ECS708 Machine Learning Assignment 2: Clustering and MoG

Aim: The aim of this assignment is to become familiar with clustering using the Mixture of Gaussians model

Task 1.

- Loading the dataset to the workspace.
- In order to accomplish a 2D matrix X_full was created within a task_1.py, containing the values of F1 in the first column and values of F2 in second column. This is shown in the code given below.

```
########################################
# 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

X_full = X_full.astype(np.float32)
```

- When the code was run this new matrix plot F1 against F2 was produced as shown in Fig.1. It can be seen this plot contains all phonemes of F1 and F2 and it is represented by different colored points.
- The distinction between the different phonemes as each phoneme can be seen to group in different part of the plot.

```
# Create array containing only samples that belong to phoneme 1
# X_phoneme_1 = np.zeros((np.sum(phoneme_id==1), 2))
# X_phoneme = ...
#####################################
print(np.sum(phoneme_id == 1))

X_phoneme_index = np.where(phoneme_id == p_id)
X_phoneme_1 = np.take(X_full,X_phoneme_index,axis=0).reshape(X_phoneme_index[0].size,2)
```

Output:

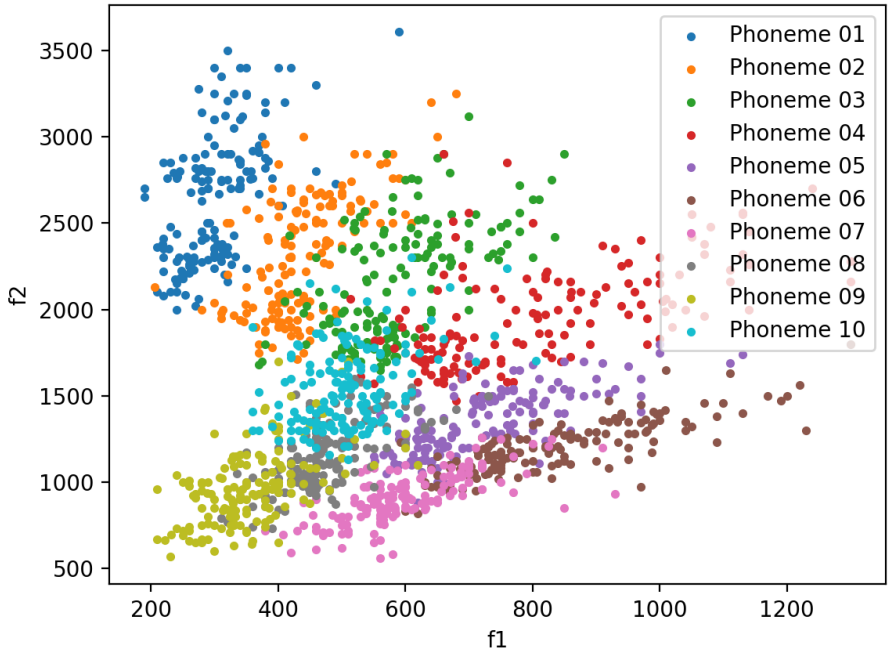


Fig.1 Plot of f1 against f2, for all phonemes

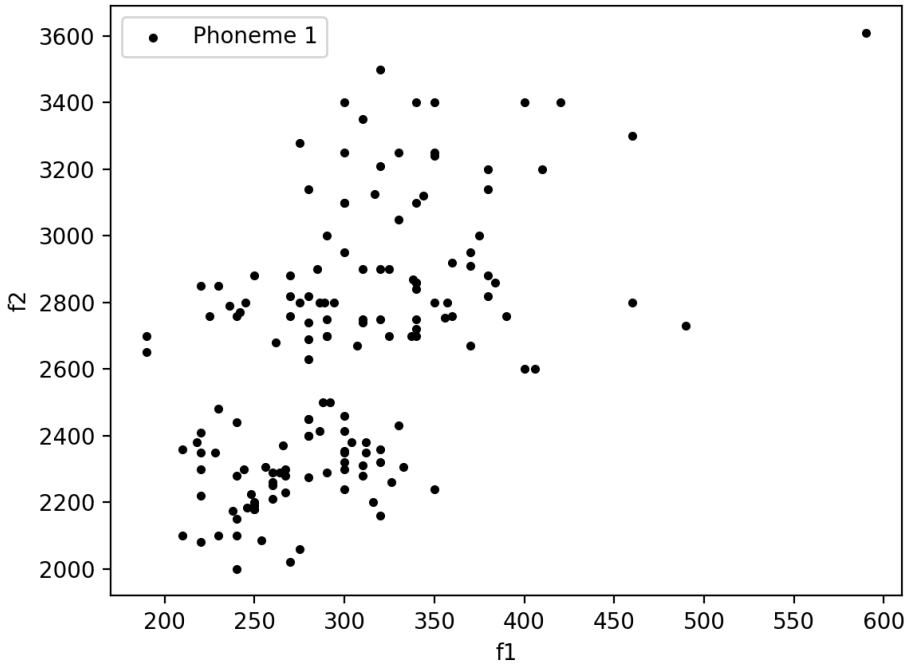


Fig.1.1 Plot of f1 against f2, for all phonemes

Task 2.

- ⇨ We generate the matrices containing f1 and f2 for phonemes 1 and 2
- ⇨ Multiple runs of the model produce different clusters.
- ⇨ This task is to train the data or phonemes 1 and 2 using Multiple of Guassians (MoGs)

```
########################################
# 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

X_full = X_full.astype(np.float32)
```

```
# We will train a GMM with k components, on a selected phoneme id which is stored in variable "p_id"

# number of GMM components
k = 6
#k = 3
# you can use the p_id variable, to store the ID of the chosen phoneme that will be used (e.g. phoneme 1, or phoneme 2)
p_id = 2
#p_id = 1
```

```
X_phoneme_index = np.where(phoneme_id == p_id)
X_phoneme = np.take(X_full,X_phoneme_index,axis=0).reshape(X_phoneme_index[0].size,2)
```

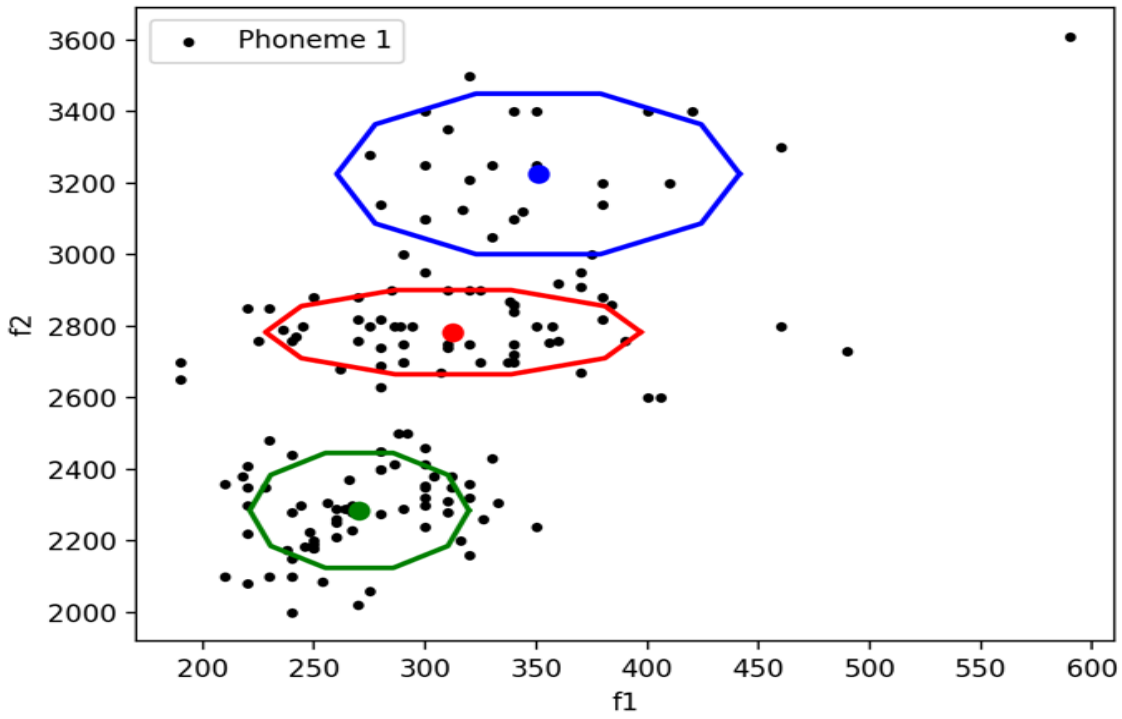


Fig.2.1(a) Phoneme 1, with k=3. First run

```
Implemented GMM | Mean values
[ 312.591   2783.8967]
[ 270.3952  2285.4653]
[ 350.84442 3226.3333 ]

Implemented GMM | Covariances
[[3562.60141748     0.          ]
 [    0.          7657.6974084 ]]
[[ 1213.73838664     0.          ]
 [    0.          14278.42136779]]
[[ 4102.84466181     0.          ]
 [    0.          27831.10766425]]

Implemented GMM | Weights
[0.38099175 0.43514437 0.18386389]
```

Fig.2.1(b) Phoneme 1, with k=3. First run

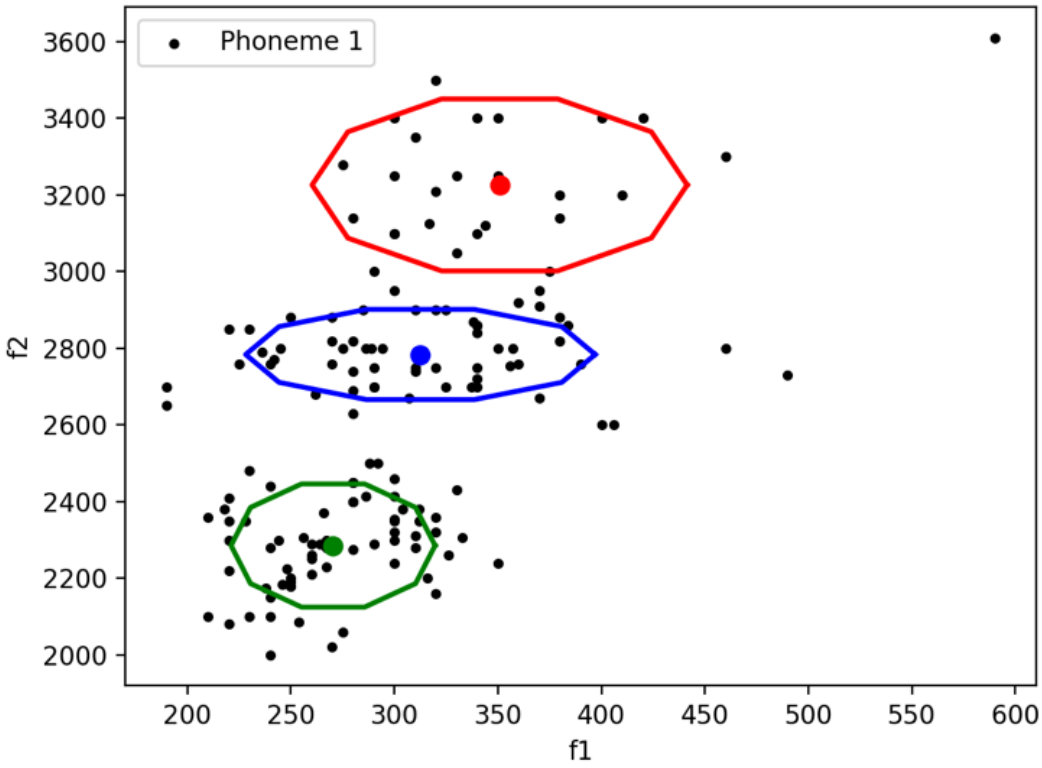


Fig.2.2(a) Phoneme 1, with k=3. Second run

```
Implemented GMM | Mean values
[  350.8446  3226.3394]
[  270.3952  2285.4653]
[  312.59125 2783.898  ]

Implemented GMM | Covariances
[[  4102.875375         0.          ]
 [     0.          27829.54221423]]
[[  1213.73843494       0.          ]
 [     0.          14278.4202995 ]]
[[3562.59743765     0.          ]
 [    0.          7657.84897245]]

Implemented GMM | Weights
[0.18386038 0.43514434 0.38099528]
```

Fig.2.2(b) Phoneme 1, with k=3. Second run

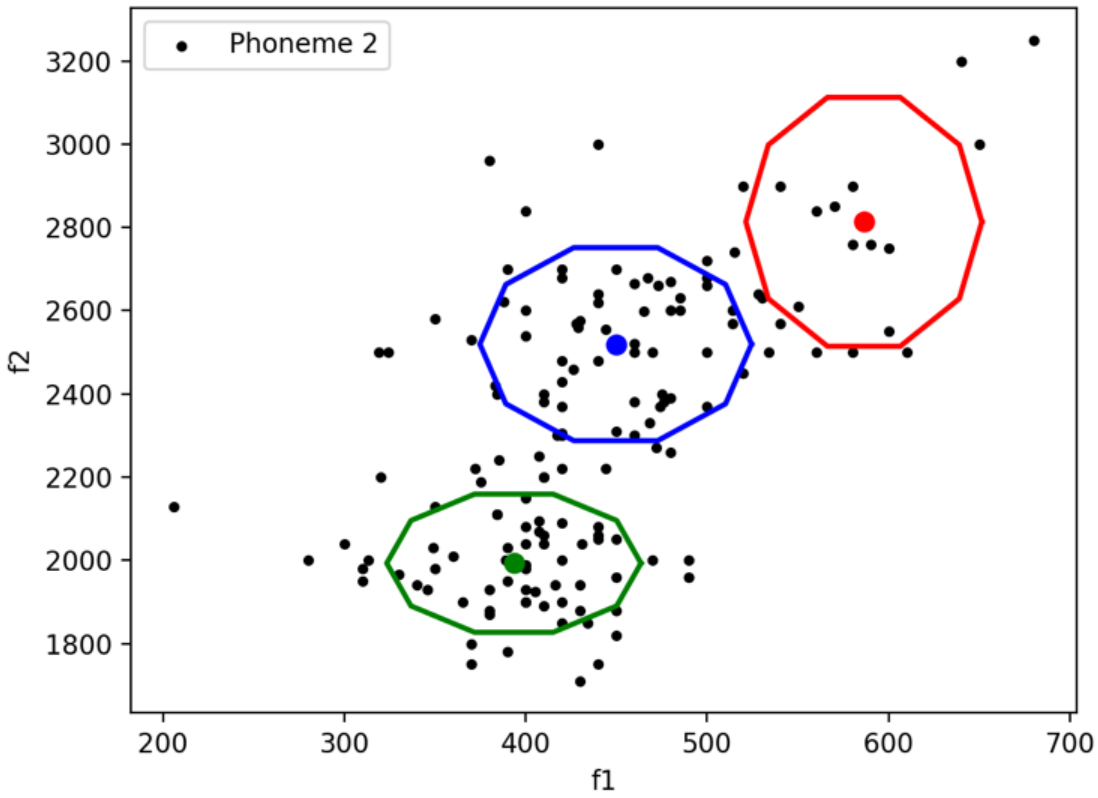


Fig.2.3(a) Phoneme 2, with k=3. First run

```
Implemented GMM | Mean values
[  585.70715  2810.9888  ]
[  393.41473  1992.6967  ]
[  449.26697  2518.7126  ]

Implemented GMM | Covariances
[[  2137.54041735      0.          ]
 [     0.          49416.114584 14]]
[[  2456.9528523       0.          ]
 [     0.          15189.567470 74]]
[[  2780.74228698      0.          ]
 [     0.          29809.776217 84]]

Implemented GMM | Weights
[0.09669633 0.43441173 0.46889193]
```

Fig.2.3(b) Phoneme 2, with k=3. First run

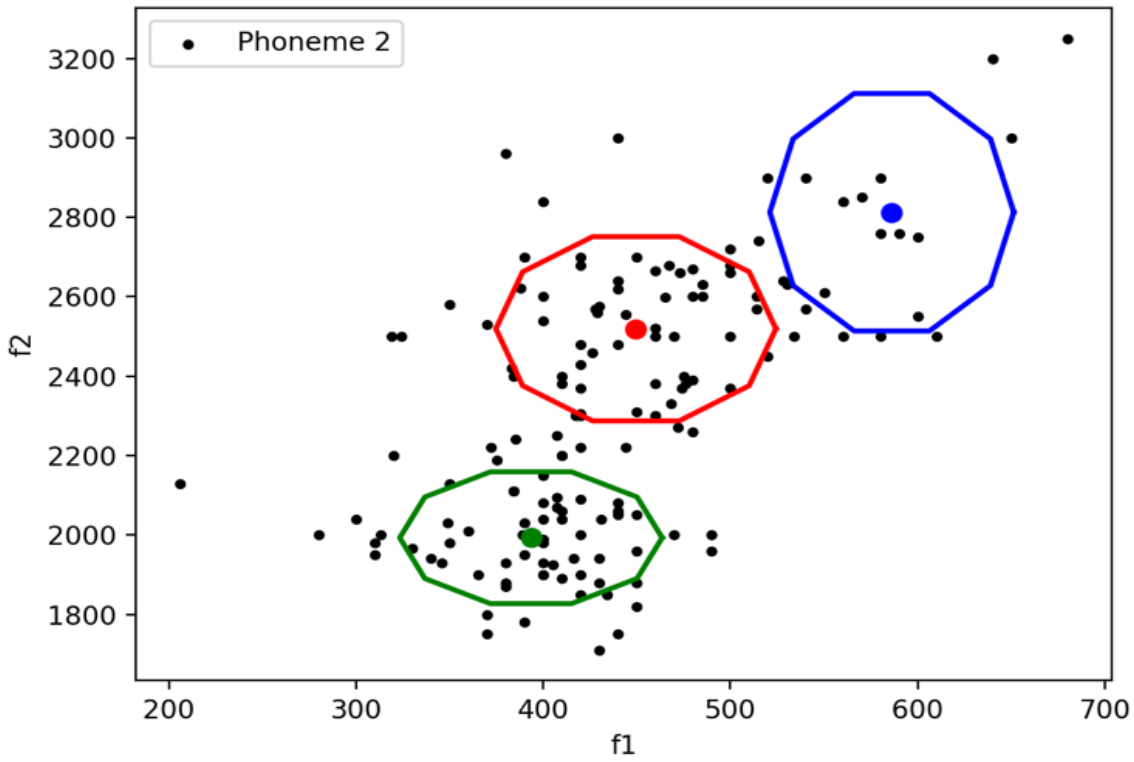


Fig.2.4(a) Phoneme 2, with k=3. Second run

```
Implemented GMM | Mean values
[ 586.38196 2813.4878 ]
[ 393.44437 1992.9967 ]
[ 449.58743 2519.4749 ]

Implemented GMM | Covariances
[[ 2116.85067262     0.          ]
 [    0.          49424.45752548]]
[[ 2455.35885663     0.          ]
 [    0.          15247.14591611]]
[[ 2803.08590069     0.          ]
 [    0.          29740.29439459]]

Implemented GMM | Weights
[0.09526804 0.43499956 0.4697324 ]
```

Fig.2.4(b) Phoneme 2, with k=3. Second run

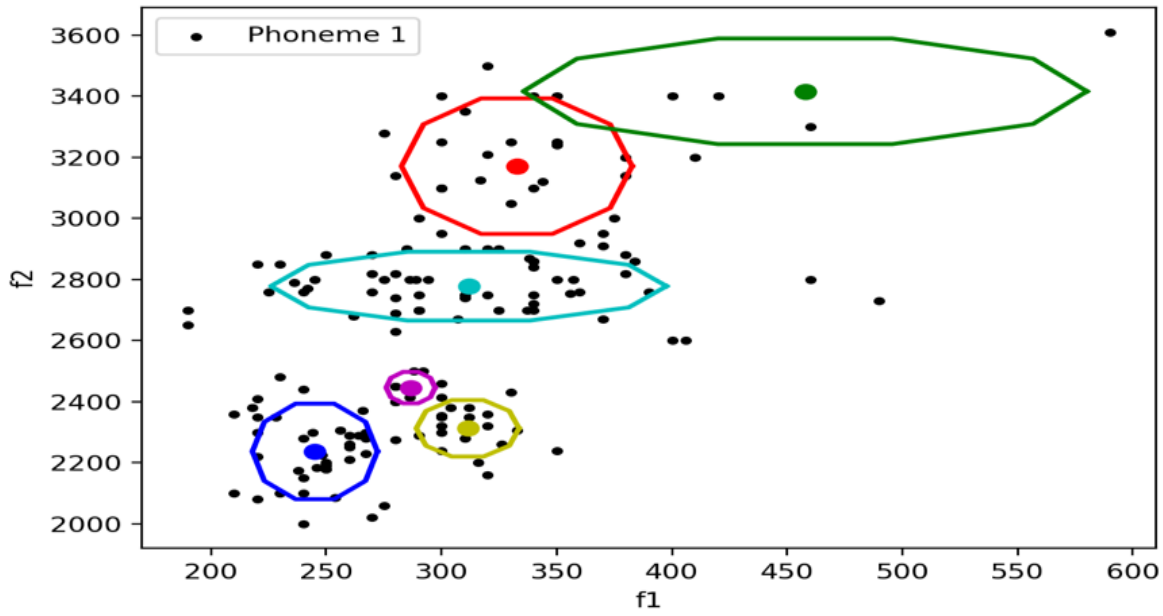


Fig.2.5(a) Phoneme 1, with k=6. First run

```
Implemented GMM | Mean values
[ 332.7129 3171.3167]
[ 457.77   3416.3794]
[ 245.11943 2237.088  ]
[ 311.85492 2778.3447 ]
[ 286.65182 2446.0862 ]
[ 311.38666 2312.607  ]

Implemented GMM | Covariances
[[ 1251.44904446     0.         ]
 [    0.          27224.79150053]]
[[ 7475.29661197     0.         ]
 [    0.          16553.74817166]]
[[  367.835604       0.         ]
 [    0.          13514.10250802]]
[[3685.5310221      0.         ]
 [   0.          7021.22494036]]
[[  56.12222788     0.         ]
 [   0.          1460.31454278]]
[[ 248.32442011     0.         ]
 [   0.          4718.64986502]]

Implemented GMM | Weights
[0.1724025  0.02608943 0.249796   0.36735596 0.0500499  0.13430622]
```

Fig.2.5(b) Phoneme 1, with k=6. First run

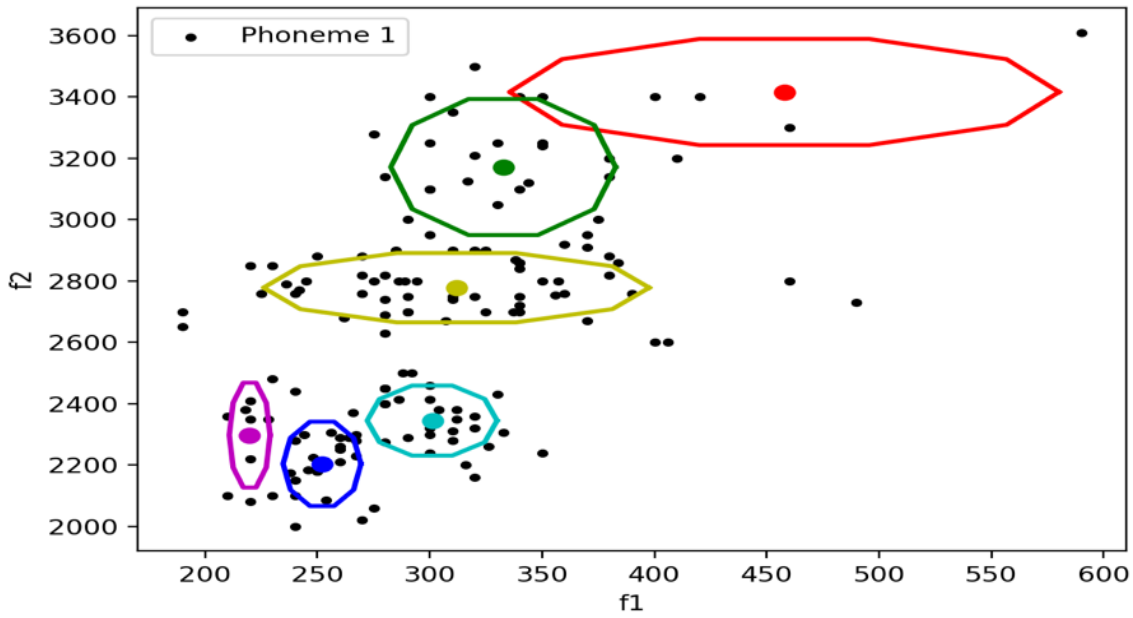


Fig.2.6(a) phoneme 1, with k=6. Second run

```
Implemented GMM | Mean values
[ 457.80908 3416.4097 ]
[ 332.71423 3171.699  ]
[ 252.07738 2203.9736 ]
[ 301.07275 2345.0056 ]
[ 219.87886 2297.635  ]
[ 311.86758 2778.3516 ]

Implemented GMM | Covariances
[[ 7474.71047062     0.         ]
 [    0.          16556.49566493]]
[[ 1251.73603144     0.         ]
 [    0.          27156.73397312]]
[[  150.10139889     0.         ]
 [    0.          10443.90507229]]
[[ 417.61045016    0.         ]
 [   0.          7209.52496494]]
[[   41.36491305     0.         ]
 [    0.          16137.45621551]]
[[3683.93618428    0.         ]
 [   0.          7056.67865784]]

Implemented GMM | Weights
[0.02607684 0.17217654 0.1642437  0.20501918 0.06485695 0.36762679]
```

Fig.2.6(b) phoneme 1, with k=6. Second run

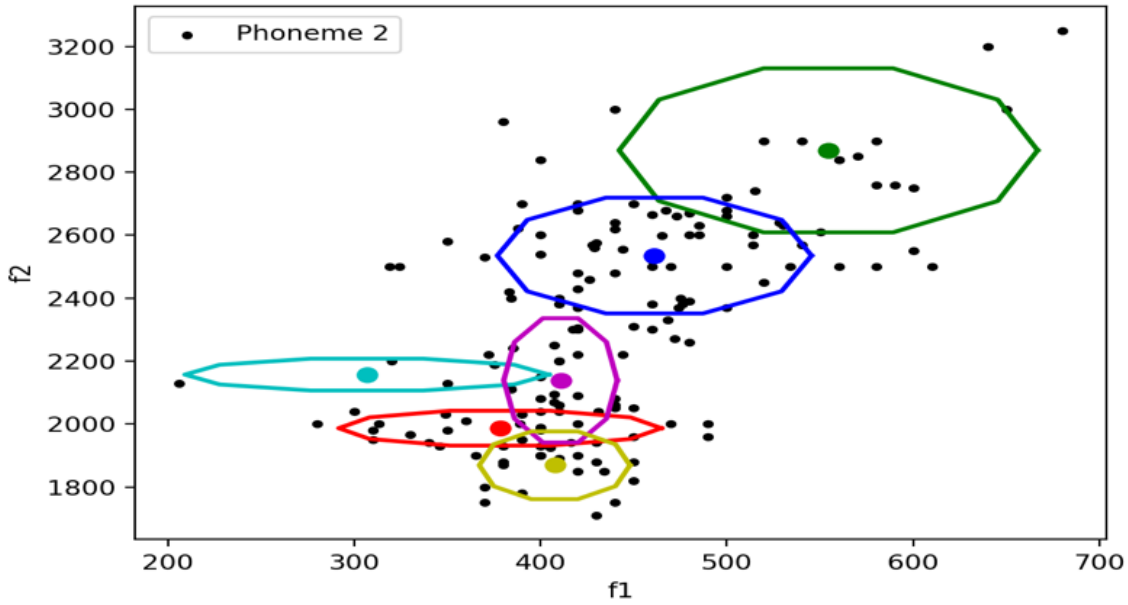


Fig.2.7(a) phoneme 2, with k=6. First run

```
Implemented GMM | Mean values
[ 378.29276 1986.7115 ]
[ 554.471  2869.6648]
[ 461.1307 2535.2246]
[ 306.80814 2156.9458 ]
[ 410.70084 2138.6494 ]
[ 407.53906 1868.8344 ]

Implemented GMM | Covariances
[[3748.35190702    0.        ]
 [   0.          1694.5424515 ]]
[[ 6306.11988538     0.        ]
 [    0.          37585.47577339]]
[[ 3555.14280642     0.        ]
 [    0.          18696.74738513]]
[[4769.32712385    0.        ]
 [   0.          1413.43650507]]
[[  464.0326501     0.        ]
 [    0.          21550.6860834]]
[[ 817.58573528    0.        ]
 [   0.          6405.56131755]]

Implemented GMM | Weights
[0.14022119 0.09782192 0.41084843 0.02258862 0.20349966 0.12502019]
```

Fig.2.7(b) phoneme 2, with k=6. First run

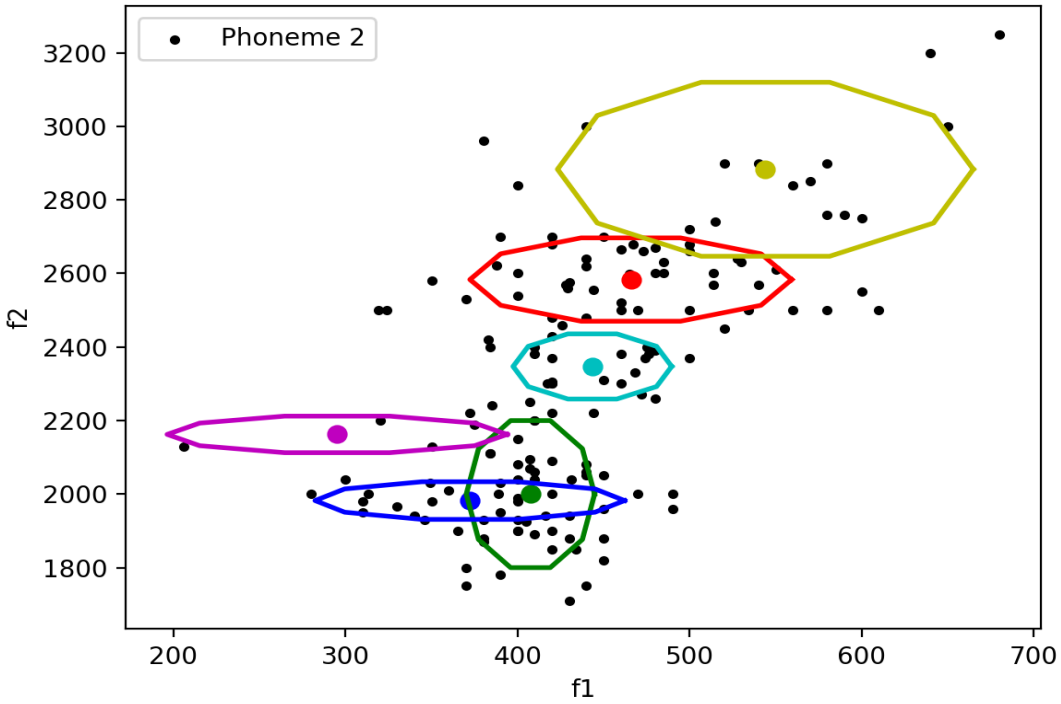


Fig.2.8(a) phoneme 2, with k=6. Second run

```
Implemented GMM | Mean values
[ 465.8205 2583.3188]
[ 407.47998 1999.8734 ]
[ 372.2424 1982.2787]
[ 443.48544 2346.9946 ]
[ 295.14053 2162.2144 ]
[ 544.062 2883.226]

Implemented GMM | Covariances
[[4370.14860287    0.        ]
 [   0.         7100.22052119]]
[[  694.97936082    0.        ]
 [    0.         22092.53079892]]
[[4043.18846517    0.        ]
 [   0.         1449.74906167]]
[[1061.35639645    0.        ]
 [   0.         4336.37315505]]
[[4879.03615053    0.        ]
 [   0.         1366.10048104]]
[[ 7298.36271842    0.        ]
 [    0.         30933.1410375 ]]

Implemented GMM | Weights
[0.3062966  0.31800327 0.11669869 0.13682943 0.01869282 0.1034792 ]
```

Fig.2.8(b) phoneme 2, with k=6. Second run

Observation:

- ⇨ Different mixtures of Gaussians are obtained when the algorithm is run multiple times.
- ⇨ Multiple runs of the model produce different clusters.

Task 3.

- ⇨ This task is to take the trained MoGs from the previous task and build a classifier to distinguish between phonemes 1 and 2 which is to be done in task_3.py

- ⇨ For this the Maximum Likelihood criterion to be used.

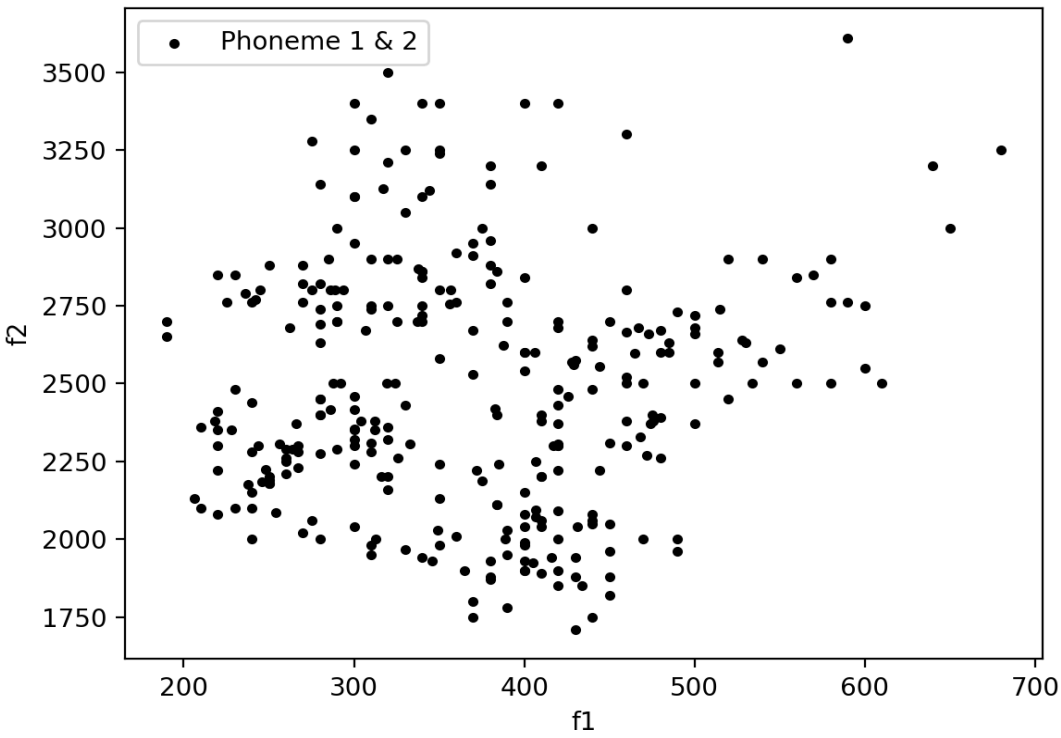


Fig.3 Phoneme 1 & 2

```
####################################
# 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

X_full = X_full.astype(np.float32)
```

```
X_phonemes_1_2 = np.zeros((np.sum(phoneme_id == 1) + np.sum(phoneme_id == 2), 2))
index = 0
phoneme_label = np.zeros((np.sum(phoneme_id == 1) + np.sum(phoneme_id == 2), 1))
groundtruth_index = 0
for i in range(len(phoneme_id)):
    if phoneme_id[i] == 1 or phoneme_id[i] == 2:
        X_phonemes_1_2[index] = X_full[i]
        index += 1
        if phoneme_id[i] == 1:
            phoneme_label[groundtruth_index] = 1
        elif phoneme_id[i] == 2:
            phoneme_label[groundtruth_index] = 2
        groundtruth_index += 1
```

⇨ **Load data from phoneme1 & phoneme 2**

```
# File path of phoneme 1 & phoneme 2
if k==3:
    phoneme1_path = 'data/GMM_params_phoneme_01_k_03.npy'
    phoneme2_path = 'data/GMM_params_phoneme_02_k_03.npy'
else:
    phoneme1_path = 'data/GMM_params_phoneme_01_k_06.npy'
    phoneme2_path = 'data/GMM_params_phoneme_02_k_06.npy'

# Load data for phoneme1

data_phoneme1 = np.load(phoneme1_path, allow_pickle=True)
data_phoneme1 = np.ndarray.tolist(data_phoneme1)
means1 = data_phoneme1['mu']
weights1 = data_phoneme1['p']
covariance1 = data_phoneme1['s']
predictions1 = get_predictions(means1, covariance1, weights1, X)

# load data for phoneme2

data_phoneme2 = np.load(phoneme2_path, allow_pickle=True)
data_phoneme2 = np.ndarray.tolist(data_phoneme2)
means2 = data_phoneme2['mu']
weights2 = data_phoneme2['p']
covariance2 = data_phoneme2['s']
predictions2 = get_predictions(means2, covariance2, weights2, X)
```

⇨ **Get the sum value for each row**

```
# get sum value from each row for predictions 1 & 2
sumValue_predictions1 = np.zeros((np.sum(phoneme_id == 1) + np.sum(phoneme_id == 2), 1))
sumValue_predictions2 = np.zeros((np.sum(phoneme_id == 1) + np.sum(phoneme_id == 2), 1))
for i in range(len(X)):
    sumValue_predictions1[i] = np.sum(predictions1[i])
    sumValue_predictions2[i] = np.sum(predictions2[i])
print(sumValue_predictions1)
```

```
# compare sum values from each array
predicted_label = np.zeros((np.sum(phoneme_id == 1) + np.sum(phoneme_id == 2), 1))
predicted_index = 0
for i in range(len(X)):
    if sumValue_predictions1[i] > sumValue_predictions2[i]:
        predicted_label[predicted_index] = 1
    elif sumValue_predictions1[i] < sumValue_predictions2[i]:
        predicted_label[predicted_index] = 2
    predicted_index += 1
```

```
# calculate accuracy
correct_samples = 0
total_samples = 0
for i in range(len(predicted_label)):
    if predicted_label[i] == phoneme_label[i]:
        correct_samples += 1
    total_samples += 1

accuracy = correct_samples/total_samples
print('Accuracy using GMMs with {} components: {:.2f}%'.format(k, accuracy))
```

After filling the task_3.py with correct code, it can now be run. Once the code run it outputs the accuracy measurements shown below. As it can be seen both GMMs have a very high accuracy measurement and both are around 95%-96% and after observing it is been noticed that using 6 Gaussians instead of 3 to model the data seems to give a slight improvement in accuracy as would be expected due to more components covering a larger area.

Accuracy using GMMs with 3 components: **95%**
Accuracy using GMMs with 6 components: **96%**

Misclassification error with 3 components: **5 %**
Misclassification error with 6 components: **4 %**

Observation : Here we can observe that accuracy using GMMs with 6 components is higher than accuracy using GMMs with 3 components.

From the results It can be observed that 6 clusters can better perform the characteristics of the phoneme.

Task 4.

- ⇨ This task is to create a grid of points which spans the area of the search space, that is the area between the minimum and maximum values of F1 and F2.

- ⇨ Then classify each of these points using the classifier trained in task 2.

```
########################################
# 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

X_full = X_full.astype(np.float32)
```

```
phonemes1_index = np.where(phoneme_id == 1)
phonemes1_index_size = phonemes1_index[0].size
phonemes_1 = np.take(X_full,phonemes1_index,axis=0).reshape(phonemes1_index_size,2)
print(len(phonemes_1))

phonemes2_index = np.where(phoneme_id == 2)
phonemes2_index_size = phonemes2_index[0].size
phonemes_2 = np.take(X_full,phonemes2_index,axis=0).reshape(phonemes2_index_size,2)
print(len(phonemes_2))
# print(phonemes_2)

X_phonemes_1_2 = np.vstack((phonemes_1,phonemes_2))
print(len(X_phonemes_1_2))
# print(X_phonemes_1_2)

# as dataset X, we will use only the samples of phoneme 1 and 2
X = X_phonemes_1_2.copy()
```

```
#get f1 values between [minf1, maxf1]
f1_values = np.linspace(min_f1,max_f1,num=N_f1)
#get f2 values between [minf2, maxf2]
f2_values = np.linspace(min_f2,max_f2,num=N_f2)
#get every possible combination of values
newf1,newf2=np.meshgrid(f2_values,f1_values)

# print(newf1.shape)
# print(newf2.shape)

# generate the 2 arrays for every possible combination between f1 and f2
custom_grid = np.array([newf2,newf1]).T
```

```
# File path of phoneme 1 & phoneme 2
if k == 3:
    phoneme1_path = 'data/GMM_params_phoneme_01_k_03.npy'
    phoneme2_path = 'data/GMM_params_phoneme_02_k_03.npy'
else:
    phoneme1_path = 'data/GMM_params_phoneme_01_k_06.npy'
    phoneme2_path = 'data/GMM_params_phoneme_02_k_06.npy'
```

```
# Load data for phoneme1
data_phoneme1 = np.load(phoneme1_path, allow_pickle=True)
data_phoneme1 = np.ndarray.tolist(data_phoneme1)
# load data for phoneme2
data_phoneme2 = np.load(phoneme2_path, allow_pickle=True)
data_phoneme2 = np.ndarray.tolist(data_phoneme2)
```

```
# predict each row
predict_row = lambda X, p: np.sum(get_predictions(p["mu"], p["s"], p["p"], X), axis=1)
predict_data_phoneme_1 = lambda X: predict_row(X, dict(data_phoneme1))
predict_data_phoneme_2 = lambda X: predict_row(X, dict(data_phoneme2))

P_1 = np.array([predict_data_phoneme_1(F_n) for F_n in custom_grid])
P_2 = np.array([predict_data_phoneme_2(F_n) for F_n in custom_grid])

# Assign 0 if False, 1 if True
M = (P_1 < P_2).astype(np.float32)
print(M)
```

- ⇨ When the code was run using the 3 Gaussian GMM a graph of the predicted values of each point is produced, this graph can be found in fig.4 The red dots show the data points that belong to phoneme1, while the green dots belong to phoneme 2.
- ⇨ The state space of the graph is then divided into two colours
- ⇨ The Purple represents the points which the classifier thinks would belong to phoneme 1based off the data points it was trained on
- ⇨ The yellow shows the area classified as phoneme 2, as can be seen the classifier has done a good job in classifying the data points, as there is a clear line running through the middle which bends appropriately to follow the data and fits the vast majority of the data well.

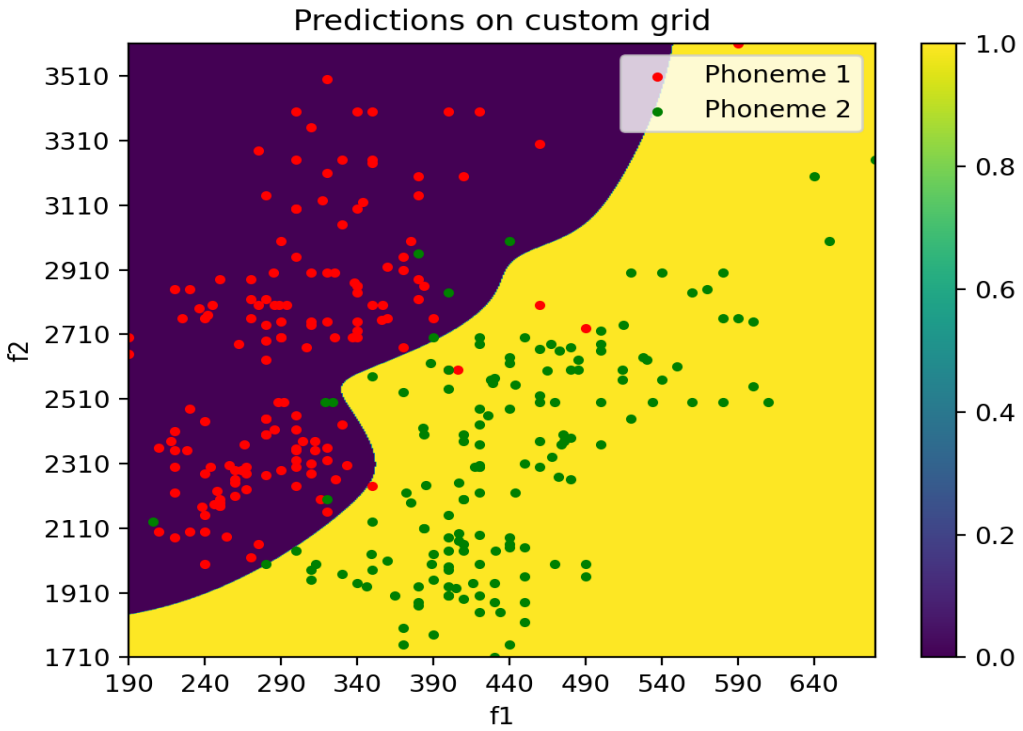


Fig 4 Graph showing the predicted phonemes of each point in the state space, 3 Gaussian model

```
152
152
304
f1 range: 190-680 | 490 points
f2 range: 1710-3610 | 1900 points
[[1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 ...
 [0. 0. 0. ... 1. 1. 1.]
 [0. 0. 0. ... 1. 1. 1.]
 [0. 0. 0. ... 1. 1. 1.]]
```

f1 range: 190-680 | 490 points
f2 range: 1710-3610 | 1900 points

⇨ This code was also run on the 6 Gaussian model for comparison purposes, the graphical output of this can be found in fig.4.1

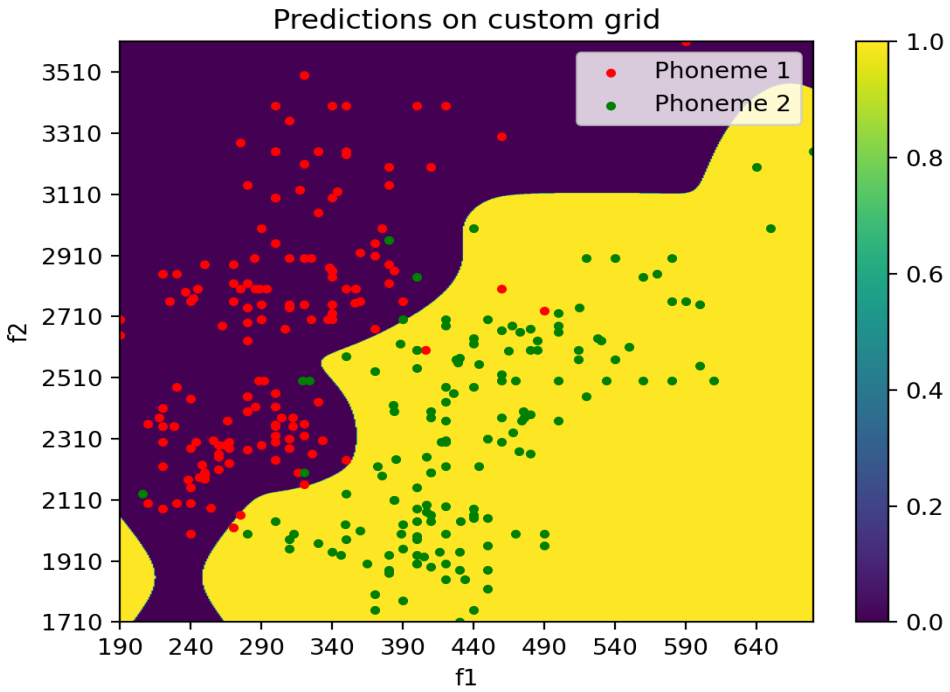


Fig.4.1 Graph showing the predicted phonemes of each point in the state space, 6 Gaussian model

```
152
152
304
f1 range: 190-680 | 490 points
f2 range: 1710-3610 | 1900 points
[[1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 [1. 1. 1. ... 1. 1. 1.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

f1 range: 190-680 | 490 points
f2 range: 1710-3610 | 1900 points

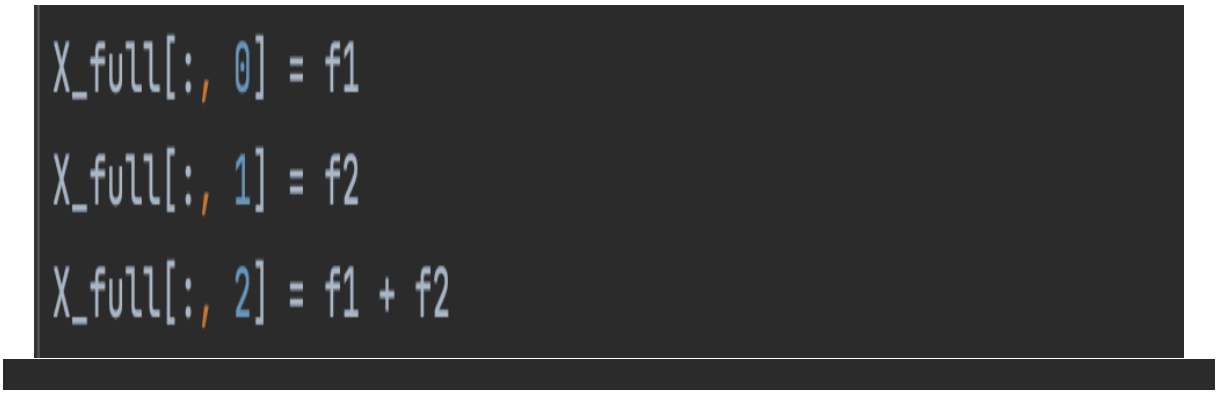
Task 5.

- ⇨ When running the model with the general gaussians form for our covariance matrix, there is a singular matrix error for the covariance matrix.

- ⇨ When I test, I saw the following error:

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

- ⇨ This basically means that inverse cant be calculated when computing the likelihood under the gaussians distribution probability density function because its determinant will be zero. This is caused by the third column f1+f2 and the fact that it is composed of a linear combination of first two columns.



```
X_phoneme = np.zeros((np.sum(phoneme_id == 1) + np.sum(phoneme_id == 2), 3))
j = 0
for i in range(len(phoneme_id)):
    if phoneme_id[i] == 1 or phoneme_id[i] == 2:
        X_phoneme[j, :] = X_full[i, :]
        j += 1
```

⇨ In the code given below, where some noise is added to the diagonal of the covariance matrix that contains the variance of each variable within the gaussian distribution.

```
########################################
# Write your code here
# Suggest ways of overcoming the singularity
########################################/
s = s + np.multiply(np.eye(D), 0.1)
```

⇨ **Singularity** in a likelihood function Is when a component collapse on a data point. It is a form of over fitting. The variance becomes zero which is the case leads to singular covariance matrix. When the variance is zero, the likelihood of the component becomes infinity and hence the model over fits. There are two ways to overcome singularity.

1) Resetting the mean and variance when singularity occurs.
2) Using MAP instead of MLE.

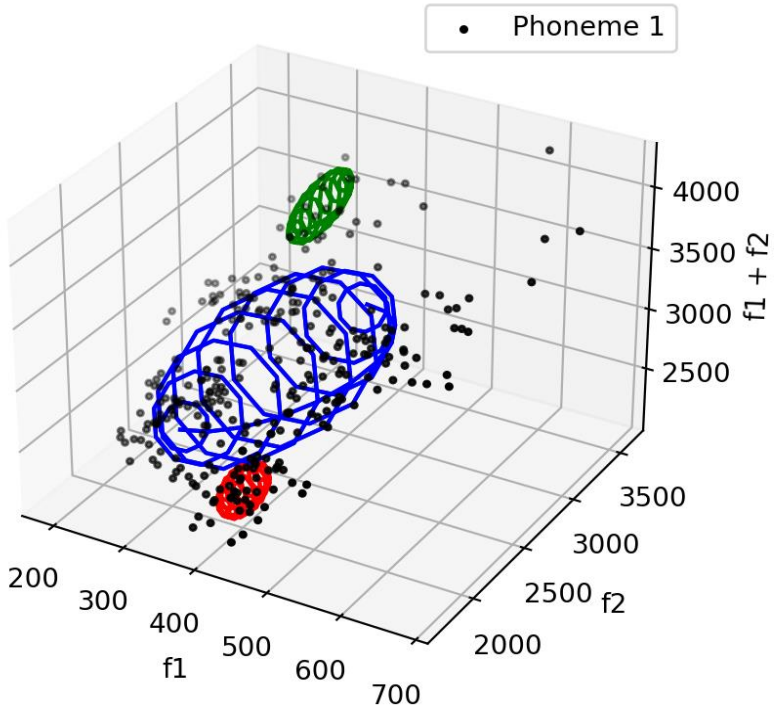


Fig.5 MoG on J= [F1, F2, F1+F2], with K=3

```
Implemented GMM | Mean values
[ 411.01769777 1942.29218521 2353.30988299]
[ 319.66429683 3259.65241777 3579.3167146 ]
[ 368.27537152 2496.87050676 2865.14587828]
Implemented GMM | Covariances
[[ 519.10649928 -143.34867581  375.45782348]
 [-143.34867581 8830.72207247 8687.07339666]
 [ 375.45782348 8687.07339666 9062.83122014]]
[[  546.6306258    496.66144192  1043.09206772]
 [  496.66144192 17937.18810232 18433.64954424]
 [ 1043.09206772 18433.64954424 19476.94161196]]
[[ 10206.6425646    6491.3308227   16697.8733873 ]
 [  6491.3308227  118520.09489963 125011.32572233]
 [ 16697.8733873  125011.32572233 141709.29910964]]
Implemented GMM | Weights
[0.08069784 0.04102607 0.87827609]
```

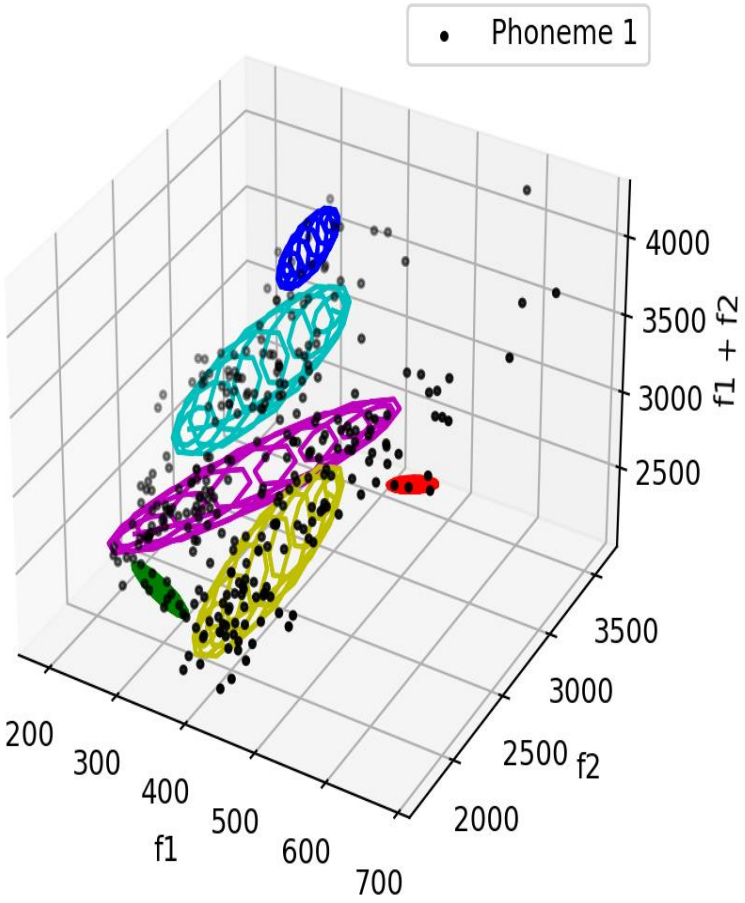


Fig.5 MoG on J= [F1, F2, F1+F2], with K=6

```
Implemented GMM | Mean values
[ 291.22351295 1995.56463854 2286.78815149]
[ 590.19195955 2722.30892728 3312.50088683]
[ 639.91002163 3271.02980864 3910.93983027]
[ 252.22967502 2212.85209707 2465.08177209]
[ 410.81636918 1952.47252504 2363.28889422]
[ 364.79736255 2551.79244214 2916.58980468]
Implemented GMM | Covariances
[[ 3159.02770786 -3883.31112803  -724.88342017]
 [-3883.31112803  4800.67305951   916.76193147]
 [ -724.88342017   916.76193147   192.4785113 ]]
[[  168.38578436 -1479.39869069 -1311.51290632]
 [-1479.39869069 18419.70520161 16939.80651092]
 [-1311.51290632 16939.80651092 15628.7936046 ]]
[[ 1073.61543597 -5162.28324619 -4089.06781022]
 [-5162.28324619 47686.26972768 42523.58648149]
 [-4089.06781022 42523.58648149 38434.91867127]]
[[ 116.12446551  610.28972885  726.11419436]
 [ 610.28972885 3833.02724624 4443.01697509]
 [ 726.11419436 4443.01697509 5169.43116946]]
[[ 517.24009055 -120.46851965  396.5715709 ]
 [-120.46851965 9723.47710075 9602.8085811 ]
 [ 396.5715709  9602.8085811  9999.58015199]]
[[  7850.86236651  -1484.80721708   6365.95514943]
 [ -1484.80721708 130629.29297024 129144.38575316]
 [  6365.95514943 129144.38575316 135510.44090259]]
Implemented GMM | Weights
[0.01803694 0.01846299 0.01253672 0.04881146 0.09005134 0.81210056]
```