier. However, it is also important to have a testing subset in order to actually test the classifier. For the vowel classification problem the class is split into 70 vowels for training and 69 for testing.

2.3 Confusion matrix

For a classifier it is common to use a confusion matrix as a means of analyzing performance. A confusion matrix looks like the following for a classification problem with three classes.

		Predicted		
		A	B	C
Actual	A	7	8	9
	B	4	5	6
	C	1	2	3

Where there is a difference between the predicted class which is the output of the classifier and the actual class which is the blueprint given by the data you analyze. The confusion matrix will always be a N by N matrix where N is the number of classes. For a problem with 10 classes the confusion matrix will be a 10 by 10 matrix.

The ideal classifier has a confusion matrix with only non zero values on the diagonal. Every element not on the diagonal is an error which will in turn lower the accuracy of the classifier. Because the structure of the confusion matrix it is an easy way of interpreting performance. Hence, why it is widely used as an analyzing tool.

For simplicity, a python script[3] for printing latex confusion matrices is used on the vowel task. The code is found in Appendix B.4. This was done to avoid typing them into latex manually.

[3]https :// github.com/kristeey/TTT4275_EDC/

3 Iris task

In this task, the aim is to implement a linear classifier capable of distinguishing between three flowers; Iris Setosa, Iris Versicolor and Iris Virginica by means of four key features for each flower. These features are sepal width and length and petal width and lengths. As described by the task, this is a problem that is close to linearly separable. The linear classifier performed with a success-ratio of 95% and above when analyzing all 4 features, which is close to what one would expect using the libraries found online.

3.1 Implementation of the perceptron

The linear classifier is implemented using python3.8, and the data is extracted from *sklearn's* library *load_iris*. Matrix operations are handled using *numpy*, and no other functionality is used from *sklearn* but implemented manually. There are, however, many functions in *sklearn* that implement solutions such as the linear classifier that is implemented to solve this problem.

The linear classifier takes on the form[4] $g = argmax\{\mathbf{g}\}$, where $\mathbf{g} = \mathbf{Wx} + \mathbf{W_0}$. Here, $\mathbf{x} \in \mathbb{R}^{\mathbf{Dx1}}$ is a set of features, in this case widths and lengths, thus $D = 4$. Furthermore, $\mathbf{W}$ contains the weights of each feature for a given class. In this case there are $C = 3$ classes, and the $\mathbf{W} \in \mathbb{R}^{\mathbf{CxD}}$ matrix is therefore a matrix of dimension $3x4$ The $\mathbf{W_0}$ vector holds an offset value, which we assume to be zero for the entire task. The training algorithm goes as follows:

initialize **W**
initialize one training set for each class of desired size
for iterations = 0 to n_iterations **do**
  for classes = 0 to m_classes **do**
    calculate ∇**MSE**
  end for
  W$- = \alpha\nabla$**MSE**
end for

To test one set of features, simply calculate $argmax(\mathbf{g})$ and test it against the corresponding iris-index $i \in 0, 1, 2$

How to calculate ∇**MSE** can be found in [1] on page 17.

Note that in this algorithm, $\mathbf{W_0}$ is not mentioned. This is because it is set to $\mathbf{0}$ in this implementation. The implementation in python can be found in Appendix A.1

3.2 Selected α and MSE plots

The training was initially done using 30 samples for training and 20 for testing for each class. These values were found by testing for low and high values, and then narrowing down these values until the amount of iterations were as low as possible while the α remained as high as possible. However, the training set seems to converge after 1000 iterations as seen by the mean-square-error plot as seen in Figure 1. Using $\alpha = 0.005$ over 1500 iterations however was significantly smoother as seen in Figure 2. Yet, the error rate on the training - and test set was not quite as good as whilst using 3000 iterations and $\alpha = 0.01$. Thus, throughout this task, 3000 iterations and $\alpha = 0.01$ was used, despite the fact that 1000 or 1500 iterations and one of the two aforementioned alphas would give a more time efficient algorithm.

[4] Classification, Johnsen, 2017, page 9, 10 and 15-17

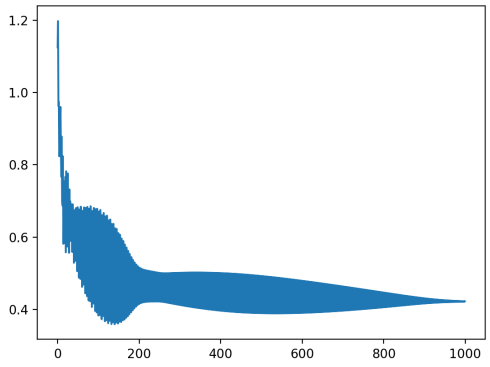


Figure 1: MSE, 4 features and $\alpha = 0.01$

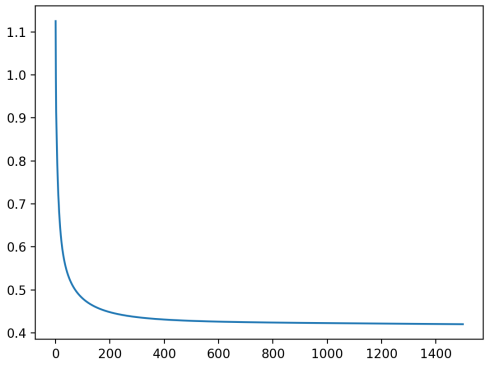


Figure 2: MSE, 4 features and $\alpha = 0.05$

3.3 Confusion matrices and results

The first classifier was implemented using the first 30 samples for training and the 20 last for testing. See the confusion matrices for training and test below:

Table 1: Training Set

class	Set.	Ver.	Vir.
Set.	30	-	-
Ver.	-	28	2
Vir.	-	1	29

Table 2: Test Set

class	Set.	Ver.	Vir.
Set.	20	-	-
Ver.	-	18	2
Vir.	-	-	20

First 30 samples for training. Last 20 for testing

The result is giving the classifier an accuracy rate of 96,67% for both the training and test set. Hence, the error rate is 3,33%.

The next test was to train the classifier on the last 30 samples and test it on the first 20. The result was as following:

Table 3: Training Set

class	Set.	Ver.	Vir.
Set.	30	-	-
Ver.	-	27	3
Vir.	-	5	25

Table 4: Test Set

class	Set.	Ver.	Vir.
Set.	20	-	-
Ver.	-	20	-
Vir.	-	-	20

Last 30 samples for training. First 20 for testing

The result is an accuracy of 91,11% on the training set and 100% on the test set. The error rate on the training set is not ideal. However the performance on the test is fantastic with no wrong classifications.

For the remainder of the Iris task the object was to observe the classifier's performance while removing features of the flowers. First we made histograms of the different features

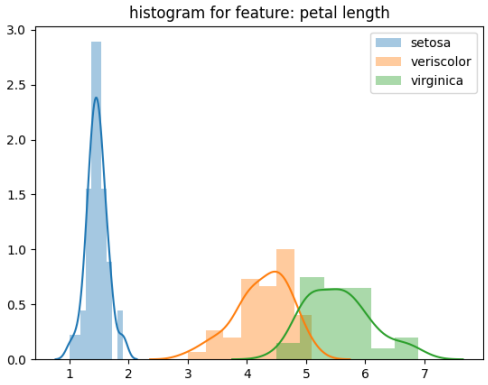


Figure 3: Petal length of iris flowers

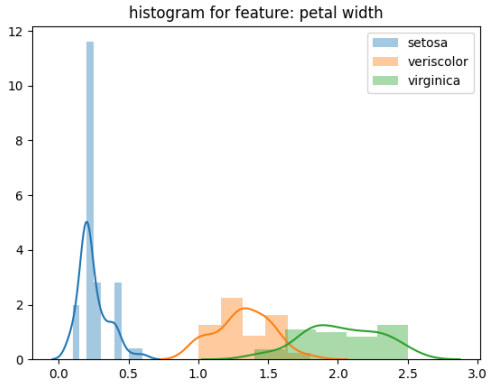


Figure 4: Petal width of iris flowers

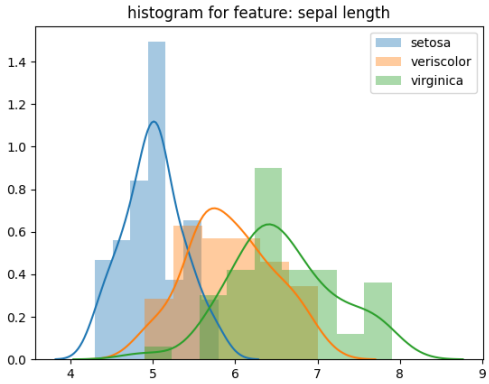


Figure 5: Sepal length of iris flowers

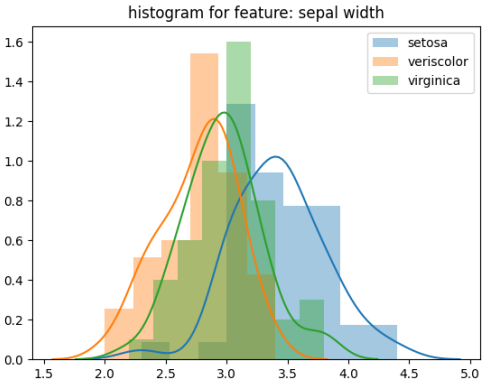


Figure 6: Sepal width of iris flowers

Then we started to remove features. The first feature to be removed was sepal width since it was the feature that showed the most overlap between the iris flowers. When training and testing the classifier for the remaining three features we obtained the following confusion matrices:

Table 5: Training Set

class	Set.	Ver.	Vir.
Set.	30	-	-
Ver.	-	28	2
Vir.	-	1	29

Table 6: Test Set

class	Set.	Ver.	Vir.
Set.	20	-	-
Ver.	-	18	2
Vir.	-	1	19

3/4 features used for training and testing.

Netting an accuracy of 96,67% for the training set and 95% for the testing set.

The next feature to be removed was sepal length. Meaning we now train and test a classifier on two distinct features. The accuracy of the classifier decreased drastically.

Table 7: Training Set

class	Set.	Ver.	Vir.
Set.	26	4	-
Ver.	3	14	13
Vir.	-	8	22

Table 8: Test Set

class	Set.	Ver.	Vir.
Set.	18	1	1
Ver.	-	16	4
Vir.	1	3	16

2/4 features used for training and testing

Giving an accuracy of 68,89% on the training set and 83,34% on the test set. The result of removing two features clearly has its effect on the classifier.

For the final test we used the classifier on petal width, meaning petal length, sepal width and length were removed as traits of the iris flowers. The result is found in the following confusion matrices

Table 9: Training Set

class	Set.	Ver.	Vir.
Set.	-	-	30
Ver.	-	-	30
Vir.	-	-	30

Table 10: Test Set

class	Set.	Ver.	Vir.
Set.	-	-	20
Ver.	-	-	20
Vir.	-	-	20

1/4 features used for training and testing

Which guesses correct on 33,33% of the flowers. 33,33% is statistically the result you would obtain while randomly guessing between 3 different flowers. Hence the classifier with only one feature is no better than guessing.

3.4 Discussion

Every iris flower has some distinct traits that makes them separable from each other. Sepal width and length and petal width and length creates a trace of data that can be linked to each flower. When we remove data about a flower we lose valuable information which makes the classification problem more difficult. When removing sepal length the linear classifier loses one element of valuable data of each flower. Because sepal length was the most overlapping trait for the iris flowers the classifier performed as well as when all four traits were in use. Never the less information about the flowers were lost.

When removing two traits from the data it is clear that the classifier's performance has decreased. When sepal width and length were removed from the data the classifier only had information about the petals of each flower. There is a huge downgrade in performance, which is no surprise as we have removed valuable data that were useful in order to create a decision border.

When the classifier only had information about petal width the conclusion was the same for every flower. Every flower that was tested was classified as Iris Virginica, giving the classifier a poor performance with a success rate of 33,33% which is no better than random guessing between the three flower types. Hence, it is obvious that removing valuable information makes the decision border less distinct. However, we suspect the classifier using different number of iterations and a different value for α may produce a better result. Judging from the histograms, petal width is the feature with the least amount of overlap and setosa has no overlap with either versicolor or virginica. Therefore, in theory the classifier should be able to at least tell setosa apart from veriscolor and virginica.

From the observation and our reasoning on the importance of data as information in use of classification, we believe that all information which can be used to isolate classes from one another is

valuable information. Our belief is if we were to have more information about other distinct traits of the iris flowers the performance of the classifier would be better.

4 Vowels Part 1

For the vowels task the goal is to classify different vowels based on a sequence of numerical values describing the frequency spectrum of each pronounced vowel. In order to classify the vowels, a Maximum Likelihood (ML) approach is used. In the first part, a Single Gaussian distribution is assumed for each class, while the latter part assumes a Mixed Gaussian distribution of 2-3 components for each class.

4.1 Implementation of the Maximum Likelihood Classifier using single gaussian

In this task, vowels represented as a series of 15 numbers for each sample will be classified by choosing the most likely vowel. By assuming a gaussian distribution for each vowel, for each and every sample-series ("ae","ih",...) consisting of 70 test-vowels each, the sample mean and covariance matrix is found. Then, by calculating $P(x = w_i)$ for each class w_i, the most likely vowel is chosen. The algorithm goes as follows:

extract 12 training sets of length N.
for each training set, calculate the sample mean
and the covariance matrix.
for vowel = first vowel to last vowel **do**
  for x = first to last in test set **do**
    calculate $P(x = vowel)$
  end for
  return the most likely vowel
end for

The implementation of this in python can be found in Appendix B.2.

4.2 Finding the covariance matrix and the mean vector

For a sample of size N, the mean vector and covariance matrix is found by[5]:

$$\hat{\mu} = \frac{1}{N}\sum_{i=0}^{N}(x_i) \tag{5}$$

and

$$\hat{\varepsilon} = \frac{1}{N}\sum_{i=0}^{N}[(x_i - \hat{\mu})(x_i - \hat{\mu})^T] \tag{6}$$

4.3 Pitfall

In this implementation, when extracting the training sets, these were extracted chronologically. In practice, this means that the mean and covariance are based exclusively by using pronounciations from men and some women (50 men and 20 women in the case of a training size of 70 samples). The test sets however consist of only women and children. For this reason, the performance when predicting the training set was far better than that of the test set. This was corrected by dividing each sample list in such a way that every class of people (man,woman,boy,girl) were as equally represented as possible. After making this change, the accuracy of the testing set went drastically up, while the accuracy of the training set went down a bit. This probably has to to with the fact that the training is done on more types of pronounciations, being men, women, boys and girls rather than mostly men. In the training set we use 20 men, 20 women, 20 boys and 10 girls while the test set contains 26 men, 27 women, 7 boys and 9 girls.

[5]Classification, Johnsen, 2017, page 15

4.4 Results and Confusion Matrices

When using the ML Single Gaussian Classifier on this vowel task assuming full covairance pdfs, 95.6% of the training set was classified correctly, while 85.9% of the test set was classified correctly. Clearly, similar sounds may in some cases cause confusion. "ae" may for example be confused with "eh". This is also the case when a person is to distinguish between two quite similar sounds. The Confusion matrices are shown below:

Table 11: Training Set

class	ae	ah	aw	eh	er	ei	ih	iy	oa	oo	uh	uw
ae	69	-	-	1	-	-	-	-	-	-	-	-
ah	-	64	4	-	-	-	-	-	-	-	2	-
aw	-	2	66	-	-	-	-	-	-	-	2	-
eh	3	-	-	66	-	-	-	-	-	-	1	-
er	-	-	-	-	62	-	4	4	-	-	-	-
ei	-	-	-	-	1	69	-	-	-	-	-	-
ih	-	-	-	-	-	-	70	-	-	-	-	-
iy	-	-	-	-	2	-	-	68	-	-	-	-
oa	-	-	-	-	-	-	-	-	67	-	-	3
oo	-	-	-	-	-	-	-	-	-	69	1	-
uh	-	-	3	-	-	-	-	-	-	1	66	-
uw	-	-	-	-	-	-	-	-	-	-	-	70

Table 12: Test Set

class	ae	ah	aw	eh	er	ei	ih	iy	oa	oo	uh	uw
ae	56	1	-	7	3	2	-	-	-	-	-	-
ah	-	50	7	-	-	1	-	-	-	-	11	-
aw	-	10	50	-	-	1	-	-	-	-	8	-
eh	8	1	-	56	-	-	-	-	-	-	4	-
er	1	-	-	-	56	-	4	8	-	-	-	-
ei	-	-	-	1	2	64	-	-	-	-	2	-
ih	1	-	-	1	-	-	67	-	-	-	-	-
iy	3	-	-	-	-	-	1	65	-	-	-	-
oa	-	-	1	-	-	-	-	-	65	-	1	2
oo	-	-	-	-	-	-	-	-	-	62	6	1
uh	-	-	4	-	-	2	-	-	-	-	63	-
uw	1	-	1	-	-	-	2	-	6	1	-	58

Single Gaussian with full covariance matrix

We also tested the Single Gaussian classifier with a diagonal covariance matrix. The result yielded 75,36% accurate for the training set and 69,93% accurate for the test. The confusion matrices are as following:

Table 13: Training Set

class	ae	ah	aw	eh	er	ei	ih	iy	oa	oo	uh	uw
ae	50	1	-	13	-	1	2	-	-	2	1	-
ah	1	48	17	-	-	-	-	-	-	-	4	-
aw	-	16	47	-	-	-	-	-	5	-	2	-
eh	7	-	-	46	-	4	-	-	-	9	4	-
er	1	-	-	-	46	1	20	2	-	-	-	-
ei	-	-	-	1	-	68	-	-	-	1	-	-
ih	-	-	-	-	6	-	63	-	-	-	-	1
iy	-	-	-	-	4	-	1	65	-	-	-	-
oa	-	-	1	-	-	-	-	-	58	2	2	7
oo	-	-	-	4	-	-	-	-	1	41	8	16
uh	-	3	-	3	-	-	-	-	2	10	52	-
uw	-	-	-	-	-	-	1	-	8	12	-	49

Table 14: Test Set

class	ae	ah	aw	eh	er	ei	ih	iy	oa	oo	uh	uw
ae	38	1	-	16	4	8	-	-	-	1	1	-
ah	-	43	10	-	-	1	-	-	-	-	15	-
aw	-	13	42	-	-	1	-	-	8	-	5	-
eh	3	1	-	48	-	1	2	-	-	12	2	-
er	-	-	-	-	32	1	29	7	-	-	-	-
ei	-	-	-	2	-	67	-	-	-	-	-	-
ih	-	-	-	-	2	-	64	1	-	-	-	2
iy	-	-	-	-	6	-	5	58	-	-	-	-
oa	-	-	-	-	-	-	-	-	55	-	4	10
oo	-	-	-	4	-	-	1	-	-	42	4	18
uh	-	3	-	4	-	-	-	-	-	20	42	-
uw	-	-	-	-	-	1	4	-	10	6	-	48

Single Gaussian with diagonal covariance matrix

4.5 Discussion

We observe a decrease in performance when removing of non diagonal elements in the covariance matrix essentially only containing variances. Like with the iris task all information is valuable information that may tell vowels apart. Hence when removing information about the covariance we lose valuable information that may be the difference when telling similar sounding vowels apart.

5 Vowels Part 2

5.1 Implementation of the Maximum Likelihood Classifier Using Mixed Gaussian

This implementation is similar to single gaussian, but for each normal distribution use a mixture of 2-3 gaussian mixtures per class. That is, it fits the model to a gaussian mixture of 2-3 different components. In this case, 4 would probably be better since we assume 4 different classes of people pronouncing the vowels (men, women, boys and girls)[6]. The library used for this operation in python is *sklearn.mixture*. The implementation in python can be found in Appendix B.3.

5.2 Results and Confusion Matrices

Firstly we trained and tested the classifier for two gaussian mixtures. The result is the following confusion matrices.

Table 15: Training Set

class	ae	ah	aw	eh	er	ei	ih	iy	oa	oo	uh	uw
ae	50	1	-	12	-	1	2	-	-	-	4	-
ah	1	49	19	-	-	-	-	-	-	-	1	-
aw	-	14	48	-	-	-	-	-	6	-	2	-
eh	6	-	-	55	-	3	1	-	-	-	3	2
er	1	-	-	1	49	-	17	2	-	-	-	-
ei	-	-	-	1	1	66	-	-	-	-	2	-
ih	-	-	-	-	5	-	63	-	-	-	-	2
iy	-	-	-	-	17	-	-	53	-	-	-	-
oa	-	-	-	-	-	-	-	-	60	2	2	6
oo	-	-	-	1	-	-	-	-	-	66	2	1
uh	-	3	2	5	-	-	-	-	3	8	49	-
uw	-	-	-	-	-	-	1	-	8	3	1	57

Table 16: Test Set

class	ae	ah	aw	eh	er	ei	ih	iy	oa	oo	uh	uw
ae	39	2	1	17	1	2	-	4	-	-	3	1
ah	1	42	13	1	-	-	-	-	1	-	12	-
aw	-	12	42	2	-	1	-	-	8	-	5	-
eh	4	1	-	54	2	2	4	-	-	-	1	2
er	1	-	-	-	37	5	21	6	-	-	-	-
ei	-	-	-	2	-	62	6	-	-	-	-	-
ih	-	-	-	-	3	-	57	8	-	-	-	2
iy	-	-	-	-	18	-	4	40	4	-	-	4
oa	-	-	-	-	-	-	-	-	51	9	4	6
oo	-	-	-	4	-	-	1	-	-	53	9	3
uh	-	3	1	3	-	-	-	-	-	9	43	11
uw	-	-	-	-	-	1	4	-	10	3	-	40

Training and testing with two gaussians

Resulting in an accuracy of 79,17% for the training set and 67,63% for the test set. We observe that the performance has decreased compared to when using a single gaussian.

Next of we trained and tested the classifier for three gaussian mixtures. The confusion matrices are the following:

Table 17: Training Set

class	ae	ah	aw	eh	er	ei	ih	iy	oa	oo	uh	uw
ae	61	1	-	3	-	-	1	-	-	-	4	-
ah	1	56	12	-	-	-	-	-	-	-	1	-
aw	-	9	59	-	-	-	-	-	-	-	2	-
eh	8	-	-	57	1	1	1	-	-	-	2	-
er	1	-	-	-	47	-	9	13	-	-	-	-
ei	-	-	-	1	-	68	-	-	-	-	1	-
ih	-	-	-	-	8	-	61	1	-	-	-	-
iy	-	-	-	-	6	-	-	64	-	-	-	-
oa	-	-	-	-	-	-	-	-	60	4	-	6
oo	-	-	-	-	-	-	-	-	-	65	3	2
uh	-	2	3	1	-	-	-	-	-	6	58	-
uw	-	-	-	1	1	-	-	-	6	3	-	59

Table 18: Test Set

class	ae	ah	aw	eh	er	ei	ih	iy	oa	oo	uh	uw
ae	51	2	-	11	5	-	-	-	-	-	1	-
ah	1	51	13	-	-	-	-	-	-	-	5	-
aw	-	11	47	2	-	1	-	-	1	-	8	-
eh	11	1	-	54	4	-	-	-	-	-	-	-
er	-	-	-	-	48	5	7	10	-	-	-	-
ei	-	-	-	2	-	61	6	-	-	1	-	-
ih	-	-	-	-	9	-	54	7	-	-	-	-
iy	-	-	-	-	12	-	-	50	5	-	-	3
oa	-	-	1	-	-	-	-	-	54	10	2	3
oo	-	-	-	2	-	-	-	-	-	52	10	6
uh	-	3	4	1	-	-	-	-	1	2	49	10
uw	-	-	-	-	2	1	2	-	7	3	-	43

Training and testing with three gaussians

[6] https://en.wikipedia.org/wiki/Mixture_model

Giving the classifier an accuracy of 85,12% on the training set and 74,15% on the test set. We observe that the performance have increased compared to when using two gaussians. However, the classifier is still not as accurate as when using a single gaussian.

5.3 Discussion

Clearly, the best classifier in this case was the ML Single Gaussian classifier assuming full covariance pdfs, (Table 12) with a performance of 85.9% on the test set. The second best was the Mixed Gaussian using 3 components (Table18) with an accuracy for the test set of 74.15%

The diagonal Single Gaussian ML classifier (Table 14) had an accuracy of 69.9%. This classifier confused more vowels with quite different vowels such as "er" and "ih", than the full covariance pdf version which confuses quite few of these cases.

The mixed gaussian case using 2 components (Table 16) performed the worst with an accuracy on the test set of 67.6%, confusing quite a few different vowels for each class.

Plotting a multivariate distribution of more than 2 variables is unfortunately not possible, and thus one may only assume from these results that the pdfs of these functions are in fact closer to a single gaussian than a mixed one. This is surprising due to the fact that the vowels are pronounced by men, women, boys and girls. Intuitively, this would make the distribution for each class closer to a Mixed Gaussian with 4 components since the frequency of a childs voice is usually more high-pitched than that of a man and a woman's voise more high-pitched than that of a man.

6 Conclusion

The Linear Classifier in Section 3 performed well on the almost linearly separable problem of distinguishing between three different Iris flowers. By eliminiting feature by feature, it was clear that the problem became less and less separable resulting in a much worse performance for the last set using only one feature than the first and second using 4 and 3 features, respectively.

The Maximum Likelihood classifier in Section 4 and Section 5 proved to give a performance of 85.9% at best for the test sets, which is not great but not outrageous. Surprisingly, the best performance for the vowel classification in these tasks was obtained assuming a Single Gaussian distribution, despite the vowels being pronounced by men, women, boys and girls, perhaps leading one to believe that this would be a Mixed Gaussian distribution of 3-4 components.

A Iris Python code

A.1 iris.py

```
import numpy as np
import random
import matplotlib.pyplot as plt
import seaborn as sns



def allocate_data(data, target):
  training_data1 = data[0:30]
  training_data2 = data[50:80]
  training_data3 = data[100:130]
  training_data = [training_data1,training_data2,training_data3]
  training_target1 = target[0:30]
  training_target2 = target[50:80]
  training_target3 = target[100:130]
  training_target = [training_target1, training_target2, training_target3]

  testing_data1 = data[30:50]
  testing_data2 = data[80:100]
  testing_data3 = data[130:150]
  testing_data = [testing_data1,testing_data2,testing_data3]
  testing_target1 = target[30:50]
  testing_target2 = target[80:100]
  testing_target3 = target[130:150]
  testing_target = [testing_target1, testing_target2, testing_target3]
  return training_data, training_target, testing_data, testing_target


def sigmoid(z):
  return 1 / (1 + np.exp(-z))


def update_mse_grad(test_data,t,W,C):
  grad_mse = np.zeros((C,len(test_data[0])+1))
  for i in range(len(test_data)):
    x = test_data[i]
    x = np.append(x,0)
    x = x.reshape(len(x),1)
    t = t.reshape(len(t),1)
    z = W@x
    g = sigmoid(z)
    grad_mse+= ((g-t)*(g*(1-g)))@x.T

  return grad_mse


def find_W(data, m_iterations,n_classes,alpha):
    C = n_classes
    D = len(data[0][0])
    W_x= np.zeros((C,D))
    W_0 = np.ones((C,1))
    W = np.concatenate((W_x,W_0),axis = 1)

    t_k1 = np.array([1,0,0]) #class1
    t_k2 = np.array([0,1,0]) #class2
    t_k3 = np.array([0,0,1]) #class3
    t  = [t_k1, t_k2, t_k3]

    for m in range(m_iterations):

        W_prev = W
        grad_mse = np.zeros((C,D+1))
```

```
        for (data_k,t_k) in zip(data,t):
          grad_mse += update_mse_grad(data_k,t_k,W_prev,C)
        W = W_prev - alpha*grad_mse
    return W


def test_instance (W,x,solution,confusion):
  x = np.append(x,0)
  Wx = W@x
  answer = np.argmax(Wx)
  confusion[solution][answer] += 1
  if solution == answer:
    return True
  return False



def test_sequence(W,x_sequence,solution_sequence,n_classes,confusion_matrix):
  correct = 0
  wrong = 0
  for nclass in range (len(x_sequence)):
    if test_instance(W,x_sequence[nclass],solution_sequence[nclass],confusion_matrix)
    :
      correct += 1
    else:
      wrong += 1
  return correct,wrong,confusion_matrix




def assignment_1_trainingset(x_sequence,t_sequence,n_classes):
  confusion_matrix = np.zeros((3,3))
  W = find_W(x_sequence, 3000,n_classes, 0.01)
  tot = 0
  correct = 0
  wrong = 0
  for classes in range (3):
    c,w,matrix = test_sequence(W,x_sequence[classes],t_sequence[classes],3,
    confusion_matrix)
    correct += c
    wrong += w
    tot += w+c
  return correct/tot,matrix,W


def assignment_1_testingset(W, training_data, testing_data, testing_solution,
    n_classes):
  confusion_matrix = np.zeros((3, 3))
  tot = 0
  correct = 0
  wrong = 0
  for classes in range(3):
    c,w,matrix = test_sequence(W, testing_data[classes], testing_solution[classes],
    n_classes, confusion_matrix)
    correct += c
    wrong += w
    tot += w+c
  return correct/tot, matrix



def printHistograms(data, features):
  for i in range(0,4):
    plt.figure(i)
    for j in range(3):
```

```
      correctdata = data[50*j : 50*(j+1), i]
      sns.distplot(correctdata, kde=True, norm_hist=True)
    labels = 'setosa', 'veriscolor', 'virginica'
    title = 'histogram for feature: ' + features[i]
    plt.legend(labels)
    plt.title(title)
  plt.show()
```

A.2 main.py

```
#Iris task 1 and 2

from sklearn.datasets import load_iris
import numpy as np

import iris

iris_data = load_iris()
target = iris_data.target
data = iris_data.data

def task_1a():
    training_data1 = data[0:30]
    training_data2 = data[50:80]
    training_data3 = data[100:130]
    training_data = [training_data1,training_data2,training_data3]
    training_target1 = target[0:30]
    training_target2 = target[50:80]
    training_target3 = target[100:130]
    training_target = [training_target1, training_target2, training_target3]

    testing_data1 = data[30:50]
    testing_data2 = data[80:100]
    testing_data3 = data[130:150]
    testing_data = [testing_data1,testing_data2,testing_data3]
    testing_target1 = target[30:50]
    testing_target2 = target[80:100]
    testing_target3 = target[130:150]
    testing_target = [testing_target1, testing_target2, testing_target3]

    #-------------------Training--------------------
    training_ratio, confusion_training, W = iris.assignment_1_trainingset(
    training_data,training_target,3)
    print("Training:")
    print(training_ratio)
    print(confusion_training)

    #-------------------Testing----------------------
    test_ratio, confusion_test = iris.assignment_1_testingset(W, training_data,
    testing_data, testing_target, 3)
    print("Testing:")
    print(test_ratio)
    print(confusion_test)


def task_1b():
    training_data1 = data[20:50]
    training_data2 = data[70:100]
    training_data3 = data[120:150]
    training_data = [training_data1,training_data2,training_data3]
    training_target1 = target[20:50]
    training_target2 = target[70:100]
    training_target3 = target[120:150]
    training_target = [training_target1, training_target2, training_target3]

    testing_data1 = data[0:20]
```

```
    testing_data2 = data[50:70]
    testing_data3 = data[100:120]
    testing_data = [testing_data1,testing_data2,testing_data3]
    testing_target1 = target[0:20]
    testing_target2 = target[50:70]
    testing_target3 = target[100:120]
    testing_target = [testing_target1, testing_target2, testing_target3]

    #------------------Training-------------------
    training_ratio, confusion_training, W = iris.assignment_1_trainingset(
    training_data,training_target,3)
    print("Training:")
    print(training_ratio)
    print(confusion_training)

    #-------------------Testing---------------------
    test_ratio, confusion_test = iris.assignment_1_testingset(W, training_data,
    testing_data, testing_target, 3)
    print("Testing:")
    print(test_ratio)
    print(confusion_test)

def task_2(data, target):
    """
    #Historgrams
    features = ['sepal length', 'sepal width', 'petal length', 'petal width']
    iris.printHistograms(data, features)
    """
    #-----------------Removing 1 feature------------
    data = np.delete(data, 1, axis=1)
    training_data, training_target, testing_data, testing_target = iris.allocate_data
    (data, target)

    print("Training with 3 features: ")
    training_ratio1, confusion_training1, W1 = iris.assignment_1_trainingset(
    training_data,training_target,3)
    print(training_ratio1)
    print(confusion_training1)
    print("Testing with 3 features:")
    test_ratio1, confusion_test1 = iris.assignment_1_testingset(W1, training_data,
    testing_data, testing_target, 3)
    print(test_ratio1)
    print(confusion_test1)

    #------------------Removing 2 features-------------
    data = np.delete(data, 0, axis=1)
    training_data, training_target, testing_data, testing_target = iris.allocate_data
    (data, target)

    print("\n\nTraining with 2 features: ")
    training_ratio2, confusion_training2, W2 = iris.assignment_1_trainingset(
    training_data,training_target,3)
    print(training_ratio2)
    print(confusion_training2)
    print("Testing with 2 features:")
    test_ratio2, confusion_test2 = iris.assignment_1_testingset(W2, training_data,
    testing_data, testing_target, 3)
    print(test_ratio2)
    print(confusion_test2)

    #---------------Removing 3 features-----------------------------
    data = np.delete(data, 0, axis=1)
    training_data, training_target, testing_data, testing_target = iris.allocate_data
    (data, target)

    print("\n\nTraining with 1 feature: ")
```

```
    training_ratio3, confusion_training3, W3 = iris.assignment_1_trainingset(
    training_data,training_target,3)
    print(training_ratio3)
    print(confusion_training3)
    print("Testing with 1 feature:")
    test_ratio3, confusion_test3 = iris.assignment_1_testingset(W3, training_data,
    testing_data, testing_target, 3)
    print(test_ratio3)
    print(confusion_test3)
```

```
def main():

    print("Task 1a")
    task_1a()
    print("\n ------------------------------------ \n")
    print("Task 1b")
    task_1b()

    print("\n ---------------------------------------- \n")
    print("Task 2")
    task_2(data, target)

main()
```

B Vowels Python code

B.1 extract_classes.py

```
import numpy as np


def extract_classes_map(filename):
    class_map = {}

    with open(filename,"r") as file:
        lines = file.read().split("\n")
        try:
            for line in lines:
                line = line.split(" ")

                x_i = []
                for element in range(1,len(line)):
                    if line[element] != '':
                        x_i.append(line[element])

                if str(line[0][-2]+line[0][-1]) not in class_map:
                    class_map[str(line[0][-2]+line[0][-1])] = [x_i]
                else:
                    class_map[str(line[0][-2]+line[0][-1])].append(x_i)
        except IndexError:
            print("End of File")


    return class_map
```

B.2 vowels_task1.py

```
import numpy as np
import extract_classes as ext
from scipy.stats import multivariate_normal
```

```
4
5
6  def sample_mean(dataset):
7      sample_size = len(dataset)
8      N = len(dataset[0])
9      x_sum = [0]*N
10     for vector in range (sample_size):
11         for element in range (N):
12             x_sum[element] += (int(dataset[vector][element]))
13     for element in range(len(x_sum)):
14         x_sum[element] /= sample_size
15
16     return x_sum
17
18 def cov_matrix(dataset):
19     N = len(dataset[0])
20     sample_size = len(dataset)
21     mean = sample_mean(dataset)
22     cov_matrix = np.zeros((N,N))
23     x = np.asfarray(dataset, float)
24     cov_matrix = np.dot((x-mean).T,(x-mean))/(sample_size-1)
25     return cov_matrix
26
27 def make_sequence(sounds):
28     list_of_sounds= []
29     for sound in sounds:
30         list_of_sounds.append(sound)
31     return list_of_sounds
32
33 def generate_mean_cov_map(classes_map,start,end):
34     mean_cov_map = {}
35     list_of_sounds = make_sequence(classes_map)
36     for sound in classes_map:
37         mean_cov_map[sound] =[sample_mean(classes_map[sound][start:end])]
38         mean_cov_map[sound].append(cov_matrix(classes_map[sound][start:end]))
39     return mean_cov_map,list_of_sounds
40
41
42 def generate_x(filename,start,end):
43     test_map = {}
44     train_map={}
45     classes_map = ext.extract_classes_map(filename)
46     for sound in classes_map:
47         train,test = equal_representation(classes_map[sound])
48         test_map[sound] = test
49         train_map[sound] = train
50
51     return train_map,test_map
52
53 def train_test_single_gaussian(start,end,diag = False):
54     train_map,test_map = generate_x("data.dat",start,end)
55     mean_cov_map,sound_list = generate_mean_cov_map(train_map,start,end)
56
57     #---------------------Training-------------------------------
58     probability_vector = np.empty((12,1))
59     correct =  0
60     wrong = 0
61     total = 0
62     confusion_matrix_train = np.zeros((12,12))
63     true_index = 0
64     for iterate_class in train_map:
65         for sample in range(len(train_map[iterate_class])):
66             for i,sound in enumerate(mean_cov_map):
67                 cov_matrix = mean_cov_map[sound][1]
68                 sample_mean = mean_cov_map[sound][0]
69
```

```
                if diag == True:
                    cov_matrix = np.diag(np.diag(mean_cov_map[sound][1]))
                    probability = multivariate_normal.pdf(train_map[iterate_class][sample],mean = sample_mean,cov = cov_matrix)
                else:
                    probability = multivariate_normal.pdf(train_map[iterate_class][sample],mean = sample_mean,cov = cov_matrix,allow_singular = True)
                probability_vector[i] = probability
            predicted_index = np.argmax(probability_vector)
            predicted_sound = sound_list[predicted_index]
            true_guess = iterate_class
            total += 1
            if true_guess == predicted_sound:
                correct += 1
            else:
                wrong += 1
            confusion_matrix_train[true_index][predicted_index] += 1
        true_index += 1
    print("Training : ")
    print(confusion_matrix_train)
    print(correct/total)
    print(total)

    #-----------------------------Testing-------------------------------
    probability_vector = np.empty((12,1))
    correct =  0
    wrong = 0
    total = 0
    confusion_matrix_test = np.zeros((12,12))
    true_index = 0

    for iterate_class in test_map:
        for sample in range(len(test_map[iterate_class])):
            for i,sound in enumerate(mean_cov_map):
                cov_matrix = mean_cov_map[sound][1]
                sample_mean = mean_cov_map[sound][0]

                if diag == True:
                    cov_matrix = np.diag(np.diag(mean_cov_map[sound][1]))
                    probability = multivariate_normal.pdf(test_map[iterate_class][sample],mean = sample_mean,cov = cov_matrix)
                else:
                    probability = multivariate_normal.pdf(test_map[iterate_class][sample],mean = sample_mean,cov = cov_matrix,allow_singular = True)
                probability_vector[i] = probability
            predicted_index = np.argmax(probability_vector)
            predicted_sound = sound_list[predicted_index]
            true_guess = iterate_class
            total += 1
            if true_guess == predicted_sound:
                correct += 1
            else:
                wrong += 1
            confusion_matrix_test[true_index][predicted_index] += 1
        true_index += 1
    print("Testing : ")
    print(confusion_matrix_test)
    print(correct/total)
    print(total)
    return confusion_matrix_train, confusion_matrix_test

def equal_representation(dataset):
    test_set = []
    training_set = []
    for man in range(0,20): #20
        training_set.append(dataset[man])
```

```
    for woman in range(46,66): #20
        training_set.append(dataset[woman])
    for boy in range(93,113): #20
        training_set.append(dataset[boy])
    for girl in range(120,130): #10
        training_set.append(dataset[girl])
    for man in range(20,46): #26
        test_set.append(dataset[man])
    for woman in range(66,93): #27
        test_set.append(dataset[woman])
    for boy in range(113,120): #7
        test_set.append(dataset[boy])
    for girl in range(130,139): #9
        test_set.append(dataset[girl])
    return training_set,test_set
```

B.3 vowels_task2.py

```
import numpy as np
import extract_classes as ext
from scipy.stats import multivariate_normal
from sklearn.mixture import GaussianMixture as GMM
import vowels_task1 as v


def map_join_array(type_map):
    x = []
    for sound in type_map:
        x.extend(type_map[sound[0:70]])
    return np.asfarray(x)
def generate_sound_list(train_map):
    sounds = []
    for sound in train_map:
        sounds.append(sound)
    return sounds

def train_test_GMM(start,end, n_components):
    train_map,test_map = v.generate_x("data.dat",0,70)
    sound_list = generate_sound_list(train_map)


    x_train = map_join_array(train_map)
    x_test = map_join_array(test_map)
    probabilities_train = np.zeros((12,x_train.shape[0]))
    probabilities_test = np.zeros((12,x_test.shape[0]))


    for i,sound in enumerate(train_map):
        gmm = GMM(n_components=n_components, covariance_type='diag', reg_covar=1e-4,
    random_state=0)
        gmm.fit(train_map[sound], sound_list)
        for j in range(n_components):
            N = multivariate_normal(mean=gmm.means_[j], cov=gmm.covariances_[j],
    allow_singular=True)
            probabilities_train[i] += gmm.weights_[j] * N.pdf(x_train)
            probabilities_test[i] += gmm.weights_[j] * N.pdf(x_test)


    confusion_matrix_train = np.zeros((12,12))
    confusion_matrix_test = np.zeros((12,12))

    predict_test = np.argmax(probabilities_test,axis = 0)
    predict_train = np.argmax(probabilities_train,axis = 0)
    true_test = np.asarray([i for i in range (12) for _ in range(70)])
    true_train = np.asarray([i for i in range(12) for _ in range(70)])
    correct =  0
```

```
    wrong = 0
    total = 0
    for index in range(len(predict_test)):
        if int(predict_test[index]) == true_test[index]:
                correct += 1
        else:
            wrong += 1
        confusion_matrix_test[true_test[index]][int(predict_test[index])] += 1
        total += 1
    ratio_test = correct/total
    correct = 0
    wrong = 0
    total = 0
    print("Testing: ")
    print(confusion_matrix_test)
    print("Testing ratio:",ratio_test)
    print("\n--------------------------------\n")
    for index in range(len(predict_train)):
        if int(predict_train[index]) == true_train[index]:
            correct += 1
        else:
            wrong += 1
        confusion_matrix_train[true_train[index]][int(predict_train[index])] += 1
        total += 1
    ratio_training = correct/total
    print("Training: ")
    print(confusion_matrix_train)
    print("training ratio:",ratio_training)

    return confusion_matrix_train, confusion_matrix_test
```

B.4 latexconfusiontable.py

```

#Code taken from github user kristeey
#https://github.com/kristeey/TTT4275_EDC/


vowels = ['ae', 'ah', 'aw', 'eh', 'er', 'ei', 'ih', 'iy', 'oa', 'oo', 'uh', 'uw']

def print_confusion(conf):
    """
    Prints a latex formatted confusion matrix
    """
    print("""\\begin{table}[H]
\\caption{}
\\centering
\\begin{tabular}{|c|lllllllllll|}""")
    conf = conf.astype(int)
    print('\\hline\nclass & '+' & '.join(vowels) + '\\\\' + '\\hline')
    for i, row in enumerate(conf):
        rw = vowels[i]
        for j, elem in enumerate(row):
            rw += ' & '
            if elem == 0:
                rw += '-'
            else:
                rw += str(elem)
        rw += '\\\\'
        if i == 11:
            rw += '\\hline'
        print(rw)
    print("""\\end{tabular}
\\end{table}""")
```

B.5 main.py

```
import vowels_task1 as vt1
import vowels_task2 as vt2
import latexconfusiontable as lct


def main():
    print("Task 1, Full covariance matrix")
    conf1false_train, conf1false_test = vt1.train_test_single_gaussian(0,70,False)
    #lct.print_confusion(conf1false_train)
    #lct.print_confusion(conf1false_test)
    print("\n_______________________________\n")

    print("Task 1, Diagonal covariance matrix")
    conf1true_train, conf1true_test = vt1.train_test_single_gaussian(0,70,True)
    #lct.print_confusion(conf1true_train)
    #lct.print_confusion(conf1true_test)
    print("\n_______________________________\n")

    print("Task 2, 2 gaussians")
    gmm2_train, gmm2_test = vt2.train_test_GMM(0,70,2)
    #lct.print_confusion(gmm2_train)
    #lct.print_confusion(gmm2_test)
    print("\n_______________________________\n")

    print("Task 2, 3 gaussians")
    gmm3_train, gmm3_test = vt2.train_test_GMM(0,70,3)
    #lct.print_confusion(gmm3_train)
    #lct.print_confusion(gmm3_test)


main()
```

References

[1] M.H.Johnsen. *Classification*. 2017.

[2] *MIXTURE MODEL*. `https://en.wikipedia.org/wiki/Mixture_model`. Accessed: 2020-29-04.