ASSIGNMENT 2: CLUSTERING AND MOG.

Machine Learning.

Carolina Raymond Rodrigues.

Student ID: 210607000

ec21209@qmul.ac.uk

INDEX

FIGURES INDEX.

TABLES INDEX.

TASK 1. Load the dataset to your workspace. We will only use the dataset for F1 and F2, arranged into a 2D matrix where the first column will be F1, and the second column will be F2. Produce a plot of F1 against F2. (You should be able to spot some clusters already in this scatter plot.). Include in your report the corresponding lines of your code and the plot.

Unsupervised learning is a Machine Learning task in which algorithms are applied to unlabelled data to find patterns or similarity traits among samples. This model aims to classify the data into subsets without human intervention.

There are different unsupervised learning approaches. However, in this coursework, we are going to focus on **clustering**. This technique will group the given data based on their differences and similarities.

Peterson and Barney's dataset of vowel formant frequencies has been given for this assignment. The goal is to use two features corresponding to the first two formant frequencies (F1 and F2) to allocate each data sample into clusters that distinguish phonemes.

The following figure shows how these two frequencies have been stored in the X_full matrix after previously loading Peterson and Barney's dataset.

```
########################################
# Write your code here
# Store f1 in the first column of X_full, and f2 in the second column of X_full
X_full[:, 0] = f1
X_full[:, 1] = f2


########################################/
```

Figure 1 Code Task 1: store F1 and F2 in X_full matrix.

Once the 2D matrix has stored the required data, Task1.py is run, generating a plot of F1 against F2, as shown in Figure 2a. From this figure, it can be seen how the subsets overlap. Additionally, the spread of the data points of each phoneme is quite different. For instance, samples in phoneme 10 are closer to each other, while the points in phoneme 6 are instead distributed along the horizontal axis. However, as defined in Figure 2b, a rough cluster division can be made.

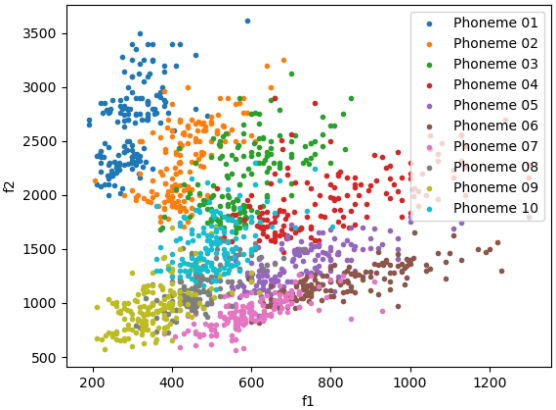

Figure 2a Phonemes scatter plot

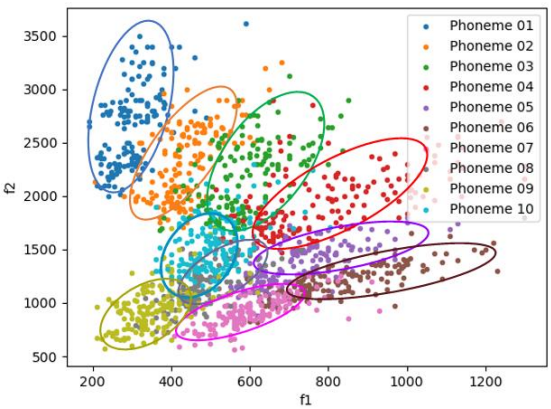

Figure 2b Cluster representation

TASK 2. Train the data for phonemes 1 and 2 with MoGs.
Look at the task 2.py code and understand what it is calculating. Pay particular attention to the initialisation of the means and covariances (also note that it is only estimating diagonal covariances).

The two mentioned characteristics that have been spotted from the plot in the previous section have significant importance while deciding which clustering algorithm must be used to classify the given dataset.

One of the most common algorithms is K-means, where based on the distance to the centroid of each group, the data sample will be assigned to the corresponding cluster. Nevertheless, this algorithm does not work well if sets are not well separated and the data points don't have a similar spread. Thus, this algorithm is discarded for the given problem.

The other algorithm, and the one that we will use, is known as the **Gaussian Mixture of Models (GMM)** or **Mixture of Gaussians (MoGs**), where a probabilistic model associated with each cluster is used to solve the problem without having concerns about the shape and distance of the subgroups.

MoGs aim is used to determine to which probability distribution a sample belongs. Therefore, given the data point (x) and the parameters of the Gaussian (θ), we want to learn to which cluster it belongs (c_k).

$$p(c_k \mid x, \theta) \tag{1}$$

Nevertheless, we do not know the values of the Gaussian parameters (i.e., $\theta = \{\pi, \mu, \Sigma\}$), where π is the mixture of coefficients, μ the mean and Σ the covariance). Consequently, the **EM algorithm** will allow us to infer those parameter values.

The procedure that the EM algorithm follows is:

1. For a given number of clusters k, randomly place k Gaussians $\{(\pi_o, \mu_0, \Sigma_o), (\pi_1 \mu_1, \Sigma_1) \ldots (\pi_{k-1}, \mu_{k-1}, \Sigma_{k-1})\}$.

2. **E-step (Expectation)**: for each data sample (x_i), and the probability distributions previously obtained, estimate the probability that it belongs to each one of the clusters (Equation 2). Then normalise these values over k clusters.

$$p(c_k \mid xi, \theta_k) = \sum_{k=1}^{K} \frac{p(c_k)}{(2\pi)^{D/2} \, |\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2} (x_i - \mu_k)^T \, \Sigma_k^{-1} \, (x - \mu_k)\right) \tag{2}$$

   Where D is the dimension of the x vector.

3. **M-step (Maximisation)**: for each Gaussian distribution, update the parameters $(\pi_k, \mu_k \; and \; \Sigma_k)$ using the weighted data points.

$$\mu_k^{(i+1)} = \frac{\Sigma_n \, q_{n,k} \, x_n}{\Sigma_n \, q_{n,k}} \tag{3}$$

$$\Sigma_k^{i+1} = \frac{\Sigma_n \, q_{n,k} \left(\left(x_n - \mu_k^{(i+1)} \right) \left(x_n - \mu_k^{(i+1)} \right)^T \right)}{\Sigma_n \, q_{n,k}} \tag{4}$$

$$\pi_k^{(i)} = \frac{1}{N}\sum_n q_{n,k} \quad (5)$$

Where $q_{n,k}$ are the probabilities estimated in the E-step.

4. Repeat steps 2 and 3 until convergence. This will guarantee that the model's likelihood function is maximised and consequently, the estimated Gaussian parameters are the ones that fit best our model.
   It is essential to highlight that the algorithm will converge to a **local minimum,** so **initialisation matters**. Since initialisation is done randomly, it is considered a good practice, to repeat the process a couple of times and then choose the model that fits best our dataset.

The previous steps are included in the given code (Task_2.py). Variable initialisation takes place on lines 87-108, where the mixture of coefficients (π) are set to 1/k, the covariance (Σ) is calculated with the sample data in X_full, and the means (μ) are randomly picked from the data samples. Hence, different clusters can be obtained on each run since the Gaussian parameters might be different, and therefore, convergence can be achieved on different local minimums.

The EM algorithm is applied from lines 114 – 133. The first five lines correspond to the E-step, and the rest belong to the M-step.

Generate a dataset X phoneme 1 containing only the F1 and F2 for the first phoneme.

In this section, all the samples corresponding to phoneme 1 must be stored in the X_phoneme matrix, as shown in Figure 3. For that, a matrix Nx2 matrix is created where N corresponds to the number of data samples labelled as phoneme p_{id} (phoneme 1 in this case). Consequently, each of the samples in the Peterson and Barney dataset that have the same phoneme ID as p_{id} is saved in the X_phoneme matrix.

```
54    X_phoneme = np.zeros((np.sum(phoneme_id==p_id), 2))
55    idx=0
56    for i in range (len(phoneme_id)):
57        if phoneme_id[i] == p_id:
58            X_phoneme[idx] = X_full[i, :]
59            idx +=1
60
```

Figure 3 Store corresponding phonemes in X_phoneme array.

Run task 2.py on the dataset using K=3 Gaussians (run the code several times and note the differences.) Save your MoG model: this should comprise the variables mu, s, and p.

Task 2 has been re-run three times. The obtained results are shown in the following Figure and Tables 1, 2 and 3. In the latter ones, it can be seen that even though the algorithm starts with different mean values, after 100 iterations, clusters will have similar, if not the same, probability distributions. The reason behind this is because, with k=3, simpler similarities and differences are captured among data, which ends up leading to grouping samples in a more general way and therefore, is expected to end up having similar Gaussian distributions that best fit tour data.

Ideally, since MoGs converges to local minima, we should choose the model that provides a better fit. Still, since the parameter value does not differ much, any of the results obtained on the re-runs can be used for future training.

Additionally, for every iteration, the code is plotting the Gaussians. It can be seen that after some iterations (around 25-40, depending on initialisation), the Gaussians do not change much. Meaning that the model's log-likelihood does not increase any further, and therefore our model has converged. Hence, we could decrease the number of iterations and save some computational time.

Moreover, as observable in the plots, most samples fall inside the clusters. However, some points don't, like, for instance, the one corresponding to F1=265 and F2=2000. Nevertheless, suppose we estimate the probability of this point belonging to each cluster. In that case, we will obtain a higher value for the group coloured in green in Figure 4a, meaning that in comparison to the others, it has more common features to the data points of the green subset.

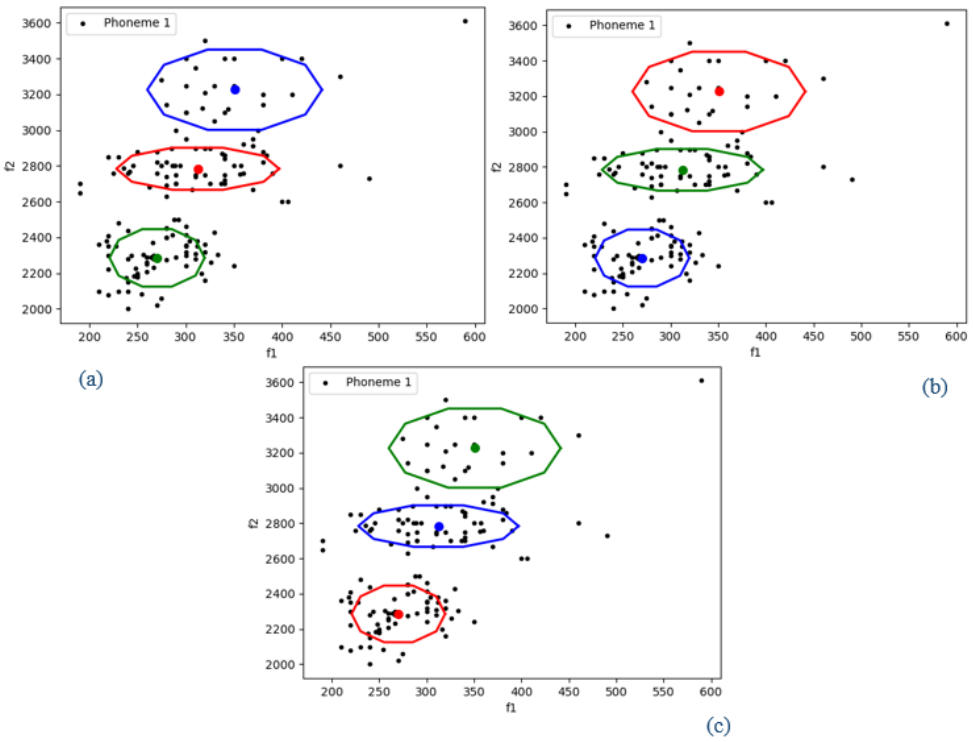

Figure 4 Train phoneme 1 with k=3 using MoGs

Run	μ_1		μ_2		μ_3	
	F1	**F2**	**F1**	**F2**	**F1**	**F2**
1	312.58	2783.87	270.39	2285.46	350.84	3226.20
2	350.84	3226.34	312.59	2783.90	270.4	2285.46
3	270.39	2285.46	350.84	3226.34	312.60	2783.89

Table 1 Means after training phoneme 1 with k=3 using MoGs

Run	Σ_1		Σ_2		Σ_3	
	F1	**F2**	**F1**	**F2**	**F1**	**F2**
1	3652.68	7654.32	1213.73	14278.42	4102.16	27866.24
2	4102.87	27829.55	3562.59	7657.84	1213.73	14278.42
3	1213.74	14278.42	4102.88	27829.55	356.60	7657.85

Table 2 Covariances after training phoneme 1 with k=3 using MoGs

Run	p_1	p_2	p_3
1	0.3809	0.4351	0.1839
2	0.1838	0.3809	0.4351
3	0.4351	0.1838	0.38099

Table 3 Weighted mixture of Gaussians after training phoneme 1 with k=3 using MoGs

Run task 2.py on the dataset using K=6.

Like in the previous question, the data from phoneme 1 is trained. However, this time, samples will be clustered into 6 subsets. The results obtained can be viewed in Figure 5 and Tables 4, 5 and 6.

In comparison to the results, we obtained for k=3, increasing the number of clusters means that a higher number of similarities and differences among the samples will be captured, contributing to a more meaningful distinction. Interestingly, as can be seen in Figures 5a and 5b., the shape of the clusters differ in each run. Since more fine-tuning is made, initialising the Gaussian parameters differently will lead to considering different traits during the subgrouping process of each run. Consequently, different clusters will be obtained.

With k=3, phonemes samples (such as datum F1=460, F2=3350) that had a low likelihood of being classified as its corresponding label, have now with k=6, a higher probability of being classified correctly. However, increasing the number of clusters might lead to overfitting, so we must take care. The elbow method will be useful to choose what k value, will provide the best modelling of the data.

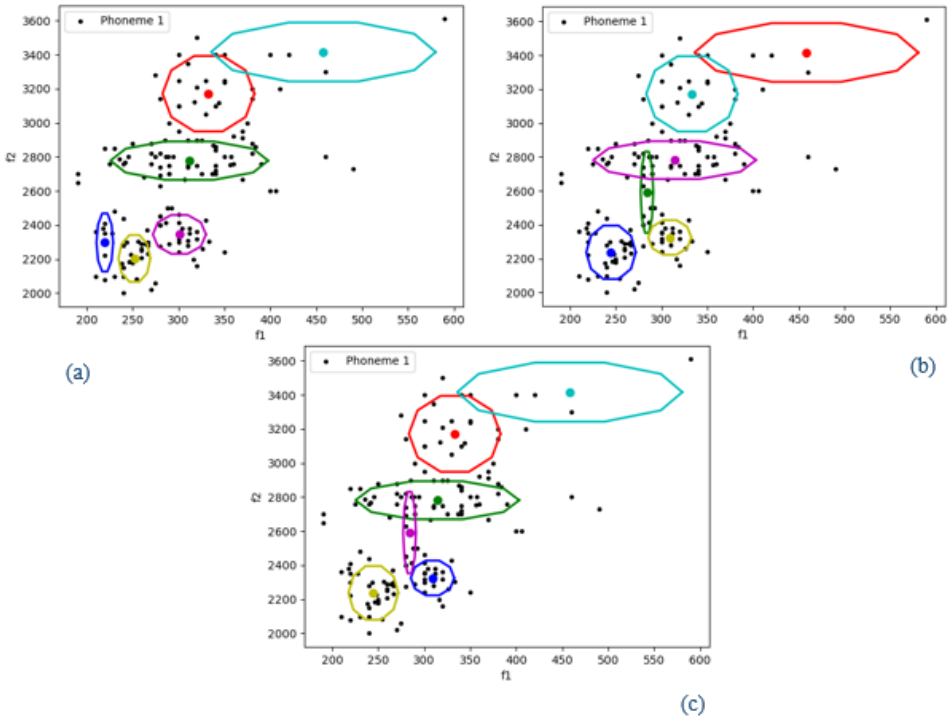

Figure 5 Train phoneme 1 with k=6 using MoGs.

Run	Means	F1	F2
1	μ_1	332.71	3171.73
	μ_2	311.86	2778.36
	μ_3	219.88	2297.64
	μ_4	457.82	3416.42
	μ_5	301.07	2345
	μ_6	252.08	2203.97
2	μ_1	458.53	3416.87
	μ_2	284.25	2592.1
	μ_3	244.96	2237.63
	μ_4	333.31	3172.87
	μ_5	314.45	2781.89
	μ_6	309.17	2325.25
3	μ_1	333.29	3171.87
	μ_2	314.42	2781.74
	μ_3	309.175	2325.24
	μ_4	458.28	3416.54
	μ_5	284.24	2591.74
	μ_6	244.96	2237.63

Table 4 Means after training phoneme 1 with k=6 using MoGs.

Run	Means	F1	F2
1	Σ_1	1251.79	27150.57
	Σ_2	3683.84	7057.16
	Σ_3	41.36	16137.45
	Σ_4	7474.58	16556.79
	Σ_5	417.61	7209.52
	Σ_6	150.10	10443.91
2	Σ_1	7477.89	16617.44
	Σ_2	24.09	31763.4
	Σ_3	366.03	13749.32
	Σ_4	1242.86	27284.38
	Σ_5	3957.34	6898.71
	Σ_6	278.03	5764.28
3	Σ_1	1241.22	27480.69
	Σ_2	3960.71	6884.6
	Σ_3	277.82	5763.81
	Σ_4	7481.62	16607.61
	Σ_5	24.07	31698.59
	Σ_6	366.04	13748.7

Table 5 Covariances after training phoneme 1 with k=6 using MoGs.

Run	p_1	p_2	p_3	p_4	p_5	p_6
1	0.1721	0.3676	0.0645	0.0261	0.2050	0.1642
2	0.0258	0.0695	0.2481	0.1711	0.3351	0.1502
3	0.1716	0.3347	0.1502	0.0259	0.069	0.24801

Table 6 Weighted mixture of Gaussians after training phoneme 1 with k=6 using MoGs.

Repeat steps 2-4 for the second phoneme

With K=3

In this section, the same procedure followed in the previous questions is applied. However, this time, the value of the phoeneme_id variable that can be seen in Figure 3 will be 2 and k=3. The results obtained are summarised in Tables 7, 8 and 9 and Figure 6.

Like before, clusters end up having similar probability distributions after 100 iterations, which reinforces what was said previously. With k=3, even though initialisation parameters in each run are different, more general distinctions among data are made, leading to similar probability distributions that best fit the data.

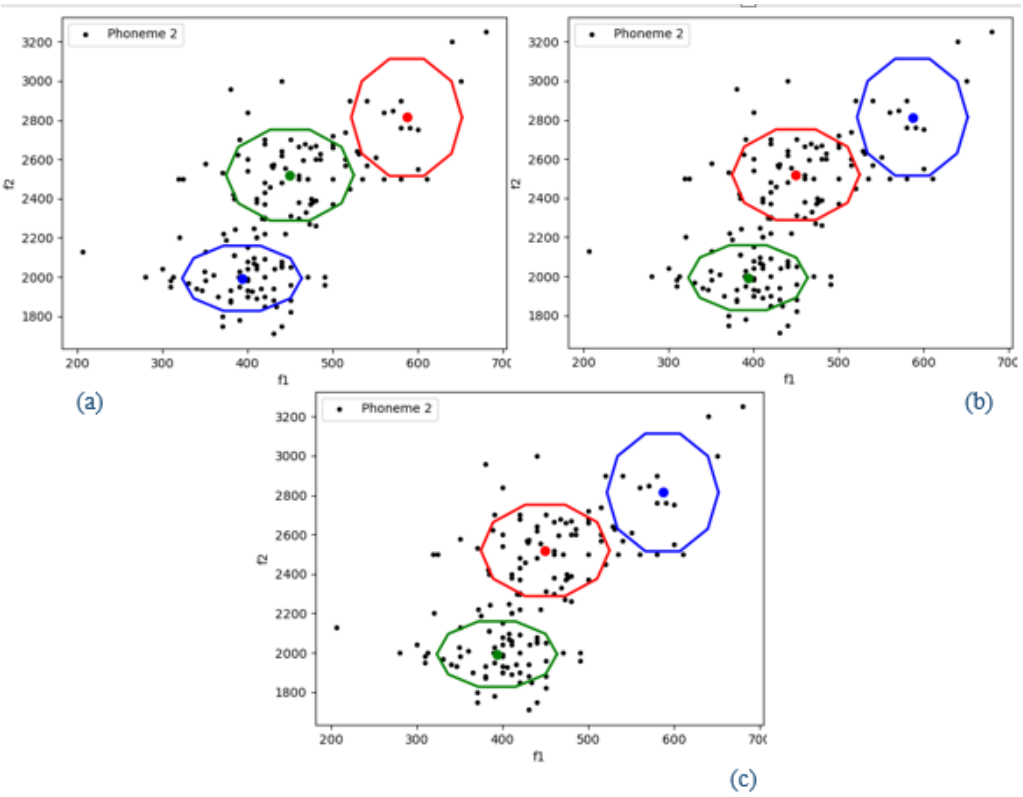

Figure 6 Train phoneme 2 with k=3 using MoGs

Run	μ_1		μ_2		μ_3	
	F1	**F2**	**F1**	**F2**	**F1**	**F2**
1	586.56	2814.16	449.67	2519.68	393.45	1993.08
2	449.67	2519.66	393.45	1993.07	586.54	2814.10
3	449.67	2519.68	393.45	1993.08	586.56	2814.15

Table 7 Means after training phoneme 2 with k=3 using MoGs

Run	Σ_1		Σ_2		Σ_3	
	F1	**F2**	**F1**	**F2**	**F1**	**F2**
1	2111.7	49424.43	2809.13	29720.95	2454.92	15263.17
2	2808.64	29722.51	2454.96	15261.88	2112.10	49424.47
3	2809.08	29721.12	2454.93	15263.03	211.74	49424.44

Table 8 Covariances after training phoneme 2 with k=3 using MoGs

Run	p_1	p_2	p_3
1	0.0948	0.4699	0.4352
2	0.4699	0.4351	0.0949
3	0.4699	0.4352	0.0949

Table 9 Weighted mixture of Gaussians after training phoneme 2 with k=3 using MoG.

With K=6

Like in the previous question, the data from phoneme 2 is trained. However, this time, samples will be clustered into 6 subsets. The results obtained can be viewed in Figure 7 and Tables 10, 11 and 12.

When the number of k components increases, something similar to phoneme 1 occurs. More similarities and distinctive traits are considered among the data. Therefore, initialisation affects clustering, since different features may be considered while obtaining the probability distributions that best fit the data.

Finally, an appreciation has been made on the data set, where the sample F1=200 vs F2=2150 could be considered as an **outlier** or **anomaly.** Anomalies are produced because the data has been contaminated with noise, maybe due to incorrect labelling or feature corruption. Ideally, these anomalies should have been removed with a pre-processing step to avoid poor training and increase the model's accuracy. However, we must highlight that considering that data point as an anomaly will depend on the nature of the problem.

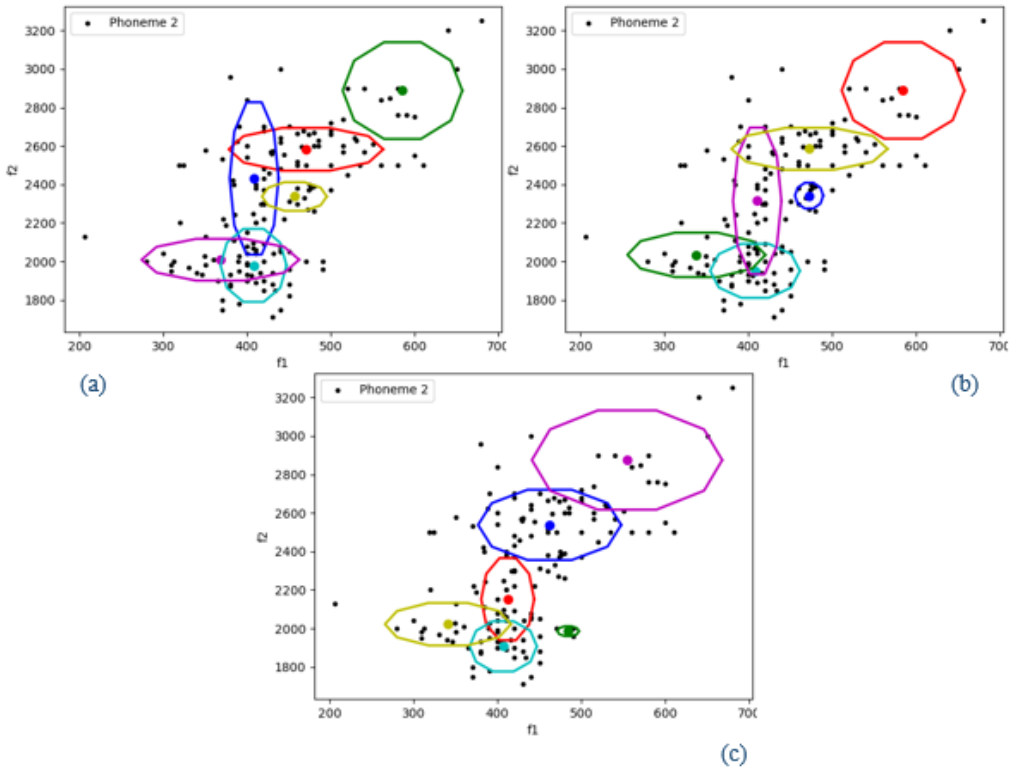

Figure 7 Train phoneme 2 with k=6 using MoGs

Run	Means	F1	F2
1	μ_1	470.35	2582.7
	μ_2	585.38	2887.81
	μ_3	408.535	2431.59
	μ_4	407.82	1979.76
	μ_5	368.28	2009.02
	μ_6	456.53	2337.61
2	μ_1	584.25	2888.42
	μ_2	338.31	2034.34
	μ_3	472.42	2341.98
	μ_4	408.12	1951.45
	μ_5	410.72	2314.72
	μ_6	472.92	2584.98
3	μ_1	412.25	2152.09
	μ_2	484.14	1985.95
	μ_3	462.16	2538.02
	μ_4	406.95	1907.08
	μ_5	554.3	2875.03
	μ_6	341.19	2022.19

Table 10 Means after training phoneme 2 with k=6 using MoGs

Run	Means	F1	F2
1	Σ_1	4243.80	6802.29
	Σ_2	2541.23	34880.12
	Σ_3	424.80	86513.29
	Σ_4	773.81	19875.34
	Σ_5	4415.73	6498.57
	Σ_6	746.05	3057.96
2	Σ_1	2664.17	34689.47
	Σ_2	3367.72	7370.22
	Σ_3	140.97	2456.83
	Σ_4	1423.36	10889.39
	Σ_5	414.73	80186.85
	Σ_6	4323.95	6573.67
3	Σ_1	496.6	25219.06
	Σ_2	83.30	364.64
	Σ_3	3626.75	18444.17
	Σ_4	799.19	9413.84
	Σ_5	6414.04	36720.24
	Σ_6	2814.94	6734.27

Table 11 Covariances after training phoneme 2 with k=6 using MoGs

Run	p_1	p_2	p_3	p_4	p_5	p_6
1	0.2842	0.079	0.1235	0.2561	0.1597	0.097
2	0.0799	0.0933	0.0675	0.2505	0.2377	0.2711
3	0.1949	0.0181	0.4056	0.1683	0.0959	0.1172

Table 12 Weighted mixture of Gaussians after training phoneme 2 with k=6 using MoGs

TASK3. Use the 2 MoGs (K=3) learned in task 2 to build a classifier discriminating between phonemes 1 and 2. Classify using the Maximum Likelihood (ML) criterion (feel free to hack parts from the MoG code in task 2.py so that you calculate the likelihood of a data vector for each of the two MoG models) and calculate the miss-classification error. Remember that classification under the ML compares p (x; θ1), where θ1 are the parameters of the MoG learned for the first phoneme, with p(x; θ2), where θ2 are the parameters of the MoG learned for the second phoneme. Repeat this for K = 6 and compare the results. Include the lines of the code you wrote in your report, explain what the code does, and comment on the differences in the classification performance.

In the previous section, the means, covariances, and weighted mixture of Gaussians of each phoneme and k component had been stored in a .npy file. This trained data will be uploaded (lines 86-89 and 99-102, Task 3) to train a matrix X_full that will contain all the data from phonemes 1 and 2 (lines 50-60, Task 3).

After, the uploaded data of each phoneme will be used to classify each of the data samples contained in X_full. For that, the data will be used as arguments in the get_predictions function (lines 91-93 and 95-97, Task3). The output of this function will be a matrix where each column will state the probability that a data point belongs to one of the k components. The obtained probabilities of each row will be summed and compared with the sum of probabilities of the other phoneme. Ideally, if the classification model has been trained correctly, the probability of the data obtained in the correct phoneme will be higher than the one obtained in the wrong phoneme (lines 108-116, Task 3).

The predicted values will then be compared to the ground truth labels, and the accuracy of the model is estimated using the scikit-learn Python library (lines 117-119, Task 3).

The exact process is followed for k=6, and the results obtained for each component value are 95.07% and 96.05%, respectively. These values were expected since, firstly, a higher number of clusters means that more patterns or traits will be analysed, and therefore, a better distinction among phonemes will be made.

```
82  #phoneme 1 k = 3 load pretrained data.                                    ⚠ 10 ⚠ 63
83  data01_03 = np.load('data/GMM_params_phoneme_01_k_03.npy', allow_pickle=True)
84  data01_03 = dict(enumerate(data01_03.flatten(), 0))
85
86  #Save means, covariances and weighted mixture of gaussians of the pretrained data of phoneme 1, k=3
87  mu_01_03 = data01_03[0]["mu"]
88  s_01_03 = data01_03[0]["s"]
89  p_01_03 = data01_03[0]["p"]
90
91  #Calculate how likely each data sample belongs to the k gaussians of phoneme 1.
92  Z_01_03= get_predictions(mu_01_03, s_01_03, p_01_03, X_phonemes_1_2)
93  Z_13= np.sum(Z_01_03, axis=1)
94
95  #phoneme 2 k = 3 load pretrained data.
96  data02_03 = np.load('data/GMM_params_phoneme_02_k_03.npy', allow_pickle=True)
97  data02_03 = dict(enumerate(data02_03.flatten(), 0))
98
99  #Save means, covariances and weighted mixture of gaussians of the pretrained data of phoneme 1, k=3
100 mu_02_03 = data02_03[0]["mu"]
101 s_02_03 = data02_03[0]["s"]
102 p_02_03 = data02_03[0]["p"]
```

```
#Calculate how likely each data sample belongs to the k gaussians of phoneme 2.
Z_02_03= get_predictions(mu_02_03, s_02_03, p_02_03, X_phonemes_1_2)
Z_23= np.sum(Z_02_03, axis=1)

#For each data sample, if the probability obtained for phoneme 1 is higher than for phoneme 2
# the value of the prediction of that data sample will be 1, otherwise 2.
prediction_03= np.zeros(((np.sum(phoneme_id==1)+ np.sum(phoneme_id==2)), 1))
for i in range(len(X_phonemes_1_2)):
    if Z_13[i] > Z_23[i]:
        prediction_03[i] = 1
    else:
        prediction_03[i] =2

#measure the accuracy of the model using the sklearn.
accuracy_03= accuracy_score(groundTruth, prediction_03, normalize=True, sample_weight=None)

print('Accuracy using GMMs with {} components: {:.2f}%'.format(k, accuracy_03*100))
```

Figure 8 Task 3, k=3

```
##phoneme 1 k = 6 load pretrained data.
data01_06 = np.load('data/GMM_params_phoneme_01_k_06.npy', allow_pickle=True)
data01_06 = dict(enumerate(data01_06.flatten(), 0))

#Save means, covariances and weighted mixture of gaussians of the pretrained data of phoneme 1, k=6
mu_01_06 = data01_06[0]["mu"]
s_01_06 = data01_06[0]["s"]
p_01_06 = data01_06[0]["p"]

#Calculate how likely each data sample belongs to the k gaussians of phoneme 1.
Z_01_06= get_predictions(mu_01_06, s_01_06, p_01_06, X_phonemes_1_2)
Z_16= np.sum(Z_01_06, axis=1)

#phoneme 2 k = 6 load pretrained data.
data02_06 = np.load('data/GMM_params_phoneme_02_k_06.npy', allow_pickle=True)
data02_06 = dict(enumerate(data02_06.flatten(), 0))

#Save means, covariances and weighted mixture of gaussians of the pretrained data of phoneme 2, k=6
mu_02_06 = data02_06[0]["mu"]
s_02_06 = data02_06[0]["s"]
p_02_06 = data02_06[0]["p"]

#Calculate how likely each data sample belongs to the k gaussians of phoneme 2.
Z_02_06= get_predictions(mu_02_06, s_02_06, p_02_06, X_phonemes_1_2)
Z_26= np.sum(Z_02_06, axis=1)

#For each data sample, if the probability obtained for phoneme 1 is higher than for phoneme 2
# the value of the prediction of that data sample will be 1, otherwise 2.
prediction_06= np.zeros(((np.sum(phoneme_id==1)+ np.sum(phoneme_id==2)), 1))
for i in range(len(X_phonemes_1_2)):
    if Z_16[i] > Z_26[i]:
        prediction_06[i] = 1
    else:
        prediction_06[i] =2

#Measure the accuracy of the model using the sklearn.
accuracy_06= accuracy_score(groundTruth, prediction_06, normalize=True, sample_weight=None)
print('Accuracy using GMMs with {} components: {:.2f}%'.format(k, accuracy_06*100))
########################################/
```

Figure 9 Task 3, k=6

TASK 4. Create a grid of points that spans the two datasets. Classify each point in the grid using one of the classifiers. Creating a classification matrix, M, whose elements are either 1 or 2. M(i, j) is 1 if the point x1 is classified as belonging to phoneme 1, and is 2 otherwise. x1 is a vector whose elements are between the minimum and the maximum value of F1 for the first two phonemes, and x2 similarly for F2. Display the classification matrix. Include the lines of code in your report, comment on them, and display the classification matrix.

In this section, a grid 1900 x 490 grid will be created. The first dimension of the grid will belong to f2, and values will span from 1710 to 3610, while the second dimension of the grid belongs to f1 and will span values from 190 to 680.

Each one of the points of the grid will be included in an array and passed as an argument of the get_predictions function to classify the data. This function must include as arguments the mean, the covariance, and the weighted mixture of Gaussians. Therefore, as it can be seen from lines 76 to 88 in Task 4, just as we did in Task 3, the pretrained data of k=3 for each phoneme will be used to classify each point of the grid following a similar procedure as in Task 3 (Figure 10).

```
72  # Write your code here
73  #phoneme 1 k = 3.
74  data01 = np.load('data/GMM_params_phoneme_01_k_03.npy', allow_pickle=True)
75  data01 = dict(enumerate(data01.flatten(), 0))
76
77  mu_01 = data01[0]["mu"]
78  s_01 = data01[0]["s"]
79  p_01 = data01[0]["p"]
80
81  #phoneme 2 k = 3.
82  data02 = np.load('data/GMM_params_phoneme_02_k_03.npy', allow_pickle=True)
83  data02 = dict(enumerate(data02.flatten(), 0))
84
85  mu_02 = data02[0]["mu"]
86  s_02 = data02[0]["s"]
87  p_02 = data02[0]["p"]
```

```
99  M = np.zeros((N_f2, N_f1))
100 for i in range(N_f2):
101     for j in range(N_f1):
102         pred1 = get_predictions(mu_01, s_01, p_01, np.array([[j+min_f1, i+min_f2]]))
103         pred2 = get_predictions(mu_02, s_02, p_02, np.array([[j+min_f1, i+min_f2]]))
104
105         pred1T = np.sum(pred1, axis=1)
106         pred2T = np.sum(pred2, axis=1)
107
108         if pred1T< pred2T:
109             M[i, j]= 1.0
110         else:
111             M[i,j] = 0.0
```

Figure 10 Task 4 code.

Next, a classification matrix is displayed and can be seen in the following figure. As it is observable, a decision boundary that separates phoneme 1 from phoneme 2 has been plotted, where most of the phoneme 1 and phoneme 2 samples of Peterson and Barney's dataset fall on the correct side of the plot. However, some outliers can be seen, such as the phoneme 2 sample F1= 200 vs F2=2215. This suggests that this phoneme might have been misclassified as

phoneme 2 instead of phoneme 1. Therefore, it would be beneficial for our model to make a pre-processing first that eliminates these anomalies.

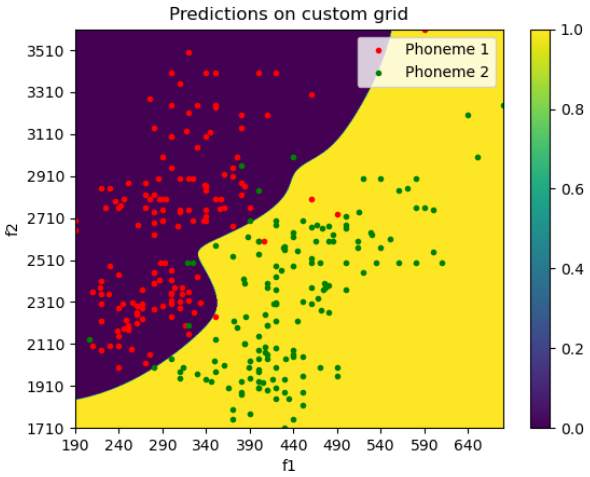

Figure 11 Classification matrix.

TASK 5. In the code of task 5.py, an MoG with a full covariance matrix fits the data. Now, create a new dataset that will contain 3 columns, as follows:

$$X = [F1, F2, F1 + F2]$$

Fit an MoG model to the new data. What is the problem that you observe? Explain why. Suggest ways of overcoming the singularity problem and implement them. Include the lines of code in your report and graphs/plots to support your observations.

A new X_full matrix is created with 3 columns. Each one will include f1, f2 and f1+f2 as shown in lines 37-40 in the code.

An error is observed once the new dataset is fitted in an MoG model (Figure 12).

```
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

Figure 12 Fit an MoG with a singularity problem.

The reason behind this error is that the density estimation has been overfitted. The third column vector is a linear combination of columns 1 and 2 since it is the sum of these two vectors. Therefore, the covariance determinant will be 0., and the probability calculated for each one of the clusters (Equation 2) will have a 0 in the denominator, leading to a wrong estimation.

There are different ways of solving this problem. The ultimate goal is to add some noise to the singular matrix and make it non-singular. Therefore, just as it has been coded in Figure 13, lines 138-140, we could add some constraints in the forms of the covariance matrix, where the covariance matrix will be multiplied by its identity matrix.

Another way of solving the overfitting problem is to regularise the covariance matrix by adding a diagonal matrix $\sum_0 \sigma_2^0 I$

```
X_full[:,0] =f1
X_full[:,1] = f2
X_full[:,2] = np.add(f1, f2)

X_phoneme = np.zeros(((np.sum(phoneme_id == 1) + np.sum(phoneme_id == 2)), 3
idx=0

for i in range(len(phoneme_id)):
    if phoneme_id[i] ==1 or phoneme_id[i] ==2:
        X_phoneme[idx] = X_full[i, :]
        idx +=1

        # Write your code here
        # Suggest ways of overcoming the singularity
        Imatrix = np.identity(len(s))
        s = s*Imatrix

```

Figure 13 Task 5 code.

Task 5 has been re-run 3 times, each one of the re-runs led to different distributions that best fitted the given data, as can be seen in the following Figure. The main reason behind this is again initialisation. Although k=3, and a general distinction among traits is made, in this section the randomness has increased. Now, not only the means are obtained randomly, but the covariance matrix does not only consider the diagonal covariance as before, increasing, therefore, the effect of initialisation.

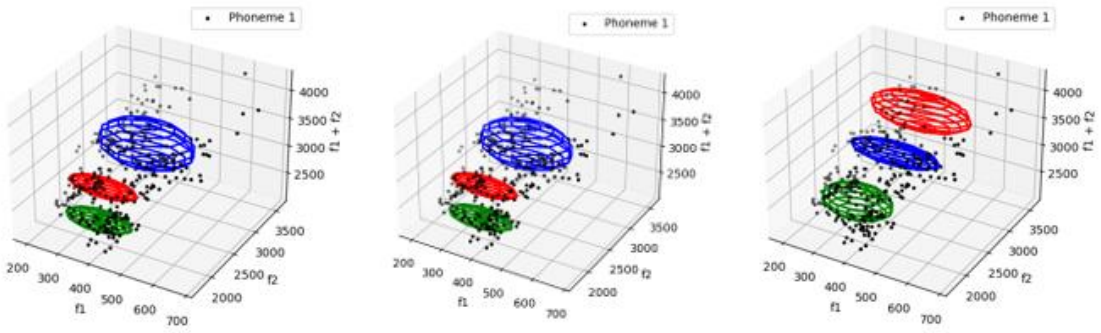

Figure 14 Clusterisation without singularities Task 5.