Name: Chandramohan Natarajan

Student ID: 190617866

Assignment: ML_2

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. Using the code in task_1.py, 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. [20 points]

Coding part -1

 # Store f1 in the first column of X_full, and f2 in the second column of X_full

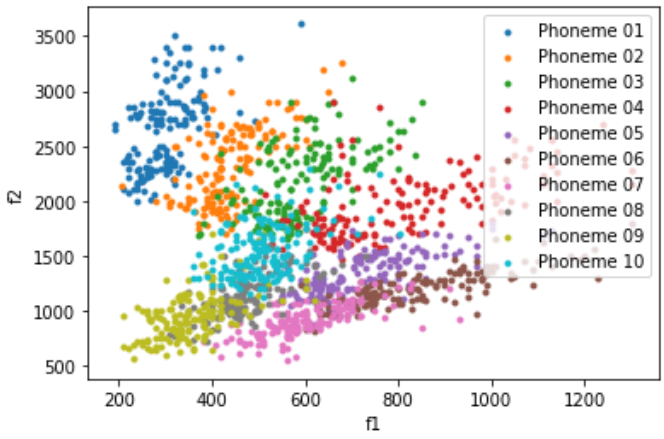
```
X_full[:, 0] = f1
X_full[:, 1] = f2
print(X_full)
X_full = X_full.astype(np.float32)
```

Coding part-2

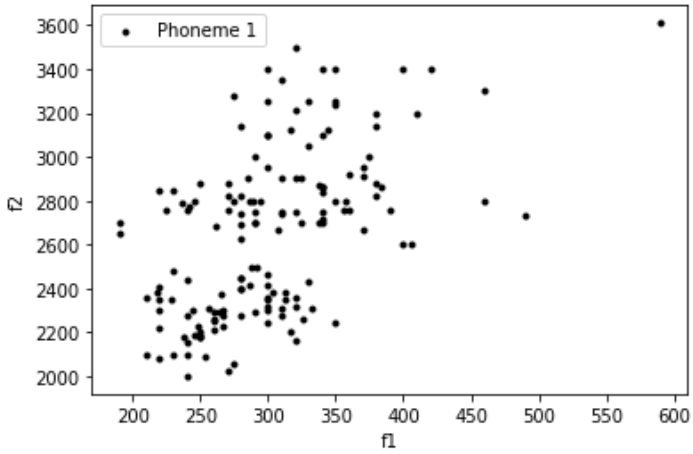
 # Create an array named "X_phoneme_1", containing only samples that belong to the chosen phoneme.
 # The shape of X_phoneme_1 will be two-dimensional. Each row will represent a sample of the dataset, and each column will represent a feature (e.g. f1 or f2)
 # Fill X_phoneme_1 with the samples of X_full that belong to the chosen phoneme
 # To fill X_phoneme_1, you can leverage the phoneme_id array, that contains the ID of each sample of X_full

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

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

k=0

for i in range(len(phoneme_id)):

   if phoneme_id[i] ==1:

      X_phoneme_1[k,0] = f1[i]

      X_phoneme_1[k,1] = f2[i]
```

Plot of F1 vs F2



Phoneme 1



Task 2:

Train the data for phonemes 1 and 2 with MoGs. You are provided with python files task_2.py, get_predictions.py and plot_gaussians.py. Specifically, you are required to:

1. Look at the task_2.py and get_predictions.py code files and understand what they are calculating. Pay particular attention to the initialisation of the means and covariances (also note that in this Task, the code is only estimating diagonal covariances).

2. Generate a dataset stored in the variable "X_phoneme_1", that contains only the F1 and F2 for the first phoneme.

3. Run task_2.py on the dataset using K=3 Gaussians (run the code a number of times and note the differences.) Save your MoG model: this should comprise the variables mu, s and p.

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

5. Repeat steps 2-4 for the second phoneme

Include in your report the lines of code you wrote, and results that illustrate the learnt models. [20 points]

Coding part

```
# Create an array named "X_phoneme", containing only samples that belong to the chosen phoneme.
# The shape of X_phoneme will be two-dimensional. Each row will represent a sample of the dataset, and each column will represent a feature (e.g. f1 or f2)
# Fill X_phoneme with the samples of X_full that belong to the chosen phoneme
# To fill X_phoneme, you can leverage the phoneme_id array, that contains the ID of each sample of X_full
```

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

m = 0

for i in range(len(phoneme_id)):

    if phoneme_id[i] == 1:

        X_phoneme[m, 0] = f1[i]

        X_phoneme[m, 1] = f2[i]

        m = m+1
```

Result:

K=3 [1st Instance] p_id =1

Implemented GMM | Mean values

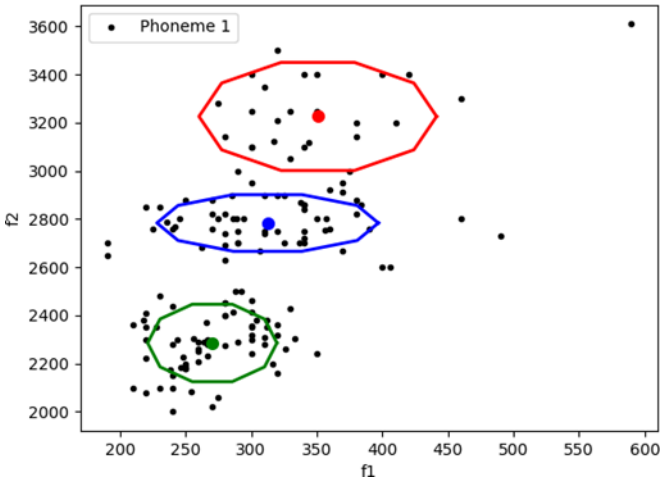
```
[ 268.55857022 2272.50067627]
[ 352.8522372  3035.28369147]
[ 296.19636246 2772.12205229]
```

Implemented GMM | Covariances

```
[[ 1200.65245239     0.       ]
 [    0.       13051.54364304]]
[[ 4313.14982822     0.       ]
 [    0.       78394.50759631]]
[[ 2100.40996989     0.       ]
 [    0.       28996.16918523]]
```

Implemented GMM | Weights

[0.39677607 0.28298583 0.3202381]



K=3 [2nd Instance] p_id =1

Implemented GMM | Mean values

[297.95941381 2748.4762065]
[266.55073095 2266.37765155]
[355.458757 3046.71607666]

Implemented GMM | Covariances

[[2047.76501793 0.]
[0. 45646.94875767]]
[[1154.96684493 0.]
[0. 12872.75483881]]
[[4490.96449857 0.]
[0. 76826.14842277]]

Implemented GMM | Weights

[0.37451229 0.36736153 0.25812618]

K=3 [3rd Instance] p_id =1

Implemented GMM | Mean values

[311.6418995 2864.99631036]
[268.55263614 2276.38496236]
[357.7582306 3004.09362928]

Implemented GMM | Covariances

[[2090.36487201 0.]
[0. 51521.07303458]]
[[1191.33461891 0.]
[0. 13555.95566351]]
[[7737.98223405 0.]
[0. 102471.59711906]]

Implemented GMM | Weights

[0.44338821 0.40409613 0.15251567]

K=6 [1st instance] p_id =1

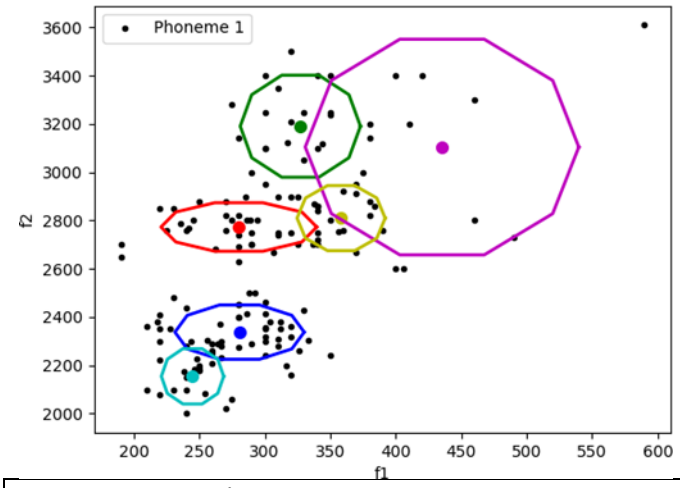
Implemented GMM | Mean values
[270.09853158 2284.2714882]
[340.74671248 2837.89425772]
[279.84649025 2771.90396414]
[329.85001096 3229.44744778]
[425.27674984 3107.90473027]
[279.84649025 2771.90396414]

Implemented GMM | Covariances

[[1210.30133384 0.]
[0. 14173.34449624]]
[[1347.70394138 0.]
[0. 22803.24132091]]
[[2060.4550819 0.]
[0. 4714.48075287]]
[[1168.3759549 0.]
[0. 20805.55650215]]
[[6676.34391242 0.]
[0. 103622.91352104]]
[[2060.4550819 0.]
[0. 4714.48075287]]

Implemented GMM | Weights

[0.43159696 0.17838647 0.10489503 0.11999054 0.06023596 0.10489503]



K=6 2nd instance p_id =1

Implemented GMM | Mean values

[341.19742994 2787.84141308]
[317.05886044 2998.4940332]
[235.93847034 2201.82265199]
[397.58642955 3114.98982306]
[258.87928984 2634.15506831]
[284.30807881 2300.83225983]

Implemented GMM | Covariances

[[1286.719194 0.]
[0. 7367.10924441]]
[[1085.81752719 0.]
[0. 63844.13534475]]
[[244.49769141 0.]
[0. 15130.92484459]]
[[5950.73973871 0.]
[0. 86668.3372506]]
[[1192.64929474 0.]
[0. 53781.36503281]]
[[970.57055045 0.]
[0. 8535.79289386]]

Implemented GMM | Weights

[0.14435597 0.21173145 0.10543163 0.10025742 0.17495592 0.26326762]

K=3 3rd instance p_id =1

Implemented GMM | Mean values

[339.18307987 2746.76906573]
[291.62163505 2779.72195184]
[299.92386104 2341.1275749]
[332.76984139 3100.60653296]
[427.94469537 3291.52560498]
[243.85116144 2233.63020384]

Implemented GMM | Covariances

[[5514.46282887 0.]
[0. 7520.35522416]]
[[2253.49236565 0.]
[0. 5082.60172899]]
[[484.01975095 0.]
[0. 7196.61840206]]
[[1179.32838813 0.]
[0. 42438.80275117]]
[[7800.98441411 0.]

[0. 56621.89896481]]
[[379.89700787 0.]
[0. 14592.94665759]]

Implemented GMM | Weights

[0.10377635 0.21498019 0.20523265 0.20811764 0.03930392 0.22858925]

K =3 1st instance p_id =2

Implemented GMM | Mean values

[270.20166348 2284.56910331]
[352.68634616 3079.32203381]
[298.64835922 2779.67885839]

Implemented GMM | Covariances

[[1211.36399784 0.]
[0. 14195.24499994]]
[[4121.42658582 0.]
[0. 67474.90159787]]
[[2555.78414017 0.]
[0. 5947.89239155]]

Implemented GMM | Weights

[0.43223207 0.27592391 0.29184402]

K=3 2nd instance p_id =2

Implemented GMM | Mean values

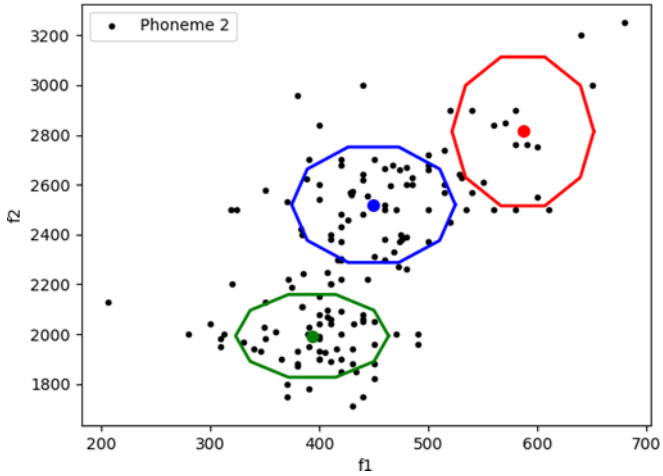
[285.58900975 2722.74694813]
[349.20321658 3001.97261468]
[268.0734615 2267.21545039]

Implemented GMM | Covariances

[[1793.09381982 0.]
 [0. 33155.75128469]]
[[3787.52898486 0.]
 [0. 73935.77936201]]
[[1196.41879687 0.]
 [0. 12676.77225991]]

Implemented GMM | Weights

[0.27131984 0.35051727 0.3781628



K=3 3rd instance p_id =2

Implemented GMM | Mean values

[354.2605065 3060.58523443]
[268.34850437 2351.96935577]
[316.26619416 2844.89186002]

Implemented GMM | Covariances

[[4950.3600424 0.]
 [0. 82113.85530984]]
[[1288.67459979 0.]
 [0. 39250.89314329]]
[[1696.86104384 0.]
 [0. 20869.78778033]]

Implemented GMM | Weights

[0.24095224 0.50415308 0.25489468]

K=6 1st instance p_id = 2

Implemented GMM | Mean values

[300.73846551 2786.12976633]
[241.91127438 2212.28180786]
[297.80935415 2360.99962284]
[351.08710898 3226.77838181]
[357.2514668 2778.20472499]
[286.47391855 2321.10678901]

Implemented GMM | Covariances

[[2548.06954502 0.]
[0. 6349.43113621]]
[[323.0357791 0.]
[0. 14329.31160714]]
[[362.21881786 0.]
[0. 7854.42663633]]
[[4116.16687852 0.]
[0. 28185.61060638]]
[[4867.08923523 0.]
[0. 15714.93034809]]
[[1099.44386369 0.]
[0. 6590.36303741]]

Implemented GMM | Weights

[0.3043539 0.18019547 0.0932114 0.18243469 0.07959411 0.16021043]

K = 6 2nd instance p_id =2

Implemented GMM | Mean values

[376.77938549 3118.2958968]
[336.90793493 2995.62037084]
[308.96863178 2327.52685971]
[263.23129424 2353.24587883]
[245.58587004 2188.42754481]
[290.27693741 2775.41324998]

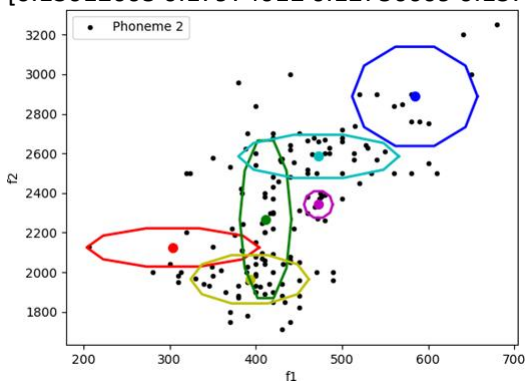
Implemented GMM | Covariances

[[6162.64508246 0.]
[0. 81584.01363528]]
[[1074.27775142 0.]
[0. 53284.42785822]]
[[281.69669145 0.]
[0. 6335.09568894]]
[[983.25091604 0.]
[0. 6997.90272049]]
[[309.12253583 0.]
[0. 10481.47469213]]

[[2381.17664643 0.   ]
 [0.   7260.55685464]]

Implemented GMM | Weights

[0.13012605 0.17974012 0.12756609 0.13701831 0.16401801 0.26153142]



K=6 3rd instance p_id = 2

Implemented GMM | Mean values

[382.98052024 3102.06453173]
[242.25142883 2217.4703343]
[291.513986   2426.10379399]
[296.87043097 2774.65873708]
[292.55024184 2332.98356393]
[326.85347158 3025.26995712]

Implemented GMM | Covariances

[[5790.0302896   0.   ]
 [0.   83713.03759932]]
[[337.56598943   0.   ]
 [0.   14062.80457061]]
[[483.47548643   0.   ]
 [0.   31341.53714278]]
[[2929.86255368   0.   ]
 [0.   5136.65158349]]
[[824.46402456   0.   ]
 [0.   6430.78221535]]
[[1171.12742092   0.   ]
 [0.   55096.86881963]]

Implemented GMM | Weights

[0.12159693 0.19133978 0.03332925 0.25953852 0.21187106 0.18232446]

Explanation:

Every gaussian distribution associated with the genome has been calculated with the mean of the respective cluster, co variance matrix of that distribution and associated determinants along with data sets. The probability is well established using the code.

We get a training model with mean , covariance matrix and weight to test the data points of phoneme 2.

Task 3:

Use the 2 MoGs (K=3) learnt in Task 2 to build a classifier to discriminate between phonemes 1 and 2. Classify using the Maximum Likelihood (ML) criterion (feel free to hack parts from the code in task_3.py, and utilize the get_predictions.py file so that you calculate the likelihood of a data vector for each of the two MoG models). Calculate the mis-classification error. Remember that a classification under the ML compares $p(x;\theta 1)$, where $\theta 1$are the parameters of the MoG learnt for the first phoneme, with $p(x;\theta 2)$, where $\theta 2$ are the parameters of the MoG learnt for the second phoneme.

Repeat this for K=6 and compare the results.

Include in your report the lines of the code that your wrote, explanations of what the code does and comment on the differences on the classification performance [20 points]

Coding part

 # Create an array named "X_phonemes_1_2", containing only samples that belong to phoneme 1 and samples that belong to phoneme 2.

 # The shape of X_phonemes_1_2 will be two-dimensional. Each row will represent a sample of the dataset, and each column will represent a feature (e.g. f1 or f2)

 # Fill X_phonemes_1_2 with the samples of X_full that belong to the chosen phonemes

 # To fill X_phonemes_1_2, you can leverage the phoneme_id array, that contains the ID of each sample of X_full

```
n1 = np.sum(phoneme_id==1)
n2 = np.sum(phoneme_id==2)
# X_phonemes_1_2 = ...
X_phonemes_1_2 = np.zeros((n1+n2, 2))
m = 0
for i in range(len(phoneme_id)):
    if phoneme_id[i] == 1:
        X_phonemes_1_2[j, 0] = f1[i]
        X_phonemes_1_2[j, 1] = f2[i]
        m = m+1
 for i in range(len(phoneme_id)):
    if phoneme_id[i] == 2:
        X_phonemes_1_2[m, 0] = f1[i]
        X_phonemes_1_2[m, 1] = f2[i]
       m = m+1
```

Explanation:

The data has been trained individually and all correspondence parameters are computed like mean, covariance and weight belonging to gaussian distributions of respective phonemes. As required, combined data set is augmented as phoneme_1_2. All those available datapoints are identified with respective clusters based on prediction code part and stored in Z1, Z2 respectively. Once the combined dataset id available the much awaited probability part is found for the gaussian distribution upon the phenome working upon. Subsequently detailed comparison is made among the phonemes and get identified with the clusters.

The two MoGs with Phoneme 1 and Phoneme 2 has been used to build a classifier which would be instrumental in discriminating the points. ML criterion plays a role in achieving likelihood of data vector.

The calculated results for misclassification errors are as follows.

K =3 /P_id=1 -→ 4%

K=3 /P_id=2 -→ 5%

K=6/P_id=1 → 4%

K=6/P_id=2 → 5%

K=3 P_id=1

Implemented GMM | Mean values

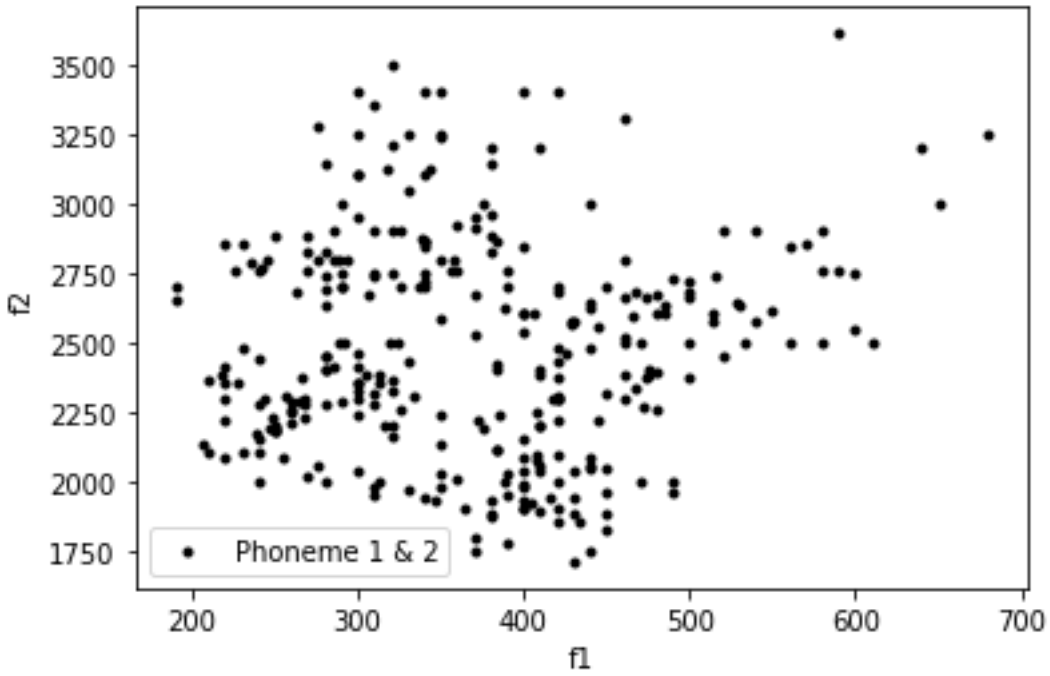
[307.30120743 2781.47703463]
[270.37089257 2285.73163282]
[353.08816365 3158.61653112]

Implemented GMM | Covariances

[[3135.11979776 0.]
[0. 7149.94489162]]
[[1214.65125349 0.]
[0. 14389.33943819]]
[[4226.90321909 0.]
[0. 49226.20070589]]

Implemented GMM | Weights

[0.34544714 0.43531456 0.2192383]



Accuracy using GMMs with 3 components: 0.96%

K =3 p_id =2

Implemented GMM | Mean values

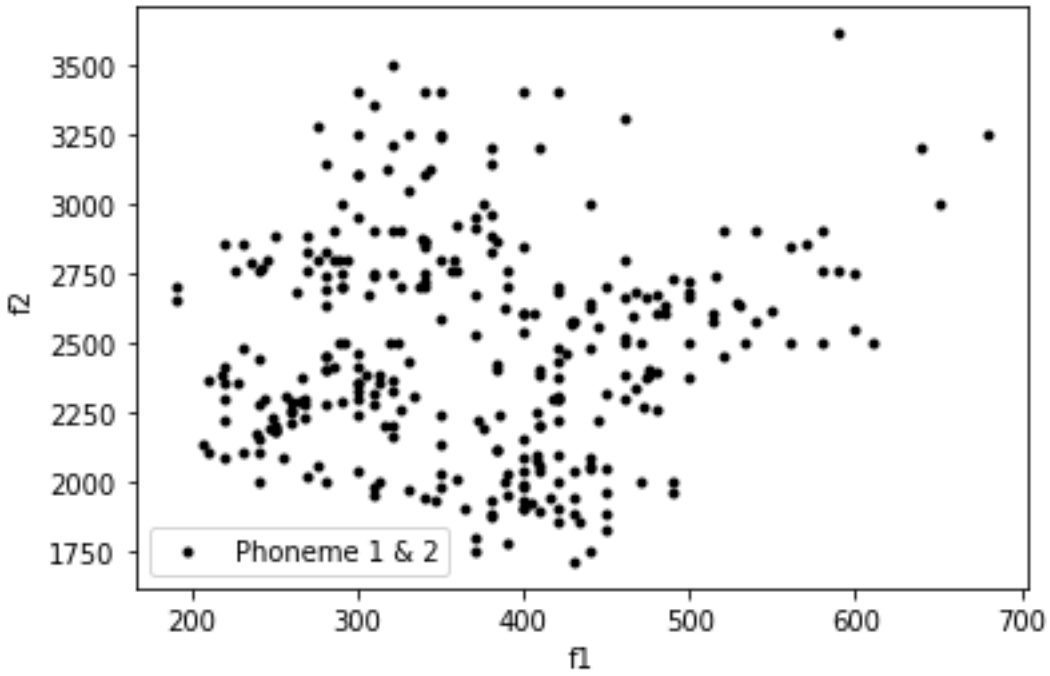
[350.3823347 3038.49672174]
[269.9621774 2283.67262012]
[292.86917397 2777.16837738]

Implemented GMM | Covariances

[[3946.30152021 0.]
 [0. 72974.96999849]]
[[1206.71287556 0.]
 [0. 14141.32436769]]
[[2302.1813159 0.]
 [0. 5296.98680925]]

Implemented GMM | Weights

[0.31670998 0.428734 0.25455602]



Accuracy using GMMs with 3 components: 0.95%

K=6 p_id =1

Implemented GMM | Mean values

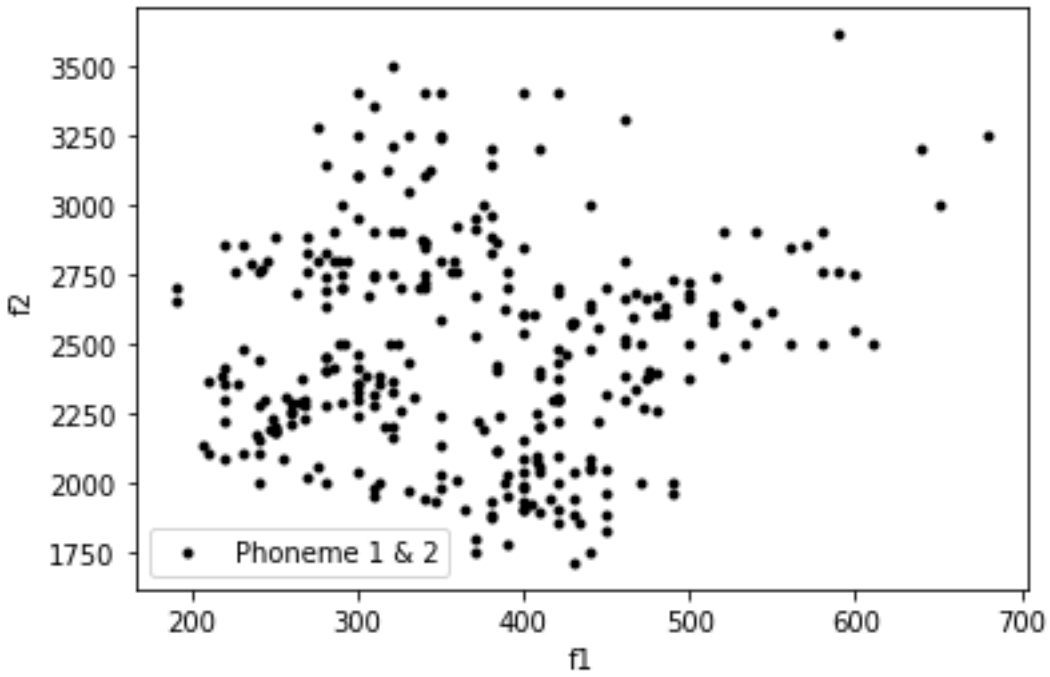
[338.78650135 3255.23630781]
[363.44281952 2824.01732443]
[301.87576415 2836.25310888]
[270.08044282 2282.93768402]
[287.19467353 2770.47167929]
[454.77733876 3077.52556382]

Implemented GMM | Covariances

[[1548.58585723 0.]
[0. 16024.74510593]]
[[437.29105796 0.]
[0. 11229.93876407]]
[[1879.55298985 0.]
[0. 29716.256253]]
[[1215.60010966 0.]
[0. 14002.08248752]]
[[2248.80091319 0.]
[0. 4513.22202615]]
[[6173.0634078 0.]
[0. 125976.45534941]]

Implemented GMM | Weights

[0.12542906 0.08089027 0.14250723 0.42836878 0.1830113 0.03979336]



Accuracy using GMMs with 3 components: 0.96%

K = 6 p_id=2

Implemented GMM | Mean values

[243.65643866 2233.92746036]
[325.34909031 3214.12023325]
[294.32424705 2775.17500298]
[336.06990323 2820.67725289]
[412.91752791 3127.35081345]
[301.50157947 2343.40239176]

Implemented GMM | Covariances

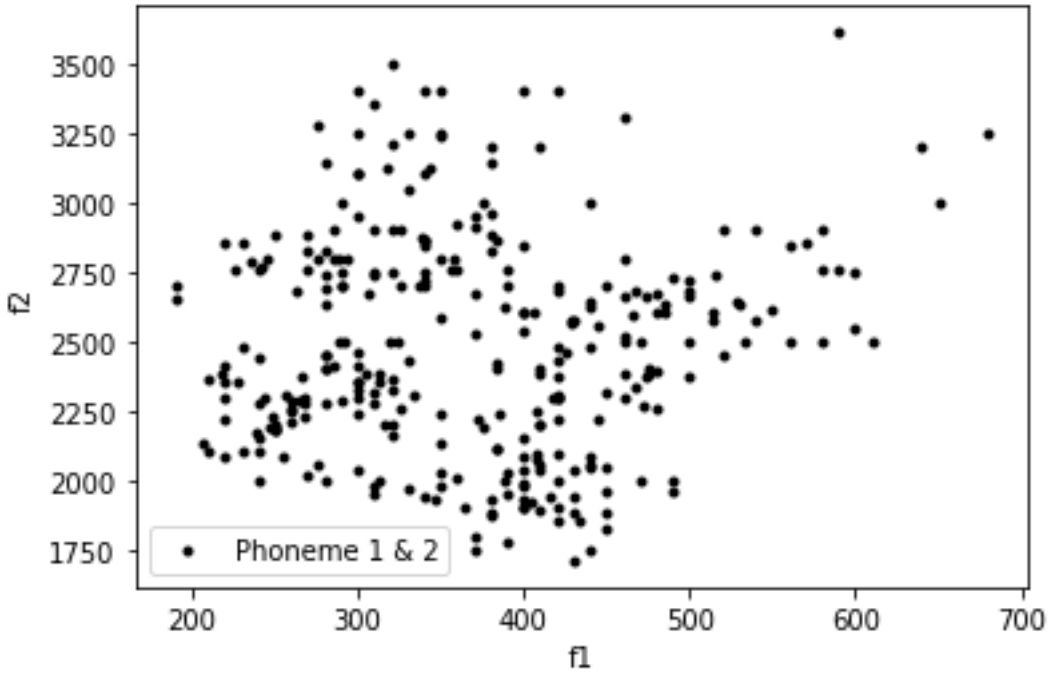
[[353.62748176 0.]
[0. 14226.55720808]]
[[891.53571654 0.]
[0. 19427.17493158]]
[[2750.37729298 0.]
[0. 5836.65731829]]
[[1820.47590519 0.]
[0. 12081.72787218]]
[[6072.94015229 0.]
[0. 91682.62935993]]

[[411.55867884 0.   ]
 [0.   7222.69863352]]

Implemented GMM | Weights

[0.23340917 0.12474645 0.25606509 0.1110994 0.07436652 0.20031336]

Accuracy using GMMs with 3 components: 95%



Task 4: For this Task, you will use the code file task_4.py. Create a grid of points that spans the two datasets (i.e., the dataset that contains phoneme 1, and the dataset that contains phoneme 2). Classify each point in the grid using one of your classifiers (i.e., the MoG that was trained on phoneme 1 and the MoG that was trained on phoneme 2). That is, create a classification matrix, M, whose elements are either 0 or 1. M(i,j) is 0 if the point x=[x1(i), x2(j)] is classified as belonging to phoneme 1, and is 1 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 them, and display the classification matrix. [20 points]

Coding:

```
n1 = np.sum(phoneme_id==1)
n2 = np.sum(phoneme_id==2)
# X_phonemes_1_2 = ...
X_phonemes_1_2 = np.zeros((n1+n2, 2))
j = 0
for i in range(len(phoneme_id)):
    if phoneme_id[i] == 1:
        X_phonemes_1_2[j, 0] = f1[i]
        X_phonemes_1_2[j, 1] = f2[i]
        j = j+1

for i in range(len(phoneme_id)):
    if phoneme_id[i] == 2:
        X_phonemes_1_2[j, 0] = f1[i]
        X_phonemes_1_2[j, 1] = f2[i]
        j = j+1
```

Classification matrix part:

```
M = np.zeros((N_f2, N_f1))
# Create a custom grid of shape N_f1 x N_f2
x = np.linspace(min_f1, max_f1, N_f1)
y = np.linspace(min_f2, max_f2, N_f2)
x1, y1 = np.meshgrid(x, y, sparse = True)
#print(y1)
Z1 = np.zeros((1,k))
Z2 = np.zeros((1,k))
a = np.zeros((1, 2))
for i in range(N_f2):
   for j in range(N_f1):
      a[0, 0] = x1[0, j]
      a[0, 1] = y1[i, 0]
      #print(a)
      Z1 = get_predictions(mu1, s1, p1, a)
      Z2 = get_predictions(mu2, s2, p2, a)
      Z1_new = np.zeros((Z1.shape[0],1))
      Z2_new = np.zeros((Z1.shape[0],1))
      Z1_new = Z1[0, 0] + Z1[0, 1] + Z1[0, 2]
      Z2_new = Z2[0, 0] + Z2[0, 1] + Z2[0, 2]
      if Z1_new > Z2_new:
        M[i, j] = 0
      else:
        M[i, j] = 1
print(M)
```

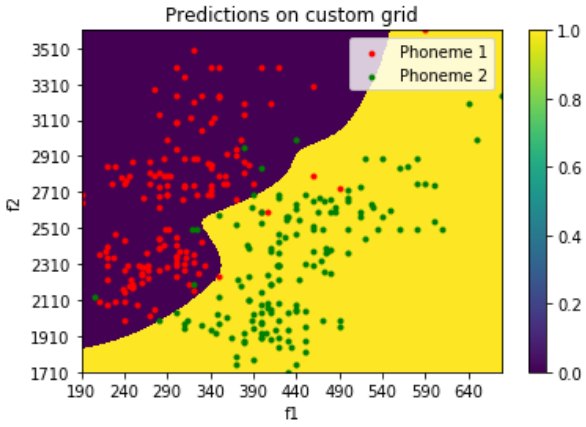
Explanation:

As required , a grid has been computed with datasets containing Phenome 1 and Phoneme 2 Training parameters are obtained to form the X_phoneme_1_2 dataset. By assessing and comparing the Z1 and Z2 values, new point for both phoneme 1 and 2 are obtained.,Each data point is classified by creating a classification matrix, M, whose elements are either 0 or 1. M(i,j) is 0 if the point x=[x1(i), x2(j)] is classified as belonging to phoneme 1, and is 1 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.These forms the basis for classification matrix with point M tends to 0 relates the dimension to Phenome 1. In other case , the M value which is 1 belongs to Phoneme 2.

Illustration:

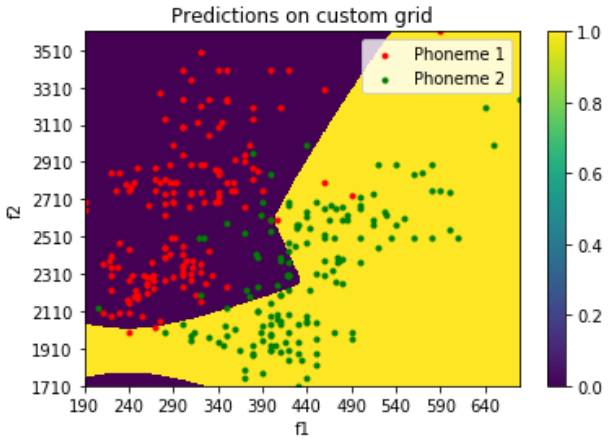
Confusion matrix for K=3

[[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.]]



Confusion matrix K=6

[[0. 0. 0. ... 1. 1. 1.]
 [0. 0. 0. ... 1. 1. 1.]
 [0. 0. 0. ... 1. 1. 1.]
 ...
 [0. 0. 0. ... 1. 1. 1.]
 [0. 0. 0. ... 1. 1. 1.]
 [0. 0. 0. ... 1. 1. 1.]]



Task 5: In the code of task_5.py a MoG with full covariance matrices is fit to the data. Now, create a new dataset that will contain 3 columns, as follows: X = [F1, F2, F1+F2]

Fit a 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 so as to support your observations. [20 points]

Coding part :

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

m = 0
for i in range(len(phoneme_id)):
    if phoneme_id[i] == 1:
        X_phoneme[m, 0] = f1[i]
        X_phoneme[m, 1] = f2[i]
        X_phoneme[m, 2] = f1[i]+f2[i]
        m = m+1
```

```
Implemented GMM | Mean values
[ 270.50623618 2285.47257414 2555.97881032]
[ 351.15641733 3254.86400315 3606.02042048]
[ 314.01869318 2791.42535943 3105.44405261]
Implemented GMM | Covariances
[[ 1217.93519332  1151.15123891  2369.08643223]
 [ 1151.15123891 14340.95882883 15492.11006774]
 [ 2369.08643223 15492.11006774 17861.19649997]]
[[ 4389.24978537  4794.81481702  9184.06460238]
 [ 4794.81481702 21554.68486428 26349.49968129]
 [ 9184.06460238 26349.49968129 35533.56428368]]
[[ 3526.16227619   324.90616501  3851.0684412 ]
 [  324.90616501  9140.16480141  9465.07096642]
 [ 3851.0684412   9465.07096642 13316.13940761]]
```

Implemented GMM | Weights
[0.4348567 0.16603463 0.39910867]

Singularity issue is because of the unexpected nul values leading to the matrix error. It can be resolved by goding to handle such exception and optimization.