Assignment 2: Clustering and MoG

1. MoG Modelling using the EM Algorithm

Task 1:

'PB_data.npy' is the file that contains the dataset that is supposed to use for clustering and MoG. The dataset consists of phoneme Id along with fundamental frequency and first 3 formant frequencies (f0, f1, f2, f3). The data is preloaded in f1 and f2 array for f1 and f2.

task1.py

```
# Initialize array containing f1 & f2, of all phonemes.
X_full = np.zeros((len(f1), 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)
```

```
{'f3': array([2850, 2790, 2640, ..., 3380, 2160, 2200], dtype=uint16), 'f2': array([2280, 2400, 2030, ...,
 1140, 1850, 1830], dtype=uint16), 'f1': array([240, 280, 390, ..., 500, 740, 660], dtype=uint16), 'f0':
 array([160, 186, 203, ..., 334, 308, 328], dtype=uint16), 'phoneme_id': array([ 1,  1,  2, ...,  9, 10, 10],
 dtype=uint8)}
f1 statistics:
Min: 190.00 Mean: 563.30 Max: 1300.00 Std: 201.1881 | Shape: 1520
f2 statistics:
Min: 560.00 Mean: 1624.38 Max: 3610.00 Std: 636.8032 | Shape: 1520
[[ 240. 2280.]
 [ 280. 2400.]
 [ 220. 2220.]
 [ 210. 2360.]
 [ 250. 2180.]
 [ 244. 2300.]
 [ 300. 2240.]
 [ 280. 2450.]
 [ 310. 2310.]
 [ 260. 2250.]
 [ 280. 3140.]
 [ 300. 3400.]
 [ 330. 3050.]
 [ 340. 2860.]
 [ 380. 3200.]
 [ 350. 3250.]
 [ 300. 3250.]
 [ 275. 3280.]
 [ 370. 2950.]
 [ 370. 2910.]
 [ 350. 3240.]
 [ 344. 3120.]]
```

Figure 1: 2D Matrix which contains f1 and f2 frequencies for every phoneme ID

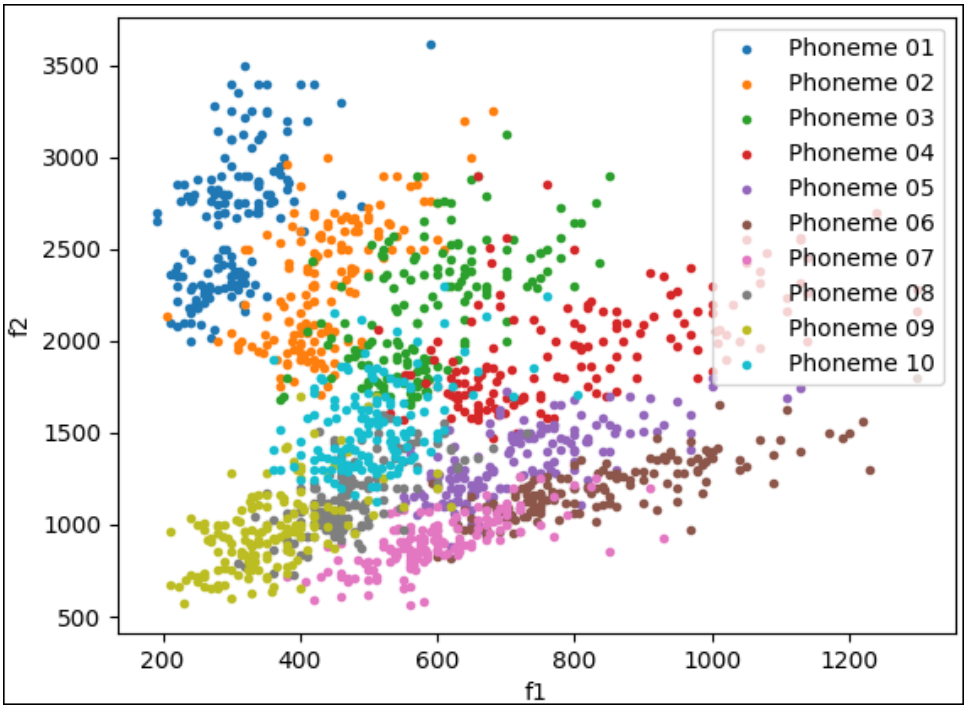
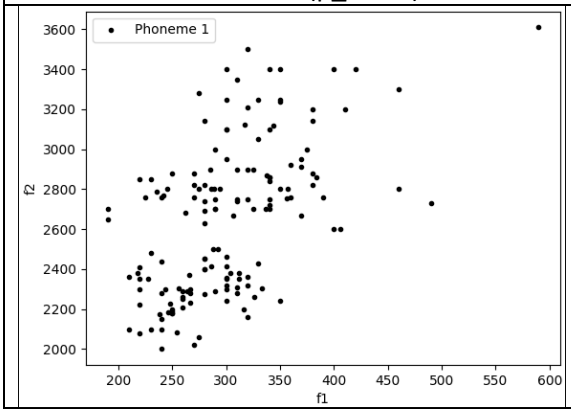
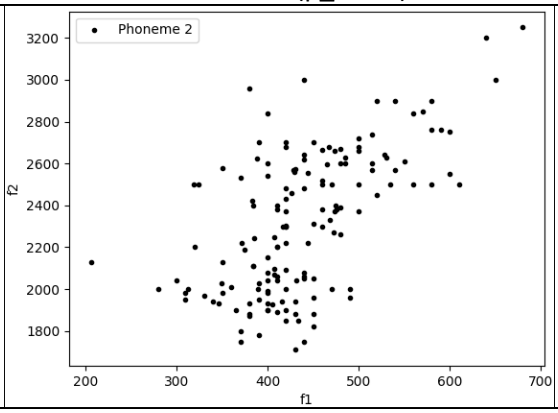


Figure 2: Scatter plot for f1 vs f2

task1.py

```
X_phoneme_1 = np.zeros((np.sum(phoneme_id == 1), 2))
X_phoneme_1_dup = np.zeros((np.sum(phoneme_id == 1), 2))
pos = np.zeros(np.sum(phoneme_id == 1))
j = 0
for i in range(len(phoneme_id)):
    if phoneme_id[i] == 1:
        pos[j] = i
        j += 1
new_f1 = np.zeros(np.sum(phoneme_id == 1))
new_f2 = np.zeros(np.sum(phoneme_id == 1))
temp_f1 = 0
for k in range(len(f1)):
    for position in range(len(pos)):
        if k == pos[position]:
            new_f1[temp_f1] = f1[k]
            temp_f1 += 1
temp_f2 = 0
for l in range(len(f2)):
    for position in range(len(pos)):
        if l == pos[position]:
            new_f2[temp_f2] = f2[l]
            temp_f2 += 1
X_phoneme_1_dup[:, 0] = new_f1
X_phoneme_1_dup[:, 1] = new_f2
print(X_phoneme_1_dup)
```

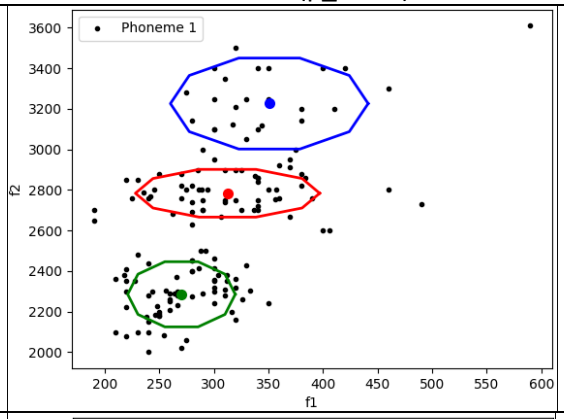
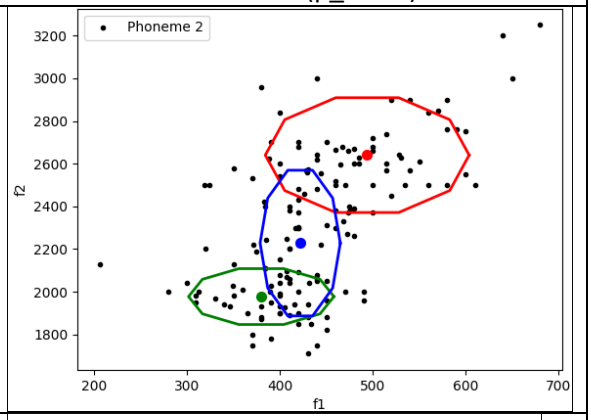
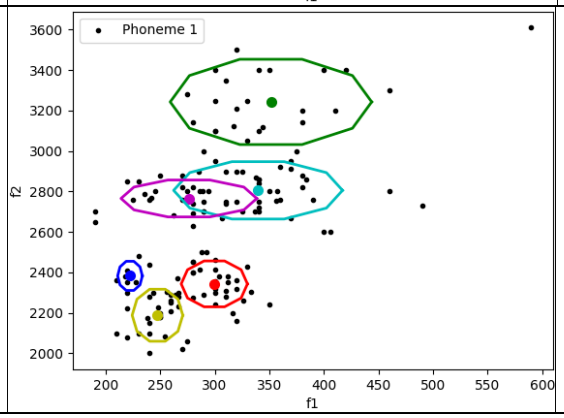
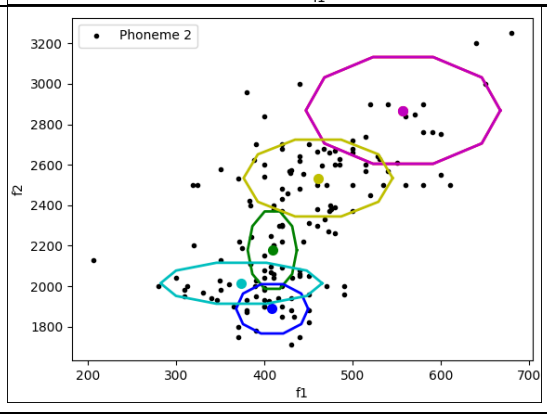
Phoneme 1 (p_id = 1)	Phoneme 2 (p_id = 2)
	

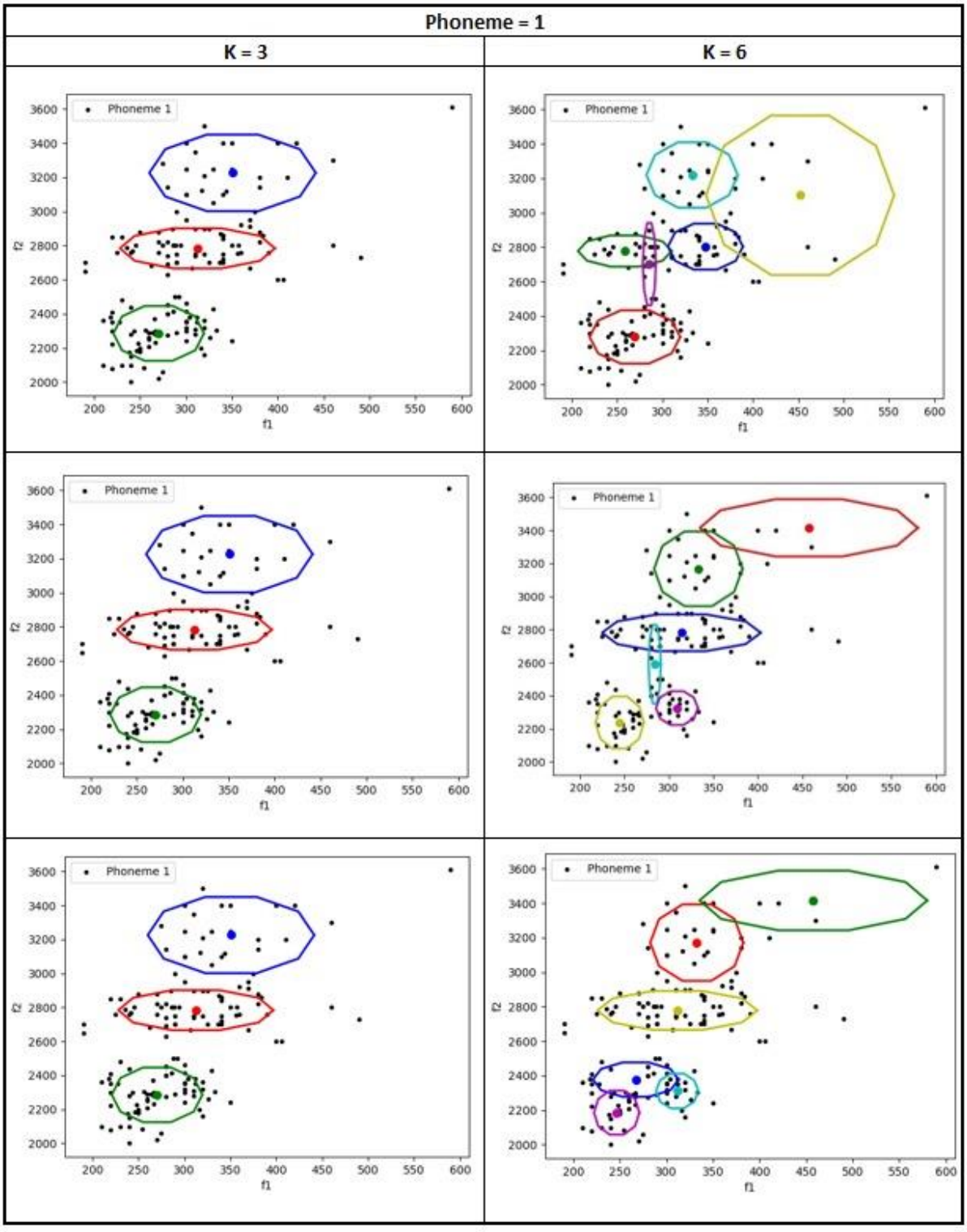
Task 2:

task2.py

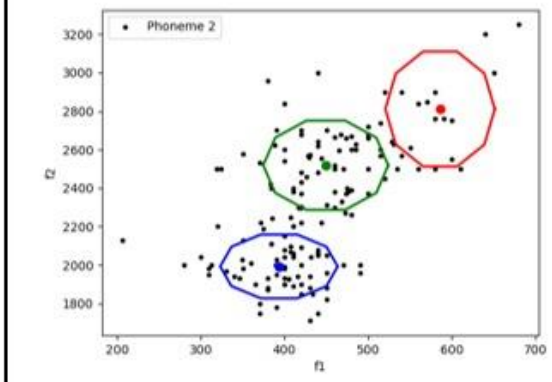
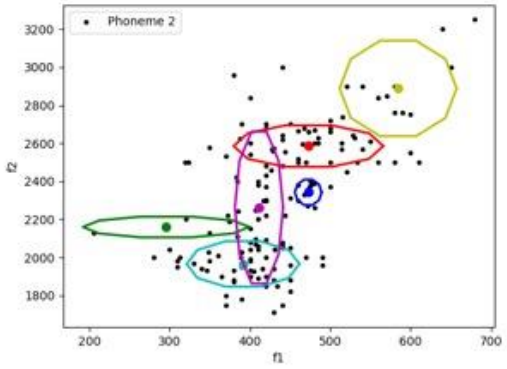
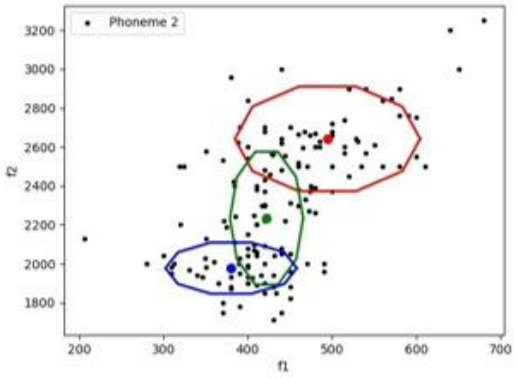
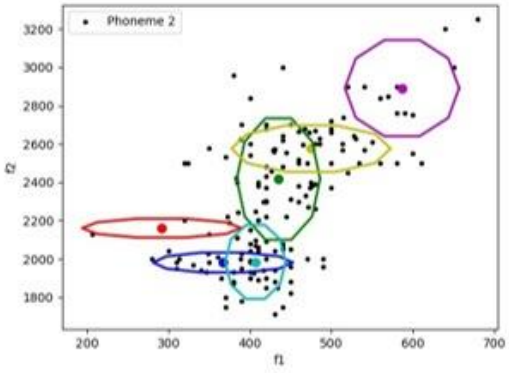
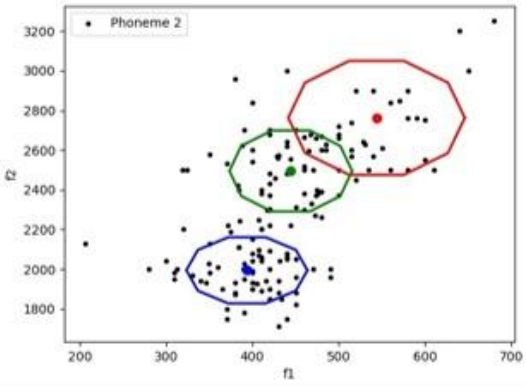
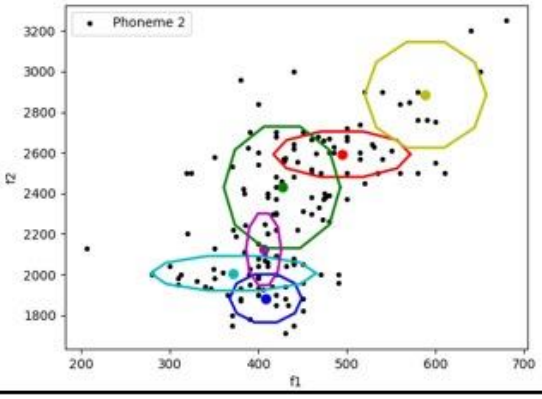
```
X_full = np.zeros((len(f1), 2))
########################################
# Write your code here
# Store f1 in the first column of X_full, and f2 in the second column of X_full
X_full[:, 0] = f1
X_full[:, 1] = f2
########################################/
X_full = X_full.astype(np.float32)
```

```
X_phoneme = np.zeros((np.sum(phoneme_id == p_id), 2))
pos = np.zeros(np.sum(phoneme_id == p_id))
j = 0
for i in range(len(phoneme_id)):
    if phoneme_id[i] == p_id:
        pos[j] = i
        j += 1
new_f1 = np.zeros(np.sum(phoneme_id == p_id))
new_f2 = np.zeros(np.sum(phoneme_id == p_id))
temp_f1 = 0
for index in range(len(f1)):
    for h in range(len(pos)):
        if index == pos[h]:
            new_f1[temp_f1] = f1[index]
            temp_f1 += 1
temp_f2 = 0
index = 0                    # re-initializing index variable
for index in range(len(f2)):
    for h in range(len(pos)):
        if index == pos[h]:
            new_f2[temp_f2] = f2[index]
            temp_f2 += 1
X_phoneme[:, 0] = new_f1
X_phoneme[:, 1] = new_f2
print(X_phoneme)
```

	Phoneme 1 (p_id = 1)	Phoneme 2 (p_id = 2)
K = 3		
K = 6		



There are fewer local minima for k=3 as compared to k=6 for phoneme ID =1.

Phoneme = 2	
K = 3	**K = 6**
Run 3 	
Run 5 	
Run 7 	

It is clear from the above tables that for every run, the shapes and colors of clusters are changing. (Color is changing as it is randomly selected). We have achieved clustering the data using MoG with EM algorithm. The values of mean(mu) and variance are randomly selected for localization.

For optimizing the algorithm, we must run the code for multiple times (preferably for higher values of k). The cost associated with the runs will be compared and the lowest cost model will be used for training data set.

For implementing MoG with EM algorithm, there are have 2 steps E-step and M-step.

- E step: Here, we get the predictions via get_prediction() function and by passing mu, s, p and X as its parameters. Then, normalizing the result and saving it in Z.
- M step: Here, the mu is maximized. We are fitting Guassians with the variance matrices.

For every iteration, we are predicting and normalizing and then maximizing the mu.

task2.py

```
for t in range(n_iter):
    print('Iteration {:03}/{:03}'.format(t+1, n_iter))

    # Do the E-step
    Z = get_predictions(mu, s, p, X)
    Z = normalize(Z, axis=1, norm='l1')

    # Do the M-step:
    for i in range(k):
        mu[i, :] = np.matmul(X.transpose(), Z[:, i]) / np.sum(Z[:, i])
        # We will fit Gaussians with diagonal covariance matrices
        mu_i = mu[i, :]
        mu_i = np.expand_dims(mu_i, axis=1)
        mu_i_repeated = np.repeat(mu_i, N, axis=1)
        X_minus_mu = (X.transpose() - mu_i_repeated)**2
        res_1 = np.squeeze(np.matmul(X_minus_mu, np.expand_dims(Z[:, i], axis=1)))/np.sum(Z[:, i])
        s[i, :, :] = np.diag(res_1)
        p[i] = np.mean(Z[:, i])
    ax1.clear()
    # plot the samples of the dataset, belonging to the chosen phoneme (f1 & f2, phoneme 1 or 2)
    plot_data(X=X_phoneme, title_string=title_string, ax=ax1)
    # Plot gaussians after each iteration
    plot_gaussians(ax1, 2*s, mu)
print('\nFinished.\n')
```

Task 3:

task3.py

For k = 3 and phoneme ID (p_id) = 1 and 2. We are loading the data. The GMM model built in task 2 is getting stored and is being used further.

```
# File that contains the data
data_npy_file = 'data/PB_data.npy'
data_npy_phoneme_1 = 'data/GMM_params_phoneme_01_k_03.npy'
data_npy_phoneme_2 = 'data/GMM_params_phoneme_02_k_03.npy'

# Loading data from .npy file
data = np.load(data_npy_file, allow_pickle=True)
data = np.ndarray.tolist(data)

data_1 = np.load(data_npy_phoneme_1, allow_pickle=True)
data_1 = np.ndarray.tolist(data_1)

data_2 = np.load(data_npy_phoneme_2, allow_pickle=True)
data_2 = np.ndarray.tolist(data_2)
```

For k = 36 and phoneme ID (p_id) = 1 and 2. We are loading the data. The GMM model built in task 2 is getting stored and is being used further.

```
# File that contains the data
data_npy_file = 'data/PB_data.npy'
data_npy_phoneme_1 = 'data/GMM_params_phoneme_01_k_06.npy'
data_npy_phoneme_2 = 'data/GMM_params_phoneme_02_k_06.npy'

# Loading data from .npy file
data = np.load(data_npy_file, allow_pickle=True)
data = np.ndarray.tolist(data)

data_1 = np.load(data_npy_phoneme_1, allow_pickle=True)
data_1 = np.ndarray.tolist(data_1)

data_2 = np.load(data_npy_phoneme_2, allow_pickle=True)
data_2 = np.ndarray.tolist(data_2)
```

```
# Initialize array containing f1 & f2, of all phonemes.
X_full = np.zeros((len(f1), 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.array([])

for i in range(len(f1)):
    if phoneme_id[i] == 1:
        if np.size(X_phonemes_1_2) == 0:
            X_phonemes_1_2 = [X_full[i][0], X_full[i][1]]
        else:
            row = [X_full[i][0], X_full[i][1]]
            X_phonemes_1_2 = np.vstack((X_phonemes_1_2, row))
for i in range(len(f1)):
    if phoneme_id[i] == 2:
        if np.size(X_phonemes_1_2) == 0:
            X_phoneme_1_2 = [X_full[i][0], X_full[i][1]]
        else:
            row = [X_full[i][0], X_full[i][1]]
            X_phonemes_1_2 = np.vstack((X_phonemes_1_2, row))
########################################/
print(X_phonemes_1_2)
```

```
mu1 = data_1['mu']
s1 = data_1['s']
p1 = data_1['p']

mu2 = data_2['mu']
s2 = data_2['s']
p2 = data_2['p']

prediction1 = get_predictions(mu1, s1, p1, X_phonemes_1_2)
prediction2 = get_predictions(mu2, s1, p2, X_phonemes_1_2)

data1 = int(len(X_phonemes_1_2)/2)
GMM1 = 0
GMM2 = 0
for i in range(data1):
    if np.sum(prediction1[i]) > np.sum(prediction2[i]):
        GMM1 += 1
for j in range(data1, len(X_phonemes_1_2)):
    if np.sum(prediction1[j]) <= np.sum(prediction2[j]):
        GMM2 += 1
accuracy = ((GMM1+GMM2)/(len(X_phonemes_1_2))*100)

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

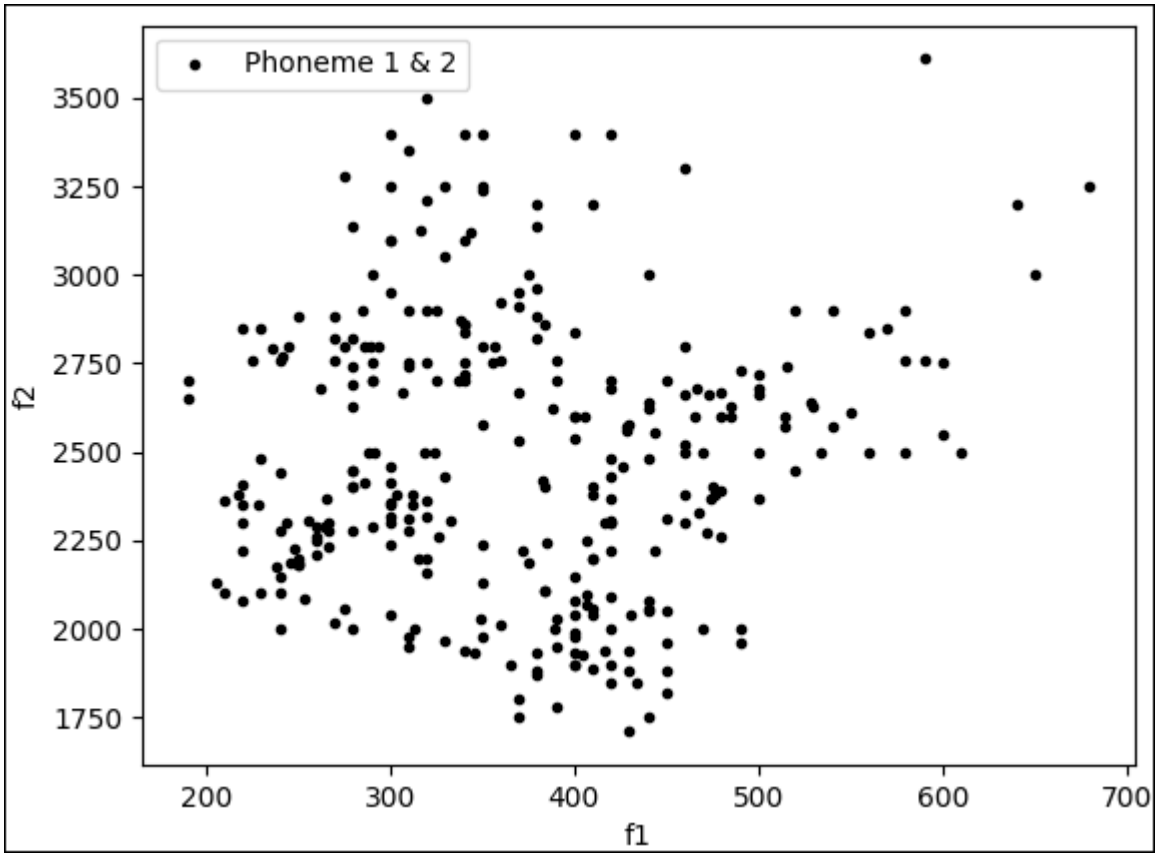
These models are stored in task 2. In task 2, the final values of mu, p and s are stored along with the model. Mu1, p1 and s1 are the variables for phoneme ID = 1 and k = 3 or 6 and mu2, p2 and s2 are the variables for phoneme ID = 2 and k = 3 or 6. X_phoneme_1_2 has values of f1 and f2 for phoneme ID = 1 and phoneme ID = 2. prediction1 is for phoneme ID 1 and prediction2 is an array of prediction of probabilities that a point is belonging to phoneme ID 2.

```
Accuracy using GMMs with 3 components: 93.42%
```

The accuracy for phoneme ID 1 and 2 for k=3 is 93.42%. The mis-classification error is 6.58%.

```
Accuracy using GMMs with 6 components: 93.42%
```

The accuracy for phoneme ID 1 and 2 for k=6 is 93.42%. The mis-classification error is 6.58%.



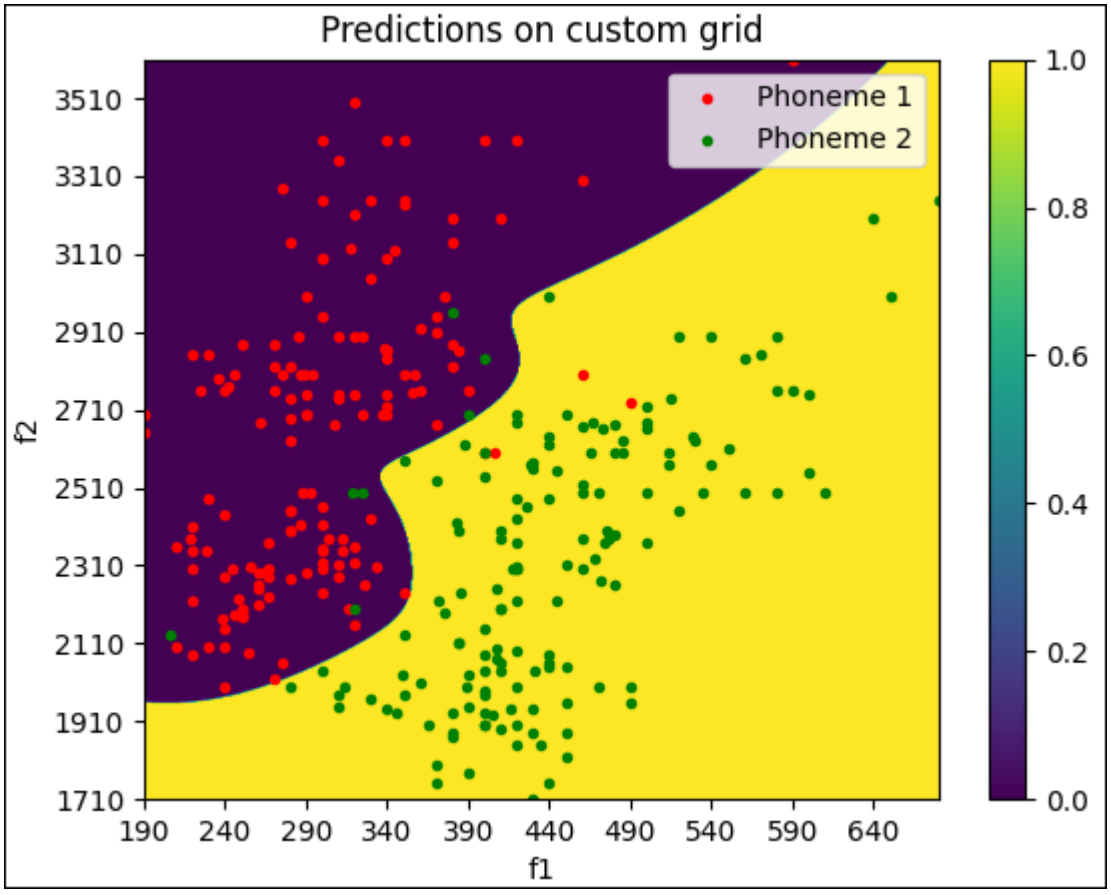
Task 4:

Creating a boundary line to predict if the new data point is assigned to phoneme ID 1 or phoneme ID 2. We create 2D matrix of size (max_f1 – min_f1, max_f2 – min_f2). The matrix has a grid of values [f1, f2] at the given position.

If the sum of probabilities for predicting the data point for phoneme ID = 1 is greater than sum of probabilities for predicting the data point for phoneme ID = 2 then we are assigning the value 0 to the matrix M. Else, value assigned is 1.

task4.py

```
grid = np.zeros((N_f1, N_f2))
M = np.zeros((N_f2, N_f1))
grid_i = np.arange(min_f1, max_f1)
grid_j = np.arange(min_f2, max_f2)
for i in range(N_f2):
    print(i)
    for j in range(N_f1):
        grid = np.array([[grid_i[j], grid_j[i]]])
        prediction1 = get_predictions(mu1, s1, p1, grid)
        prediction2 = get_predictions(mu2, s2, p2, grid)
        if np.sum(prediction1) > np.sum(prediction2):
            M[i][j] = 0.0
        else:
            M[i][j] = 1.0
```




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

We are plotting the graph for classifying phoneme 1 and phoneme 2 for f1 vs f2. The boundary is classifying the data points between the two phonemes. The red dots represent phoneme 1 and it is assigned value 1 and phoneme 2 represents green dots and the value assigned is 0. Purple region has value 1 and yellow region has value 0. It is clear from the above graph some data points are not correctly classified.

Task 5:

task5.py

```
# Initialize array containing f1, f2 & f1+f2, of all phonemes.
X_full = np.zeros((len(f1), 3))
##########################################
# Write your code here
# Store f1 in the first column of X_full, f2 in the second column of X_full
# and f1+f2 in the third column of X_full
X_full[:, 0] = f1
X_full[:, 1] = f2
X_full[:, 2] = f1+f2
##########################################/
X_full = X_full.astype(np.float32)
```

```
X_phoneme = np.zeros((np.sum(phoneme_id == p_id), 3))
X_phoneme[:, 0] = f1[phoneme_id == p_id]
X_phoneme[:, 1] = f2[phoneme_id == p_id]
X_phoneme[:, 2] = f1[phoneme_id == p_id]+f2[phoneme_id == p_id]
```

```
# Suggest ways of overcoming the singularity
s[i, :, :] = s[i, :, :] * np.identity(D)
```