ECS708 Machine Learning Assignment 2: Clustering and MoG

Animesh Devendra Chourey

210765551

1 Introduction

The data we will be working on is *Peterson and Barney's dataset.* Factors such as individual's past experience and complexity, variations can be seen in speech production.

The dataset contains a fundamental frequency *F0* along with the first three formant frequencies *F1, F2, F3*, by taking the samples from several speakers. These frequencies determine the phonetic quality of a vowel. The data is present in the *PB_data.npy* file. The file contains fundamental frequencies in following four vectors *F0, F1, F2, F3* for each phoneme and one more vector *phoneme__id"* representing a number for the id of the phoneme. The order is as follow:

phoneme_id	F0	F1	F2	F3
1	XXX	XXX	XXX	XXX
...	XXX	XXX	XXX	XXX
10	XXX	XXX	XXX	XXX
...	XXX	XXX	XXX	XXX
N	XXX	XXX	XXX	XXX

2 MoG Modelling using the EM Algorithm

A Mixture of Gaussian (Mog) is a function that comprises of several Gaussians, each identified by k $\in$ 1,....,K. Here K represents the number of clusters of our dataset. Each Gaussian *k* from the mixture comprises of some parameters:

- μ , which is the mean that defines the centre.
- Σ , which represents the co-variance.

2.1 Task 1

Store the f1 and f2 frequencies in X_full variable.

```
X_full[:,0] = f1
X_full[:,1] = f2
```

Creating an array that only contains only the samples that belong to the chosen phoneme i.e. p_id = 1

Here firstly an empty array is created that will ultimately store the indexes of phoneme_id which are equal to p_id i.e. 1. The following code iterates over every phoneme value, then it checks whether the element in phoneme_id is equal to 1 or not. If the condition proves to be true, that index value is stored in the element_index.

```
element_index = []
for i in range(len(phoneme_id)): #extract the indexes that have the phoneme_id as p_id
    if phoneme_id[i] == p_id:
    element_index.append(i)
```

Afterwards, we iterate over the X_phoneme_1 numpy array. Then we extract the index values stored in the element_index and store it in *index* variable while iterating. Finally, at the *index* from the X_full array we extract the row and update it to the X_phoneme_1[i] row.

```
for i in range(X_phoneme_1.shape[0]):
   index = element_index[i] #Store the index

   #Extract the row from X_full from the following "index" and store it to X_phoneme
   X_phoneme_1[i] = X_full[index]
```

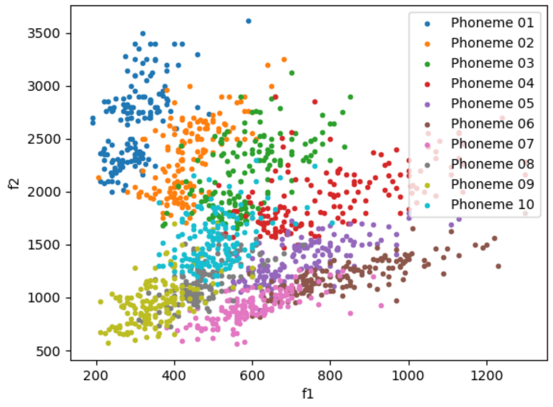


Fig 1. Cluster of every phoneme class.

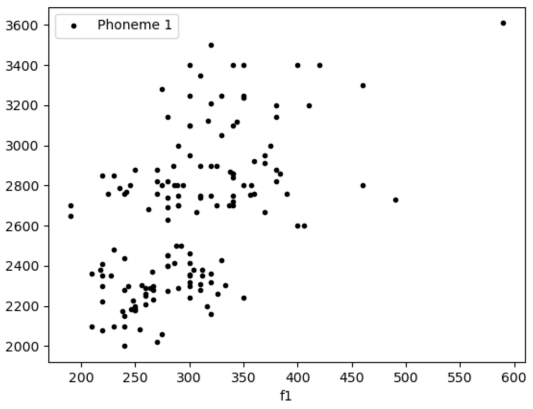


Fig 2. Cluster of phoneme 1.

2.2 Task 2

2.2.1 Look at the task_2.py code and understand what it is calculating. Pay particular attention to the initialisation of the means and covariances

This task is used to construct model for phoneme 1 and phoneme 2 for k=3 and k=6 gaussian clusters. The MOG model is trained using Expectation Maximization algorithm.

The algorithm randomly initializes the mean(mu) from the dataset. It first randomly selects the number of indices which are equal to the number of gaussian clusters and for those indices selects the data points which are present at those indexes. The shape of the mean is (k,D) while the shape of covariance matrix (s) is of shape (k,D,D)) which are initialized by first taking the covariance of the transpose of the X, and then the calculated values are divided by k. The values of p are assigned randomly and it is of shape (k,). Here k= Number of gaussian clusters and D = Number of dimensions. Also Z which stores the predictions of gaussian for each data sample is of shape (N,k), where N = number of samples.

The Expectation Maximization algorithm is iterated for n_iter times.In the E-step, predictions are calculated using the get_prediction function which takes mean, covariance, weights and the dataset. The predictions are then normalized so that the probabilities are within the range.

The maximization step runs for each gaussian separately. To make the optimization easier logarithmic operations are used to simplify the exponential terms. Here it can be noted that only the diagonal covariances are calculated. These diagonal values correspond to the variances of each dimensions.

2.2.2 Generate a dataset X phoneme 1 that contains only the F1 and F2 for the first phoneme.

```
X_full[:,0] = f1
X_full[:,1] = f2
```

Create an array named ”X_phoneme”, containing only samples that belong to the chosen phoneme. Firstly we will run the following code to store for p_id = 1 and then for p_id = 2.

```
element_index = []
for i in range(len(phoneme_id)): #extract the indexes that have the phoneme_id as p_id
    if phoneme_id[i] == p_id:
        element_index.append(i)

for i in range(X_phoneme.shape[0]):
    index = element_index[i]    #Store the index

    #Extract the row from X_full from the following "index" and store it to X_phoneme
    X_phoneme[i] = X_full[index]
```

2.2.3 Run task 2.py on the dataset using K=3 Gaussians (run the code a number of times and note the differences.)

- **Several Plots for first phoneme for K = 3 Gaussian Clusters**

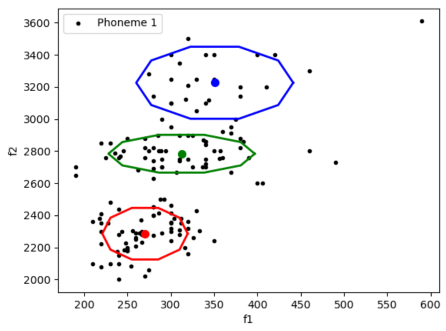


Fig 3. Phoneme = 1 K=3 First Result

```
Implemented GMM | Mean values          Implemented GMM | Covariances
 [ 270.39520145 2285.46535178]           [[ 1213.73842181     0.        ]
 [ 312.59122018 2783.89778408]            [    0.         14278.42059397]]
 [ 350.8445869  3226.33853147]           [[3562.59799481    0.        ]
                                          [   0.         7657.82825433]]
                                         [[ 4102.87114358     0.        ]
                                          [    0.         27829.75450111]]

Implemented GMM | Weights
 [0.43514434 0.3809948  0.18386085]
```

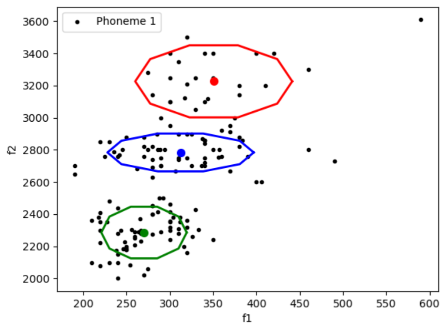


Fig 4. Phoneme = 1 K=3 Second Result

```
Implemented GMM | Mean values          Implemented GMM | Covariances
 [ 350.85055712 3226.51939665]          [[ 4103.7820642      0.          ]
 [ 270.39515719 2285.46495019]           [    0.          27783.21093101]]
 [ 312.59875992 2783.93092605]          [[ 1213.73989577      0.          ]
                                         [    0.          14278.36930212]]
                                        [[3562.47929923    0.          ]
                                         [   0.          7662.33477041]]

Implemented GMM | Weights
 [0.1837566  0.43514347 0.38109992]
```

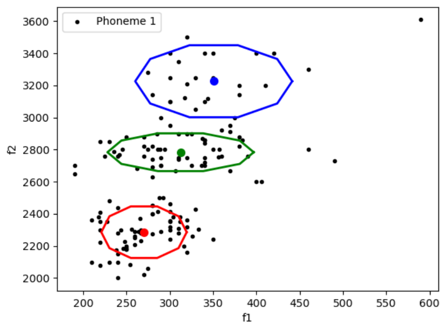


Fig 5. Phoneme = 1 K=3 Third Result

```
Implemented GMM | Mean values          Implemented GMM | Covariances
 [ 270.39520125 2285.46535    ]         [[ 1213.73842838      0.          ]
 [ 312.591254   2783.89793231]           [    0.          14278.42036678]]
 [ 350.84461358 3226.33934195]          [[3562.59746931    0.          ]
                                         [   0.          7657.8483828 ]]
                                        [[ 4102.87521422      0.          ]
                                         [    0.          27829.5458115 ]]

Implemented GMM | Weights
 [0.43514434 0.38099527 0.18386039]
```

While clustering the data for phoneme 1 and 3 Gaussian clusters, the mean, co-variance and the weights are almost similar for the first and the third run whereas second run produces different results.

2.2.4 Run task_2.py on the dataset using K=6

- **Several Plots for first phoneme for K = 6 Gaussian Clusters**

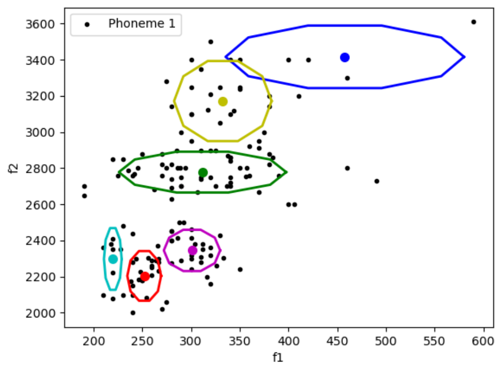


Fig 7. Phoneme = 1 K=6 First Result

```
Implemented GMM | Mean values       Implemented GMM | Covariances
 [ 252.07737462 2203.97353865]       [[  150.10139068      0.          ]
 [ 311.86755383 2778.35140843]        [     0.          10443.90423539]]
 [ 457.80888628 3416.40953819]       [[3683.93795659    0.          ]
 [ 219.87885772 2297.63504372]        [   0.          7056.67007546]]
 [ 301.07276574 2345.00550394]       [[ 7474.71389408      0.          ]
 [ 332.71420904 3171.69849197]        [     0.          16556.48906651]]
                                     [[   41.3649166      0.          ]
                                      [     0.          16137.45484333]]
                                     [[ 417.61053901    0.          ]
                                      [   0.          7209.52463114]]
                                     [[ 1251.73486573      0.          ]
                                      [     0.          27156.84888495]]
Implemented GMM | Weights
 [0.16424368 0.36762647 0.0260769  0.06485695 0.2050192  0.17217679]
```

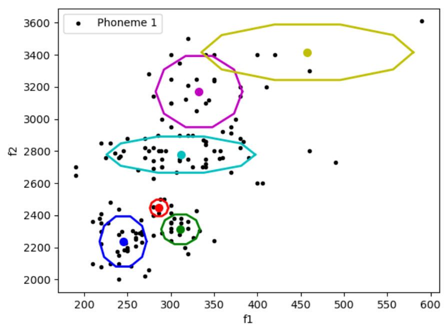


Fig 8. Phoneme = 1 K=6 Second Result

```
Implemented GMM | Mean values        Implemented GMM | Covariances
 [ 286.65181602 2446.08617191]        [[  56.12214319    0.        ]
 [ 311.38662863 2312.60709158]         [   0.         1460.3142682 ]]
 [ 245.11942366 2237.08777151]        [[ 248.32472119    0.        ]
 [ 311.85580995 2778.34900541]         [   0.         4718.65323706]]
 [ 332.71332645 3171.34422233]        [[  367.83549415      0.         ]
 [ 457.77654228 3416.38520602]         [     0.          13514.1015549 ]]
                                      [[3685.44666988    0.        ]
                                       [   0.         7021.64194545]]
                                      [[ 1251.49185341      0.         ]
                                       [     0.          27219.36250558]]
                                      [[ 7475.20073992      0.         ]
                                       [     0.          16553.98050515]]
Implemented GMM | Weights
 [0.05004983 0.1343063  0.24979596 0.36737109 0.1723895  0.02608732]
```

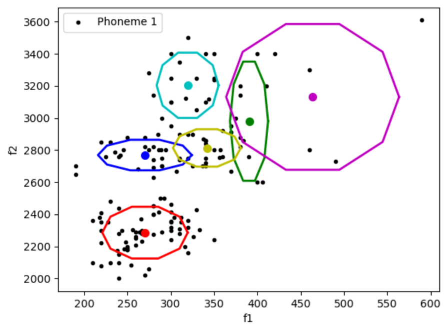


Fig 10. Phoneme = 1 K=6 Third Result

```
Implemented GMM | Mean values        Implemented GMM | Covariances
 [ 270.38726036 2285.81037569]        [[ 1213.41018233      0.         ]
 [ 390.44517421 2980.51018248]         [     0.          14366.25163233]]
 [ 270.26307283 2769.36670073]        [[  240.90566185      0.        ]
 [ 319.58656689 3204.10288877]         [    0.          76241.92423438]]
 [ 463.84625725 3131.87853795]        [[1465.00145521    0.        ]
 [ 341.81767247 2813.84057429]         [   0.         5012.48260592]]
                                      [[  639.55247075      0.        ]
                                       [    0.          22850.99184781]]
                                      [[  4978.86736126      0.        ]
                                       [     0.         114102.76315573]]
                                      [[ 766.33960697    0.        ]
                                       [   0.         7521.75458431]]
Implemented GMM | Weights
 [0.43562141 0.05246272 0.19298463 0.13304624 0.03915209 0.14673291]
```

When it comes to k=6 gaussian clusters, the first and the second run produces almost similar gaussian cluster shape whereas the third run produces completely different clusters. It should also be noted that the clusters are overlapping on some of the data points which can have a negative impact on the classification.

2.2.5 Repeat steps 2-4 for the second phoneme

- **Several Plots for second phoneme for K = 3 Gaussian Clusters**

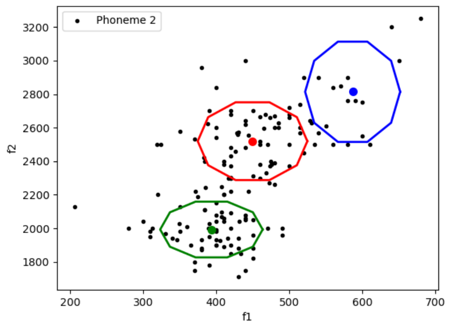


Fig 11. Phoneme = 2 K=3 First Result

```
Implemented GMM | Mean values       Implemented GMM | Covariances
 [ 449.67905188 2519.69357317]        [[ 2809.54418973     0.        ]
 [ 393.45317161 1993.08549925]         [    0.         29719.62613607]]
 [ 586.56941602 2814.20399729]        [[ 2454.89392963     0.        ]
                                       [    0.         15264.27016835]]
                                      [[ 2111.35002494     0.        ]
                                       [    0.         49424.39475286]]
Implemented GMM | Weights
 [0.46996053 0.43517302 0.09486645]
```

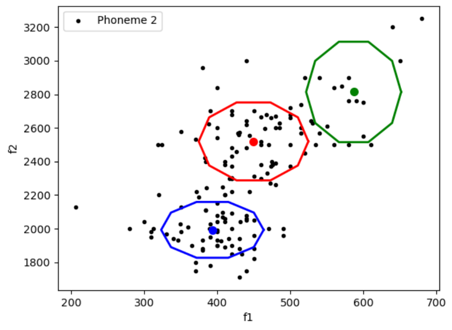


Fig 12. Phoneme = 2 K=3 Second Result

```
Implemented GMM | Mean values        Implemented GMM | Covariances
 [ 449.67125978 2519.6749721 ]        [[ 2808.99370813     0.         ]
 [ 586.55356157 2814.14303207]         [    0.         29721.39331206]]
 [ 393.45241802 1993.07790208]        [[ 2111.81123095     0.         ]
                                       [    0.         49424.44181118]]
                                      [[ 2454.9335761      0.         ]
                                       [    0.         15262.80466639]]
Implemented GMM | Weights
 [0.4699413 0.0949005 0.4351582]
```

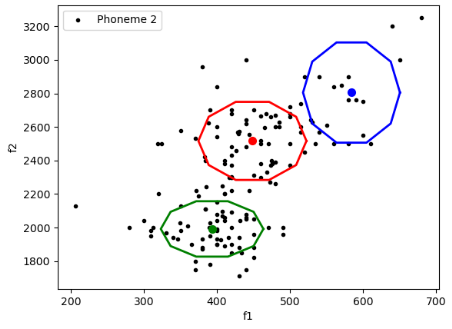


Fig 13. Phoneme = 2 K=3 Third Result

```
Implemented GMM | Mean values        Implemented GMM | Covariances
 [ 448.51172057 2516.94445596]        [[ 2729.62711831     0.         ]
 [ 393.35389928 1992.07334488]         [    0.         29946.57613098]]
 [ 583.97374131 2805.11559075]        [[ 2460.4207741      0.         ]
                                       [    0.         15071.27987986]]
                                      [[ 2197.07487583     0.         ]
                                       [    0.         49342.36855632]]
Implemented GMM | Weights
 [0.4665772  0.43318094 0.10024186]
```

For the phoneme 2 having 3 gaussian clusters, every run gives us almost similar grouping for the data.

- **Several Plots for second phoneme for K = 6 Gaussian Clusters**

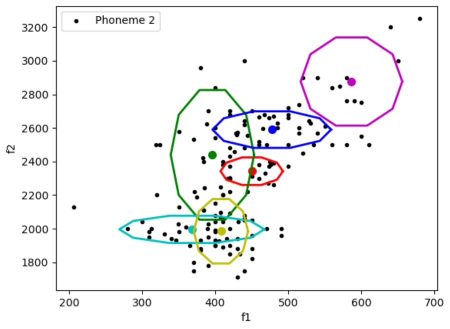


Fig 13. Phoneme = 2 K=6 First Result

```
Implemented GMM | Mean values        Implemented GMM | Covariances
 [ 449.90195277 2342.58382544]        [[ 909.73618024    0.        ]
 [ 396.18962916 2439.62844192]         [   0.         3733.41577569]]
 [ 477.34592931 2590.17011951]        [[ 1624.91659135     0.        ]
 [ 368.05520347 1996.14533231]         [    0.         82382.29499113]]
 [ 586.35279839 2876.4433529 ]        [[3287.12335654    0.        ]
 [ 407.84443042 1984.89962926]         [   0.         6545.13107675]]
                                      [[4914.22887706    0.        ]
                                       [   0.         3635.17054641]]
                                      [[ 2411.29852127     0.        ]
                                       [    0.         38080.85765708]]
                                      [[  690.2370901      0.        ]
                                       [    0.         20347.03698804]]
Implemented GMM | Weights
 [0.11291426 0.13507771 0.26007961 0.13752653 0.08164603 0.27275586]
```

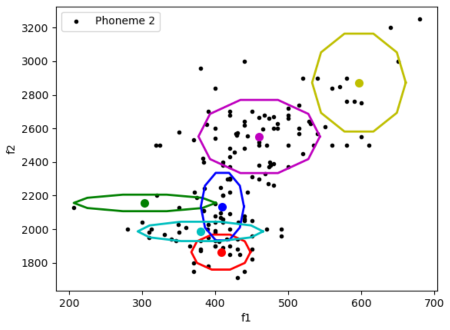


Fig 13. Phoneme = 2 K=6 Second Result

```
Implemented GMM | Mean values        Implemented GMM | Covariances
 [ 407.79921645 1863.72754958]        [[ 817.47762861    0.         ]
 [ 303.68920995 2156.57308067]         [   0.         5993.91007063]]
 [ 409.86271049 2134.58822941]        [[4715.75383984    0.         ]
 [ 379.68395135 1986.97790225]         [   0.         1337.8784866 ]]
 [ 460.14449394 2552.06947743]        [[  434.7895818     0.         ]
 [ 596.53941128 2873.01926476]         [     0.        22308.3869001]]
                                      [[3674.52183543    0.         ]
                                       [   0.         1833.90826056]]
                                      [[ 3449.31693598     0.        ]
                                       [    0.         26205.91968124]]
                                      [[ 2051.08715965     0.        ]
                                       [    0.         46937.45382404]]
Implemented GMM | Weights
 [0.11780067 0.02156947 0.19943646 0.14670946 0.44537444 0.0691095 ]
```

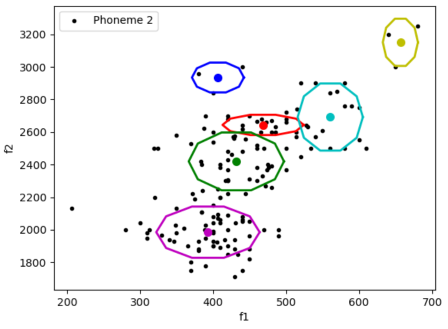


Fig 13. Phoneme = 2 K=6 Third Result

```
Implemented GMM | Mean values        Implemented GMM | Covariances
 [ 468.6675288  2643.67978471]        [[1546.00019128    0.         ]
 [ 432.02954667 2419.70333296]         [   0.         2139.0001301 ]]
 [ 406.7817423  2934.69131209]        [[ 2128.58934589     0.        ]
 [ 560.68769801 2691.9701017 ]         [    0.         17412.00093751]]
 [ 393.12255818 1985.16660325]        [[ 632.92693293    0.         ]
 [ 656.7025106  3150.71685427]         [   0.         4625.40315678]]
                                      [[  996.94366768     0.        ]
                                       [    0.         23465.97303619]]
                                      [[ 2507.7157601      0.        ]
                                       [    0.         13658.73175665]]
                                      [[  290.08983417     0.        ]
                                       [    0.         11615.8310993 ]]
Implemented GMM | Weights
 [0.12542671 0.30397986 0.01937987 0.10940147 0.42217507 0.01963703]
```

Similar results are being obtained as to when the clusters were formed with phoneme 1 having 6 gaussian clusters. Here in phoneme 2, after every run gaussian clusters are being formed of different shapes and each cluster having some different data points.

2.3 Task 3

In task 3 we need to predict the phoneme id for each of the data point with the help of models that we already trained in the previous task.

Storing f1 and f2 in the first and second column of X_full

```
X_full[:,0] = f1
X_full[:,1] = f2
```

In order to store the data corresponding tho both phoneme 1 and phoneme 2 the following steps were taken:

- Create an array X_phoneme_1 and X_phoneme_2 filled with zeros having rows equal to the sum of phoneme_id = 1 and phoneme_id = 2 respectively. Both of the arrays have 2 columns.

```
X_phoneme_1 = np.zeros((np.sum(phoneme_id==1), 1))
X_phoneme_2 = np.zeros((np.sum(phoneme_id==2), 2))
```

- Extracting the data corresponding to the phoneme_id =1. Firstly for that, an empty array is created that will ultimately store the indexes of phoneme_id which are equal to 1. The following code iterates over every phoneme value, then it checks whether the element in phoneme_id is equal to 1 or not. If the condition proves to be true, that index value is stored in the element_index_phoneme1. Afterwards, we iterate over the X_phoneme_1 numpy array. We extract the index values stored in the element_index_phoneme1 and store it in *index* variable while iterating. Finally, at the *index* from the X_full array we extract the row and update it to the X_phoneme_1[i] row.

```
#Getting the data for phoneme 1
element_index_phoneme1 = []
for i in range(len(phoneme_id)): #extract the indexes that have the phoneme_id=1
    if phoneme_id[i] == 1:
      element_index_phoneme1.append(i)

for i in range(X_phoneme_1.shape[0]):
    index = element_index_phoneme1[i]    #Store the index
    X_phoneme_1[i] = X_full[index]
```

- Similarly doing the same process as before but this time for phoneme_id=2.

```
#Getting the data for phoneme 2
element_index_phoneme2 = []
for i in range(len(phoneme_id)): #extract the indexes that have the phoneme_id=2
   if phoneme_id[i] == 2:
      element_index_phoneme2.append(i)

for i in range(X_phoneme_2.shape[0]):
   index = element_index_phoneme2[i]    #Store the index
   X_phoneme_2[i] = X_full[index]
```

- Finally, stacking the X_phoneme_1 and X_phoneme_2 to create a new array X_phonemes_1_2 having only the data of phoneme 1 and phoneme 2.

```
#Stacking X_phoneme_1 and X_phoneme_2
X_phonemes_1_2 = np.vstack((X_phoneme_1, X_phoneme_2))
```

Now we need to load the files created at the end of the task 2. Every model to be loaded comprises of the following values which we need to store-

1. mu(μ) that is the mean of the every gaussian cluster for f1 and f2.
2. s(Σ) which is the covariance matrix for every gaussian.
3. p which is the weight of every k-th gaussian component.

At the end of task-2, 4 files were created which will be loaded. The two files are for phoneme 1 having gaussian clusters k=3 and k=6, while the remaining two are for phoneme 2 having similar gaussian clusters k=3 and k=6. The data that is loaded is of type dict(dictionary). For every key the corresponding values are stored.

- The first file that is loaded contains data regarding phoneme 1 having 3 gaussian clusters.

```
# File that contains the phoneme 1 and k=3
# Loading data from .npy file
data2 = np.load('data/GMM_params_phoneme_01_k_03.npy', allow_pickle=True)
data2 = np.ndarray.tolist(data2) #data is stored in the from of dictionary
#Extracting the values from the corresponding keys
mu_p1_k3 = data2['mu']
s_p1_k3 = data2['s']
p_p1_k3 = data2['p']
```

- The second file that is loaded contains data regarding phoneme 1 having 6 gaussian clusters.

```
# File that contains the phoneme 1 and k=6
# Loading data from .npy file
data3 = np.load('data/GMM_params_phoneme_01_k_06.npy', allow_pickle=True)
data3 = np.ndarray.tolist(data3) #data is stored in the from of dictionary
#Extracting the values from the corresponding keys
mu_p1_k6 = data3['mu']
s_p1_k6 = data3['s']
p_p1_k6 = data3['p']
```

- The third file that is loaded contains data regarding phoneme 2 having 3 gaussian clusters.

```
# File that contains the phoneme 2 and k=3
# Loading data from .npy file
data4 = np.load('data/GMM_params_phoneme_02_k_03.npy', allow_pickle=True)
data4 = np.ndarray.tolist(data4) #data is stored in the from of dictionary
#Extracting the values from the corresponding keys
mu_p2_k3 = data4['mu']
s_p2_k3 = data4['s']
p_p2_k3 = data4['p']
```

- The last file that is loaded contains data regarding phoneme 2 having 6 gaussian clusters.

```
# File that contains the phoneme 2 and k=6
# Loading data from .npy file
data5 = np.load('data/GMM_params_phoneme_02_k_06.npy', allow_pickle=True)
data5 = np.ndarray.tolist(data5) #data is stored in the from of dictionary
#Extracting the values from the corresponding keys
mu_p2_k6 = data5['mu']
s_p2_k6 = data5['s']
p_p2_k6 = data5['p']
```

Now after importing the data, we need to calculate the predictions which tells us the possibility of each data point belonging to the given phoneme. The predictions are made as probabilities in the form of array, with each row corresponding to each data point. These predictions are calculated with the help of *get_predictions* function. The *get_predictions* function takes mu(μ), covariance(Σ), p and the whole data as an input to return the probabilities. For phoneme 1 and phoneme 2 having having 3 gaussians, *get_predictions* function will have the shape of (304,3), as 304 rows correspond to the 304 data points while its 3 columns correspond to the probability of each data point belonging to 3 gaussians. Similarly, the *get_predictions* function will do the same for phoneme 1 and phoneme 2 having 6 gaussian clusters, the only difference will be that the array returned will be of shape (304,6) as there are 6 gaussians clusters created here.

```
# Get the predictions in the form of probabilities based on the gaussians
created for each phoneme

prediction_p1_k3 = get_predictions(mu_p1_k3, s_p1_k3, p_p1_k3, X_phonemes_1_2)
prediction_p1_k6 = get_predictions(mu_p1_k6, s_p1_k6, p_p1_k6, X_phonemes_1_2)
prediction_p2_k3 = get_predictions(mu_p2_k3, s_p2_k3, p_p2_k3, X_phonemes_1_2)
prediction_p2_k6 = get_predictions(mu_p2_k6, s_p2_k6, p_p2_k6, X_phonemes_1_2)
```

Now the probabilities for each data point is summed. This gives us the probability that each data point belongs within the gaussian clusters created for the particular phoneme.

```
# Calculate the sum of the probabilities of the gaussians belonging to
corresponding phoneme

sum_p1_k3 = np.sum(prediction_p1_k3, axis=1)
sum_p1_k6 = np.sum(prediction_p1_k6, axis=1)
sum_p2_k3 = np.sum(prediction_p2_k3, axis=1)
sum_p2_k6 = np.sum(prediction_p2_k6, axis=1)
```

After calculating the overall probability, we can now predict which phoneme does a data point belong to.

- For the gaussian clusters of 3, both the summed up probability for phoneme 1 and phoneme 2 is compared. Whichever phoneme will have higher probability, the data point will belong to that phoneme. The predicted phoneme value for each data point is stored in prediction_k3.

  ```
  # Calculate whether the data point is of phoneme 1 or phoneme 2
   in the gaussian set of 3

  prediction_k3 = np.zeros(len(X_phonemes_1_2))
  for i in range(len(prediction_k3)):
      if sum_p1_k3[i] > sum_p2_k3[i]:
          prediction_k3[i] = 1
      else:
          prediction_k3[i] = 2
  ```

- Similarly as before, now for the gaussian clusters of 6, both the summed up probability for phoneme 1 and phoneme 2 is compared. Whichever phoneme will have higher probability, the data point will belong to that phoneme. The predicted phoneme value for each data point is stored in prediction_k6.

```
# Calculate whether the data point is of phoneme 1 or phoneme 2
  in the gaussian set of 6

prediction_k6 = np.zeros(len(X_phonemes_1_2))
for i in range(len(prediction_k6)):
    if sum_p1_k6[i] > sum_p2_k6[i]:
        prediction_k6[i] = 1
    else:
        prediction_k6[i] = 2
```

An array of truth values is created which will tell us the actual phonemes that a given data point belongs to. This array is created to compare the ground truth values with the predicted values to estimate the proper correct values classified by our algorithm. As we know that in X_phonemes_1_2 we have stacked the X_phoneme_1 and X_phoneme_2 we can store the truth values of the following shape.

```
# Storing the ground truth phoneme for each data point

ground_truth = np.zeros(len(X_phonemes_1_2))

# From range(0,152) value 1 will be stored
for i in range(len(X_phoneme_1)):
    ground_truth[i] = 1

# From range(152,304) value 2 will be stored
for i in range(len(X_phoneme_1),len(X_phonemes_1_2)):
    ground_truth[i] = 2
```

After storing the ground truth, we now need to compare it with the predicted values by the algorithm.

- Iterating after every value from *prediction_k3* which is the predicted values for 3 gaussian clusters, we compare it with ground truth values one by one. If the phonemes are predicted correctly, we increment the *total_correct_prediction_k3* by 1.

  ```
  # Calculating the correct values predicted for 3 gaussian clusters
  total_correct_prediction_k3 = 0
  for i in range(len(prediction_k3)):
    if prediction_k3[i] == ground_truth[i]:#Comparing predicted value with ground truth
       total_correct_prediction_k3 = total_correct_prediction_k3 + 1
  ```

- Iterating after every value from *prediction_k6* which is the predicted values for 6 gaussian clusters, we compare it with ground truth values one by one. If the phonemes are predicted correctly, we increment the *total_correct_prediction_k6* by 1.

```
# Calculating the correct values predicted for 6 gaussian clusters
total_correct_prediction_k6 = 0
for i in range(0, len(prediction_k6)):
  if prediction_k6[i] == ground_truth[i]: #Comparing predicted value with ground truth
     total_correct_prediction_k6 = total_correct_prediction_k6 + 1
```

All that is left for this task is to calculate the accuracy percentage for each gaussian category.

```
# Calculating the accuracy
accuracy_k3 = total_correct_prediction_k3/len(prediction_k3) * 100
accuracy_k6 = total_correct_prediction_k6/len(prediction_k6) * 100
```

To print the output the following code was used.

```
print('Accuracy using GMMs with {} components: {:.2f}%'.format(3, accuracy_k3))
print('Accuracy using GMMs with {} components: {:.2f}%'.format(6, accuracy_k6))
```

After running the file we are getting accuracy for 3 gaussian clusters having 95.07% accuracy while the 6 gaussian clusters are giving the accuracy of 95.72%. There is a slight rise in predicting correct phoneme for the data points. This can be because the there are more gaussian clusters which are able to cover more number of data points. However, it can also be noted that there is not a significant amount of increase in accurately predicting the phonemes when using this dataset. The increase in accuracy can also be noted because of the increasing number of cluster allow the model to converge at its minima which results in ultimately more accurate prediction thus giving us better accuracy.

Output-
Accuracy using GMMs with 3 components: 95.07
Accuracy using GMMs with 6 components: 95.72
Miss-Classification error:
The miss-classification error for GMM with 3 components : 4.93
The miss-classification error for GMM with 6 components : 4.28

2.4 Task 4

In this task we need to create a grid of points that spans the two datasets. Firstly, start off with storing f1 and f2 values in the first and second column of the X_full variable.

```
f1 = data['f1']
f2 = data['f2']
```

Next we need to create an array X_phonemes_1_2 which will contain the samples of both phoneme 1 and phoneme 2. To perform this, the code is written the same way it is done in task_3.py. All the data corresponding to the phoneme 1 is extracted from X_full and stored in X_phoneme_1. Similarly, the data corresponding to the phoneme 2 is extracted from X_full and stored in X_phoneme_2. In the end both the variables are stacked in X_phonemes_1_2.

```
 X_phoneme_1 = np.zeros((np.sum(phoneme_id==1), 2))
 X_phoneme_2 = np.zeros((np.sum(phoneme_id==2), 2))

 #Getting the data for phoneme 1
 element_index_phoneme1 = []
 for i in range(len(phoneme_id)): #extract the indexes that have the phoneme_id as 1
    if phoneme_id[i] == 1:
       element_index_phoneme1.append(i)

 for i in range(X_phoneme_1.shape[0]):
    index = element_index_phoneme1[i]    #Store the index
    X_phoneme_1[i] = X_full[index]


 #Getting the data for phoneme 2
 element_index_phoneme2 = []
 for i in range(len(phoneme_id)): #extract the indexes that have the phoneme_id as 2
    if phoneme_id[i] == 2:
       element_index_phoneme2.append(i)

 for i in range(X_phoneme_2.shape[0]):
    index = element_index_phoneme2[i]    #Store the index
    X_phoneme_2[i] = X_full[index]

#Stacking X_phoneme_1 and X_phoneme_2
X_phonemes_1_2 = np.vstack((X_phoneme_1, X_phoneme_2))
```

Now a custom_grid is created which contains every data point within the search space of the min_f1 and max_f1, min_f2 and max_f2. This range is within the minimum and the maximum values of f1 and f2. Also the range is stored for both f1 and f2.

```
#Initialize custom_grid array
custom_grid = np.zeros((N_f1, N_f2, 2))

#Get the range for f1 and f2
f1_range_values = range(min_f1, max_f1)
f2_range_values = range(min_f2, max_f2)
```

In the custom_grid we need to store all the points between the range. The custom_grid will have all the values of (f1,f2) pairs,each of the pair will be in between (min_f1,max_f1) on the f1 axis and in between (min_f2,max_f2) on f2 axis.

```
#Store the custom_grid with the data points between the range
#Each element in f1_range_values will be grouped with all the
elements in f2_range_values one by one

for i, v1 in enumerate(f1_range_values):
    for j, v2 in enumerate(f2_range_values):
        custom_grid[i][j] = [v1, v2]
```

Now we need to import the files saved at the end of task_2.py file. After importing the files we need the extract the values of *mu*, *s* and *p*. The process is similar to the one performed in task_3.py. Here in this task we will only be importing the files having for phoneme 1 and phoneme 2 having 3 gaussian clusters.

```
# File that contains the phoneme 1 and k=3
# Loading data from .npy file
data2 = np.load('data/GMM_params_phoneme_01_k_03.npy', allow_pickle=True)
data2 = np.ndarray.tolist(data2) #data is stored in the from of dictionary
#Extracting the values from the corresponding keys
mu_p1_k3 = data2['mu']
s_p1_k3 = data2['s']
p_p1_k3 = data2['p']


# File that contains the phoneme 2 and k=3
# Loading data from .npy file
data4 = np.load('data/GMM_params_phoneme_02_k_03.npy', allow_pickle=True)
data4 = np.ndarray.tolist(data4) #data is stored in the from of dictionary
#Extracting the values from the corresponding keys
mu_p2_k3 = data4['mu']
s_p2_k3 = data4['s']
p_p2_k3 = data4['p']
```

After importing the files successfully and extracting the data from them, we now need to make the predictions based on the probabilities.

- Firstly we will be calculating the probabilities according to the phoneme 1. An empty array is created which will store the probabilities. A loop will iterate over every row present in the array. On each iteration, get_prediction function will be called, which will take mu_p1_k3, s_p1_k3, p_p1_k3 as input. It will also take custom_grid data as input. For every iteration the data passed will be of shape (490,2). The data will be from all the arrays, all the values will be passed present at the i-th row. The function will return the probabilities which will be stored in prediction_p1_k3. Now we need to add these probabilities to calculate the maximum likelihood. Finally each of the summed up values will be appended to sum_p1_k3.

```
# Get the predictions in the form of probabilities based on the
gaussians created for each phoneme
# As well as, Calculate the sum of the probabilities of the
gaussians belonging to corresponding phoneme

#Calculating the sum for phoneme 1 for k=3
sum_p1_k3 = []
for i in range(N_f2): #for all the 1900 rows
   #All values present at the i-th row of each array of custom_grid
   prediction_p1_k3 = get_predictions(mu_p1_k3, s_p1_k3, p_p1_k3, custom_grid[:, i])
   sum_p1_k3.append(np.sum(prediction_p1_k3, axis=1)) #Summing the probabilities
sum_p1_k3 = np.array(sum_p1_k3)
```

- All the steps taken for phoneme 1 will be executed in the similar way for the phoneme 2. On each iteration, the get_function will be called, which will take mu_p2_k3, s_p2_k3, p_p2_k3 as input. It will also take custom_grid data as input. The function will return the probabilities which will be stored in prediction_p2_k3. Finally each of the summed up values will be appended to sum_p2_k3.

```
#Calculating the sum for phoneme 2 for k=3
sum_p2_k3 = []
for i in range(N_f2): #for all 1900 arrays
   #All values present at the i-th row of each array of custom_grid
   prediction_p2_k3 = get_predictions(mu_p2_k3, s_p2_k3, p_p2_k3, custom_grid[:, i])
   sum_p2_k3.append(np.sum(prediction_p2_k3, axis=1)) #Summing the probabilities
sum_p2_k3 = np.array(sum_p2_k3)
```

- Similar steps taken for phoneme 1 and phoneme 2 for k=3, will be taken for k=6 for both phoneme 1 and 2.

```
#Calculating the sum for phoneme 1 for k=6
sum_p1_k6 = []
for i in range(N_f2): #for all the 1900 rows
    #All values present at the i-th row of each array of custom_grid
    prediction_p1_k6 = get_predictions(mu_p1_k6, s_p1_k6, p_p1_k6, custom_grid[:, i])
    sum_p1_k6.append(np.sum(prediction_p1_k6, axis=1)) #Summing the probabilities
sum_p1_k6 = np.array(sum_p1_k6)

#Calculating the sum for phoneme 2 for k=6
sum_p2_k6 = []
for i in range(N_f2): #for all 1900 arrays
    #All values present at the i-th row of each array of custom_grid
    prediction_p2_k6 = get_predictions(mu_p2_k6, s_p2_k6, p_p2_k6, custom_grid[:, i])
```

```
        sum_p2_k6.append(np.sum(prediction_p2_k6, axis=1))  #Summing the probabilities
    sum_p2_k6 = np.array(sum_p2_k6)
```

All that is left in this task is to create classification matrix M and store the predictions based on classifying the points on the grid by comparing the summed up probabilities for each of the phoneme. If the summed up probability for phoneme 1 is larger the matrix will store value 0, otherwise it will store value 1.

- If we are comparing the predictions for 3 gaussian cluster for both phoneme run the following code-

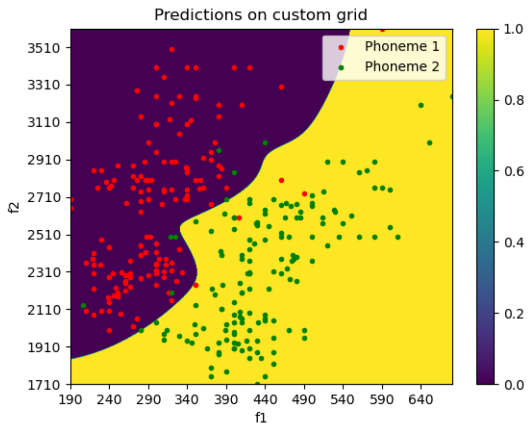
```
#Compare the predictions for the 3 gaussian clusters
for i in range(len(sum_p1_k3)):
    for j in range(len(sum_p1_k3[0])):
        #Compare the probability of which phoneme is higher
        if sum_p1_k3[i][j] > sum_p2_k3[i][j]:
            M[i][j] = 0
        else:
            M[i][j] = 1
```

- If we are comparing the predictions for 6 gaussian cluster for both phoneme run the following code-

```
#Compare the predictions for the 6 gaussian clusters
for i in range(len(sum_p1_k6)):
    for j in range(len(sum_p1_k6[0])):
        if sum_p1_k6[i][j] > sum_p2_k6[i][j]: #Compare the probability of which phoneme
            M[i][j] = 0
        else:
            M[i][j] = 1 1
```

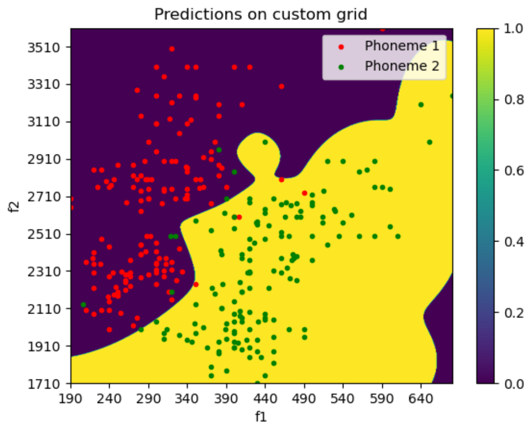
Output of classification matrix "M" for k=3 Gaussian Components:

```
M:
 [[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.]]
```



Output of classification matrix "M" for k=6 Gaussian Components:

```
M:
 [[1. 1. 1. ... 0. 0. 0.]
 [1. 1. 1. ... 0. 0. 0.]
 [1. 1. 1. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```



Conclusion:-

After seeing the plots we can say that the classifier has done extremely well while classifying the data points for 3 gaussian clusters. However it can be noted that the even though the classifier with 6 gaussian clusters is doing even better job at classifying, this could be because of overfitting. Overfitting is resulting in using the 3 gaussian clusters more favourable.

2.5 Task 5

Store the f1, f2 and f1+f2 frequencies in X_full variable.

```
X_full[:,0] = f1
X_full[:,1] = f2
X_full[:,2] = f1 + f2
```

Creating an array that only contains only the samples that belong to the chosen phoneme i.e. p_id = 1.

Here firstly an empty array is created that will ultimately store the indexes of phoneme_id which are equal to p_id i.e. 1. The following code iterates over every phoneme value, then it checks whether the element in phoneme_id is equal to 1 or not. If the condition proves to be true, that index value is stored in the element_index.

```
X_phoneme = np.zeros((np.sum(phoneme_id==1), 3))

element_index = []
for i in range(len(phoneme_id)):#extract the indexes that have the phoneme_id as p_id
    if phoneme_id[i] == p_id:
    element_index.append(i)
```

Afterwards, we iterate over the X_phoneme numpy array. Then we extract the index values stored in the element_index and store it in *index* variable while iterating. Finally, at the *index* from the X_full array we extract the row and update it to the X_phoneme[i] row.

```
for i in range(X_phoneme.shape[0]):
   index = element_index[i] #Store the index

   #Extract the row from X_full from the following "index" and store it to X_phoneme
   X_phoneme[i] = X_full[index]
```

After creating the new dataset having columns X = [F1, F2, F1 + F2], we passed the new data to fit the MoG model. After running the file there were some warnings as *Casting complex values to real discards the imaginary part.* However, after a few iterations there was a runtime warning and error that stopped the execution. The warning and error were was as following:

RuntimeWarning: divide by zero encountered in double_scalars

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

The warning states that a divide by zero was encountered, and because of this when dividing something with zero, gives us a number closer to infinity which too large, which ultimately gives us *ValueError*.

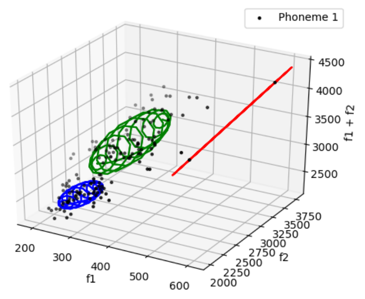
It can be noted that only the diagonal co-variance were being calculated In this task, a new feature set is added to the dataset i.e. F1+F2. This third feature is linearly dependent on the previous two features F1 and F2.

By adding new features does not guarantee that the results will improve. New features can result in our model to be overfit. Therefore, it is far more important in adding important features rather than adding new features to the dataset.

To ensure that the model does not overfit while adding new features, we have to make sure that the covariance matrix does not have the singularity problem.

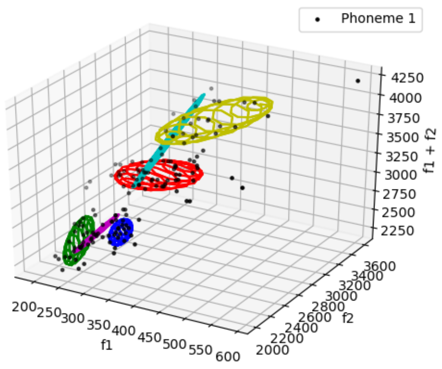
This singularity problem can be solved by adding constant to covariance matrix to ensure that the determinant remains non-zero. We are multiplying the left diagonal of identity matrix with 0.1 after every iteration. This ensure that the values will not be equal to zero, and also making sure that it is invertible. This multipled result is added to the covariance for each gaussian.

```
s[i, :, :] = np.identity(D) * 0.1 + s[i, :, :]
```



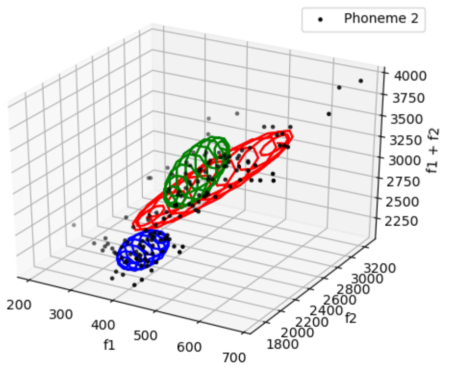
Phoneme 1 k=3

```
Implemented GMM | Mean values
[ 315.7937339  2877.86094839 3193.65468229]
[ 270.34959504 2271.18938709 2541.53898213]
[ 540.02175883 3170.19147774 3710.21323657]
Implemented GMM | Covariances
[[ 3024.08218251  5189.43397183  8213.41615434]
 [ 5189.43397183 70414.68509402 75604.01906585]
 [ 8213.41615434 75604.01906585 83817.53522019]]
[[ 1188.00690568  1264.57898505  2452.48589073]
 [ 1264.57898505 13008.83184849 14273.31083354]
 [ 2452.48589073 14273.31083354 16725.89672427]]
[[  2500.09952655  21999.99583367  24499.99536022]
 [ 21999.99583367 193600.06333628 215599.95916994]
 [ 24499.99536022 215599.95916994 240100.05453016]]
Implemented GMM | Weights
[0.60220734 0.38464051 0.01315215]
```



Phoneme 1 k=6

```
Implemented GMM | Mean values
[ 315.91724338 2786.49197073 3102.40921411]
[ 237.24571681 2234.78028611 2472.02600292]
[ 309.08276216 2324.17933389 2633.26209605]
[ 296.83000253 3063.09150612 3359.92150865]
[ 264.09170677 2300.54985258 2564.64155935]
[ 361.31204166 3240.65701284 3601.96905451]
Implemented GMM | Covariances
[[ 3716.96366144   241.72166993  3958.58533137]
 [  241.72166993  7648.7115594   7890.33322933]
 [ 3958.58533137  7890.33322933 11849.0185607 ]]
[[  361.70419922 -1076.7953359   -715.19113667]
 [-1076.7953359  19401.98981068 18325.09447478]
 [ -715.19113667 18325.09447478 17610.00333811]]
[[ 264.11487406 -358.92458964  -94.90971558]
 [-358.92458964 5727.80376072 5368.77917108]
 [ -94.90971558 5368.77917108 5273.9694555 ]]
[[  145.3297839   3064.52439225  3209.75417615]
 [ 3064.52439225 79324.80318644 82389.2275787 ]
 [ 3209.75417615 82389.2275787  85599.08175485]]
[[  231.4943853   1655.78687788  1887.18126318]
 [ 1655.78687788 12861.67924527 14517.36612315]
 [ 1887.18126318 14517.36612315 16404.64738632]]
[[ 4700.71438417  6062.72714706 10763.34153122]
 [ 6062.72714706 21845.82840384 27908.4555509 ]
 [10763.34153122 27908.4555509  38671.89708212]]
Implemented GMM | Weights
[0.36735067 0.15630367 0.15299407 0.05964069 0.12491759 0.13879332]
```

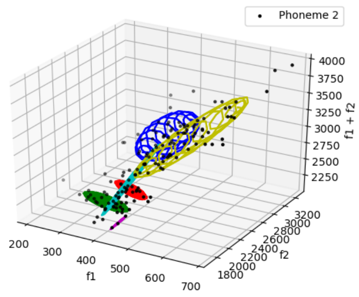


Phoneme 2 k=3

```
Implemented GMM | Mean values
[ 470.90095027 2490.68490965 2961.58585993]
[ 423.65666237 2551.0193007  2974.67596307]
[ 403.12200057 1962.83331436 2365.95531493]

Implemented GMM | Covariances
[[  8128.91135426  19557.3300288   27686.14138306]
 [ 19557.3300288   74919.55392959  94476.78395839]
 [ 27686.14138306  94476.78395839 122163.02534145]]
[[ 1917.79685181  -300.4465799   1617.25027191]
 [ -300.4465799  37663.23598612 37362.68940622]
 [ 1617.25027191 37362.68940622 38980.03967813]]
[[ 1575.42669844  -161.48609061  1413.84060783]
 [ -161.48609061 11434.21345577 11272.62736516]
 [ 1413.84060783 11272.62736516 12686.56797299]]

Implemented GMM | Weights
[0.45915349 0.19257325 0.34827326]
```



Phoneme 2 k=6

```
Implemented GMM | Mean values
[ 422.01635143 2059.84282432 2481.85917574]
[ 368.98750936 1943.8097031  2312.79721246]
[ 437.39382987 2570.53260275 3007.92643261]
[ 405.33117179 2154.48791674 2559.81908853]
[ 439.65006531 1757.90201212 2197.55207744]
[ 508.52371688 2559.88061976 3068.40433664]

Implemented GMM | Covariances
[[ 1794.41134195 -2212.14671798  -417.83537603]
 [-2212.14671798  3660.2081451   1447.96142712]
 [ -417.83537603  1447.96142712  1030.22605109]]
[[ 3211.25379986 -2960.66722526   250.4865746 ]
 [-2960.66722526  3990.04672675  1029.27950149]
 [  250.4865746   1029.27950149  1279.86607609]]
[[ 3251.47196172   737.18205197  3988.55401368]
 [  737.18205197 28917.99690562 29655.07895758]
 [ 3988.55401368 29655.07895758 33643.73297126]]
[[  334.59583815  4131.10944591  4465.60528406]
 [ 4131.10944591 59008.41039209 63139.419838  ]
 [ 4465.60528406 63139.419838   67605.12512206]]
[[  65.48856502  357.82677402  423.21533904]
 [ 357.82677402 2009.02710864 2366.75388266]
 [ 423.21533904 2366.75388266 2790.06922171]]
[[  5627.78899057  16787.72346919  22415.41245976]
 [ 16787.72346919  75811.17817711  92598.8016463 ]
 [ 22415.41245976  92598.8016463  115014.31410606]]

Implemented GMM | Weights
[0.11509717 0.17541028 0.25972943 0.15607613 0.01906525 0.27462175]
```

Conclusion:- From the results it can be seen that the data has fit reasonably well without overfitting for all the models.