Machine Learning

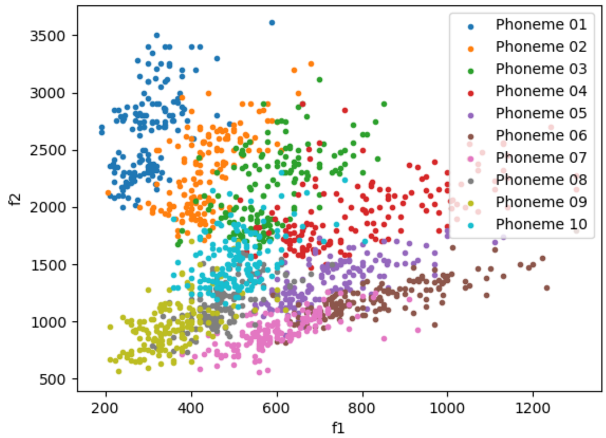
Assignment2

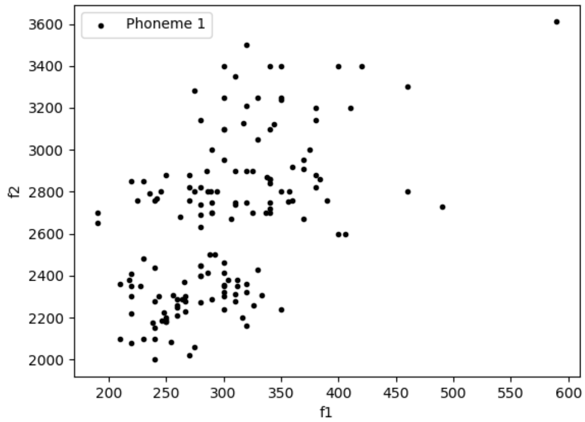
Spyridon Roumpis

181004877

Task1

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.





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

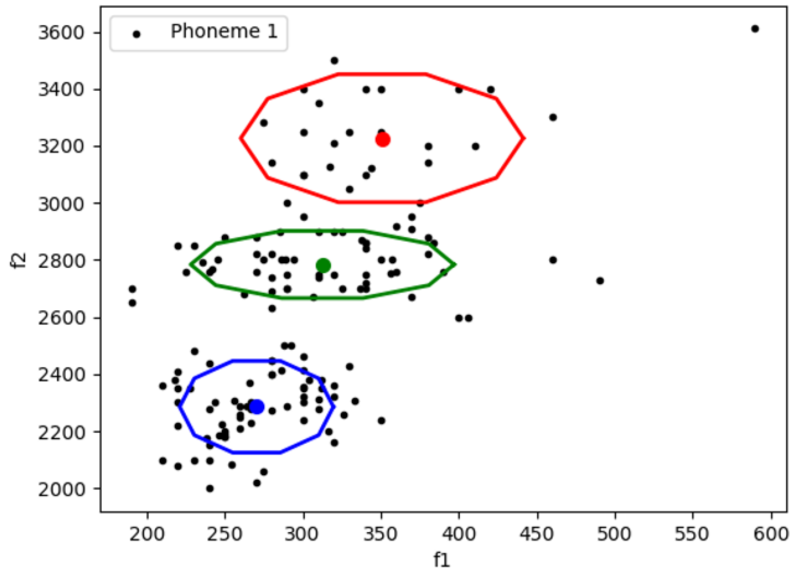
Lines of code added:

```
X_full[:,0] = f1

X_full[:,1] = f2

X_full = X_full.astype(np.float32)

p_id = 1

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

```
s=0
for i in range(len(phoneme_id)):
  if p_id == phoneme_id[i]:
    X_phoneme_1[s] = X_full[i]
    s+=1
```

Task2: For this task experiments were conducted against the k, either 3 or 6 (number of clusters), and the 2 different phonemes. For each experiment we received values of three variables mu, s and p, as it can be seen below. If you run an experiment with the same k and for the same phoneme it can be noticed that clusters each time try to classify a different area.

- K=3, Phoneme1

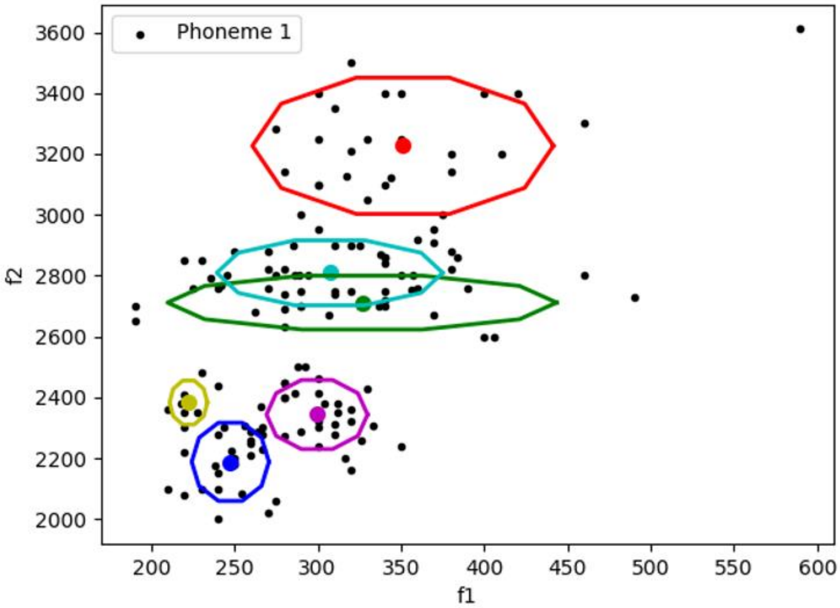


Implemented GMM | Mean values
[350.84461774 3226.3394682]
[312.59125927 2783.8979554]
[270.39520122 2285.46534972]

Implemented GMM | Covariances
[[4102.87584836 0.]
[0. 27829.51330146]]
[[3562.59738744 0.]
[0. 7657.85151852]]
[[1213.73842941 0.]
[0. 14278.42033139]]

Implemented GMM | Weights
[0.18386031 0.38099535 0.43514434]

- K=6, Phoneme1

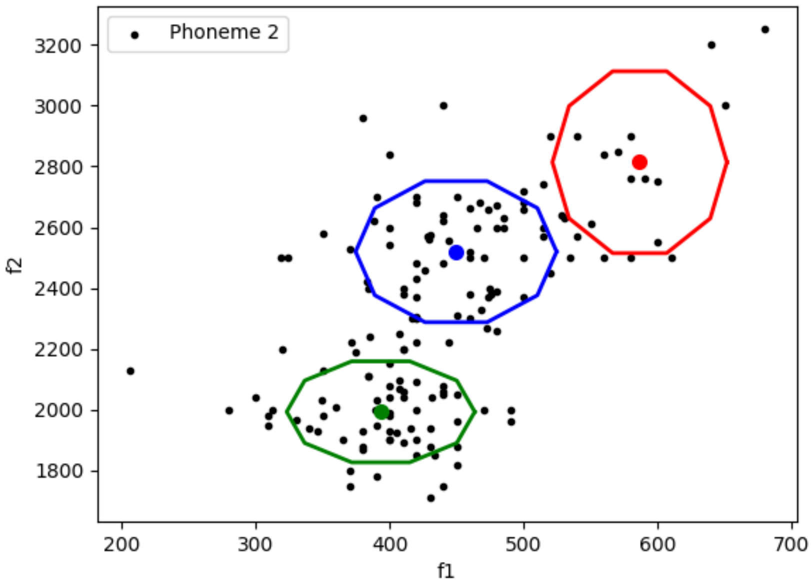


Implemented GMM | Mean values
[351.20341521 3226.59927008]
[326.66983094 2711.3743637]
[247.5268398 2188.11239252]
[307.23337782 2809.2467982]
[299.60643663 2343.43621325]
[222.39958758 2383.04056878]

Implemented GMM | Covariances
[[4086.69543211   0.   ]
 [0.   27718.01980609]]
[[6831.6728401   0.   ]
 [0.   4335.08069879]]
[[269.00479723   0.   ]
 [0.   9115.61041558]]
[[2302.66099777   0.   ]
 [0.   6291.93564771]]
[[460.38019342   0.   ]
 [0.   7128.39127112]]
[[64.45305383   0.   ]
 [0.   2874.65903362]]

Implemented GMM | Weights
[0.18377498 0.09980135 0.17603543 0.28216767 0.21274925 0.04547133]

- K=3,Phoneme2

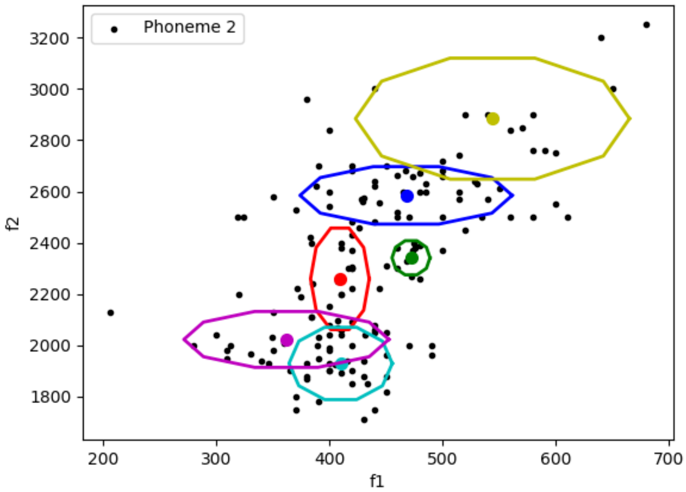


Implemented GMM | Mean values
[586.4428169 2813.71912273]
[393.44719849 1993.02525415]
[449.61704797 2519.54556884]

Implemented GMM | Covariances
[[2115.0537852   0.   ]
 [0.   49424.55456626]]
[[2455.20886927   0.   ]
 [0.   15252.65536239]]
[[2805.1702886   0.   ]
 [0.   29733.64154256]]

Implemented GMM | Weights
[0.09513792 0.43505543 0.46980665]

- K=6,Phoneme2



Implemented GMM | Mean values
[409.3214985 2259.19455689]
[472.07745624 2341.22236006]
[467.74948865 2584.71607324]
[409.96916698 1929.17991988]
[362.04983235 2022.94277401]
[543.98913383 2884.02180421]

Implemented GMM | Covariances
[[335.57839682   0.   ]
 [0.   21671.02299537]]
[[140.7701278   0.   ]
 [0.   2452.37008909]]
[[4382.09472757   0.   ]
 [0.   6932.71844033]]
[[1038.87705745   0.   ]
 [0.   10995.21979661]]
[[4090.22227866   0.   ]
 [0.   6569.73158666]]
[[7313.56902033   0.   ]
 [0.   30799.9274949]]

Implemented GMM | Weights
[0.16052829 0.06772328 0.29930962 0.22329562 0.14597382 0.10316937]

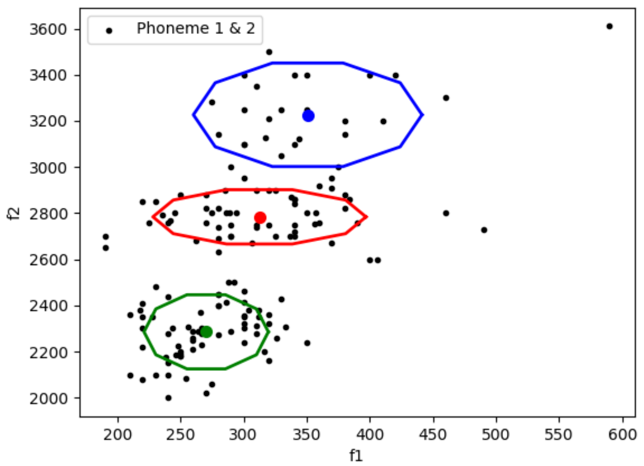
Code:

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

Task3:

- K=3, Phoneme1

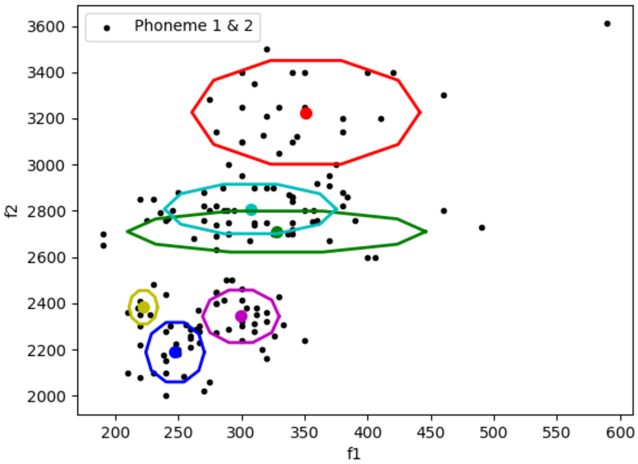


Implemented GMM | Mean values
[312.591254 2783.8979323]
[270.39520125 2285.46535   ]
[350.84461358 3226.33934189]

Implemented GMM | Covariances
[[3562.59746935   0.   ]
 [0.   7657.84838142]]
[[1213.73842838   0.   ]
 [0.   14278.4203668]]
[[4102.87521394   0.   ]
 [0.   27829.54582587]]

Implemented GMM | Weights
[0.38099527 0.43514434 0.18386039]

- K=6, Phoneme1



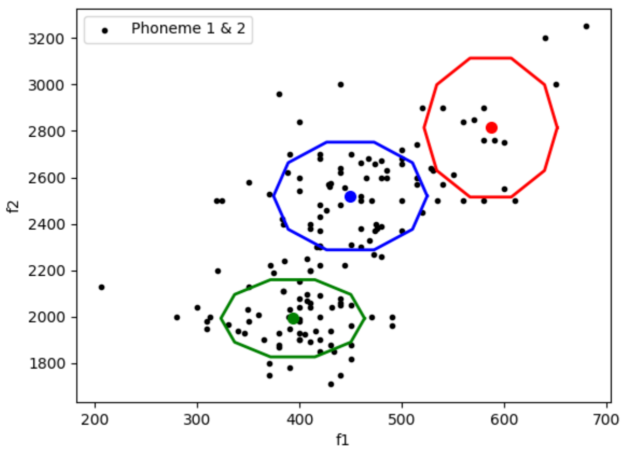
Implemented GMM | Mean values
[351.1946017 3226.21136353]
[327.86573503 2710.17978864]
[247.53325243 2188.31032938]
[307.12861812 2807.92691074]
[299.62903772 2343.46086778]
[222.37362108 2382.97188502]

Implemented GMM | Covariances
[[4084.46443826   0.   ]
 [0.   27811.36083645]]
[[6988.03134386   0.   ]
 [0.   4349.8215423]]
[[269.04740414   0.   ]
 [0.   9143.1346928]]
[[2315.13641256   0.   ]
 [0.   6324.85087285]]
[[459.80556686   0.   ]
 [0.   7129.36513689]]
[[64.1117639   0.   ]
 [0.   2873.20166811]]

Implemented GMM | Weights

[0.18400685 0.09505051 0.17625611 0.28669166 0.21261195 0.04538292]

- K=3, Phoneme2

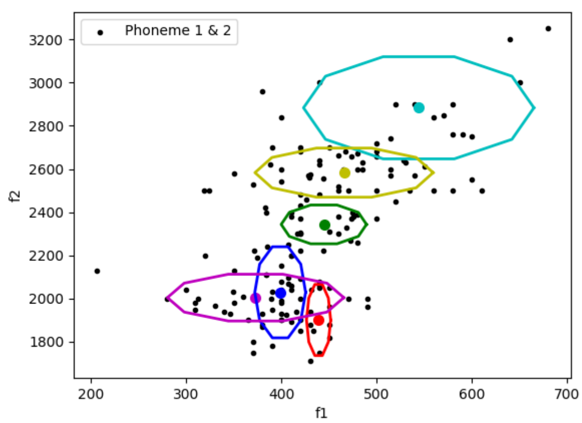


Implemented GMM | Mean values
[586.56934154 2814.20371074]
[393.45316806 1993.08546353]
[449.67901526 2519.69348575]

Implemented GMM | Covariances
[[2111.35218973   0.   ]
 [0.   49424.39499203]]
[[2454.89411599   0.   ]
 [0.   15264.26327731]]
[[2809.54160203   0.   ]
 [0.   29719.63444455]]

Implemented GMM | Weights
[0.09486661 0.43517295 0.46996044]

- K=6, Phoneme2



Implemented GMM | Mean values
[439.10465767 1900.53443187]
[444.61796046 2343.28968272]
[398.86961351 2028.77878774]
[544.09783587 2883.354677]

```
[ 373.02525366 2004.18865408]
[ 465.80830614 2582.99677477]
```

Implemented GMM | Covariances

```
[[   80.38670659     0.        ]
 [    0.         15000.93656249]]
[[ 999.89873236    0.        ]
 [   0.        4459.78016831]]
[[  355.15427403     0.        ]
 [    0.         24589.83553739]]
[[ 7297.70140352     0.        ]
 [    0.         30925.49256272]]
[[4277.91245041    0.        ]
 [   0.        6482.59409341]]
[[4364.13864253    0.        ]
 [   0.        7148.23274649]]
```

Implemented GMM | Weights

```
[0.0566072  0.13653511 0.21195421 0.10339651 0.18444121 0.30706576]
```

For the purpose of this task we get the new predictions and the new Implemented GMM | Weights, Implemented GMM | Covariances, Implemented GMM | Mean values based on our data from task 2. Also the code that's being used is taken from task 2 .

Code:

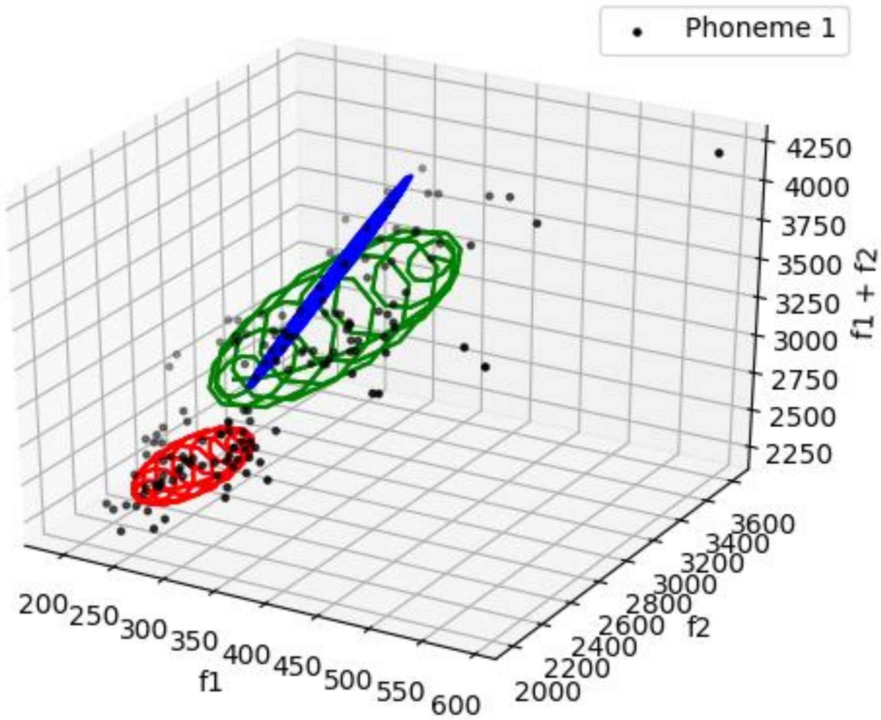
```
X_full[:,0] = f1
X_full[:,1] = f2

X_full = X_full.astype(np.float32)

# number of GMM components
k = 6
p_id = 2

X_phonemes_1_2 = np.zeros((np.sum(phoneme_id==1), 2))
s=0
for i in range(len(phoneme_id)):
    if p_id == phoneme_id[i]:
        X_phonemes_1_2[s] = X_full[i]
        s+=1

GMM_params_phoneme =
np.load('GMM_params_phoneme_02_k_06.npy',allow_pickle=True)
GMM_params_phoneme = np.ndarray.tolist(GMM_params_phoneme)
```

```
mu=GMM_params_phoneme['mu']
s=GMM_params_phoneme['s']
p=GMM_params_phoneme['p']

X = X_phonemes_1_2.copy()
# get number of samples
N = X.shape[0]
# get dimensionality of our dataset
D = X.shape[1]


n_iter = 100

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_phonemes_1_2, title_string=title_string, ax=ax1)
   # Plot gaussians after each iteration
   plot_gaussians(ax1, 2*s, mu)
print('\nFinished.\n')

print('Implemented GMM | Mean values')
for i in range(k):
   print(mu[i])
print('')
print('Implemented GMM | Covariances')
for i in range(k):
   print(s[i,:,:])
```

```
print('')
print('Implemented GMM | Weights')
print(p)
print('')
```

Task5

- K=3, Phoneme1



Implemented GMM | Mean values

[270.57551315 2271.35385223 2541.92936538]

[321.86495121 2881.09697268 3202.96192388]

[295.85962895 3028.71185984 3324.57148879]

Implemented GMM | Covariances

[[1189.92551957 1270.88408839 2460.80960796]

 [1270.88408839 13124.32164347 14395.20573186]

 [2460.80960796 14395.20573186 16856.01533982]]

```
[[ 4228.1879231   7172.63910358 11400.82702668]
 [ 7172.63910358 70848.76686439 78021.40596796]
 [11400.82702668 78021.40596796 89422.23299464]]
[[   169.47103182   3756.27873931   3925.74977112]
 [  3756.27873931 101049.35919834 104805.63793765]
 [  3925.74977112 104805.63793765 108731.38770877]]
Implemented GMM | Weights
[0.38815904 0.58517972 0.02666124]
```

Code:

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

# id of the phoneme that will be used (e.g. 1, or 2)
p_id = 1
# number of GMM components
k = 3
#########################################
X_phoneme = np.zeros((np.sum(phoneme_id==1), 3))
s=0
for i in range(len(phoneme_id)):
    if p_id == phoneme_id[i]:
        X_phoneme[s] = X_full[i]
        s+=1
```

The singularity problem is occurring when there is only one point and so the variance is zero, which in the multi-variate Gaussian case, leads to a singular covariance matrix. When the variance gets to zero the likelihood goes to infinity and as a result of this our model is overfitting.