Neeraj Vashistha
190573735

EECS Machine Learning

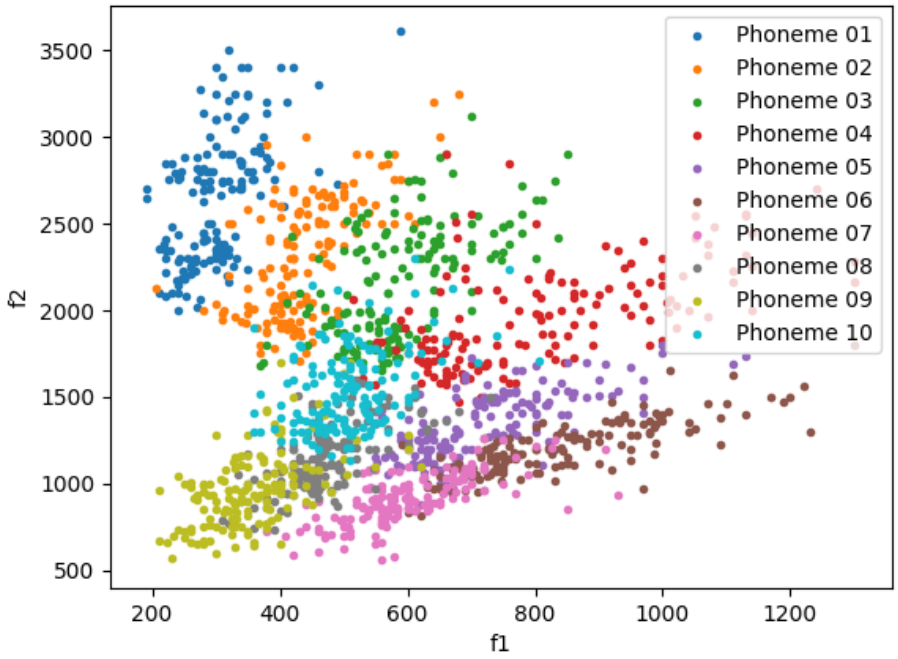
Assignment 2

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.).

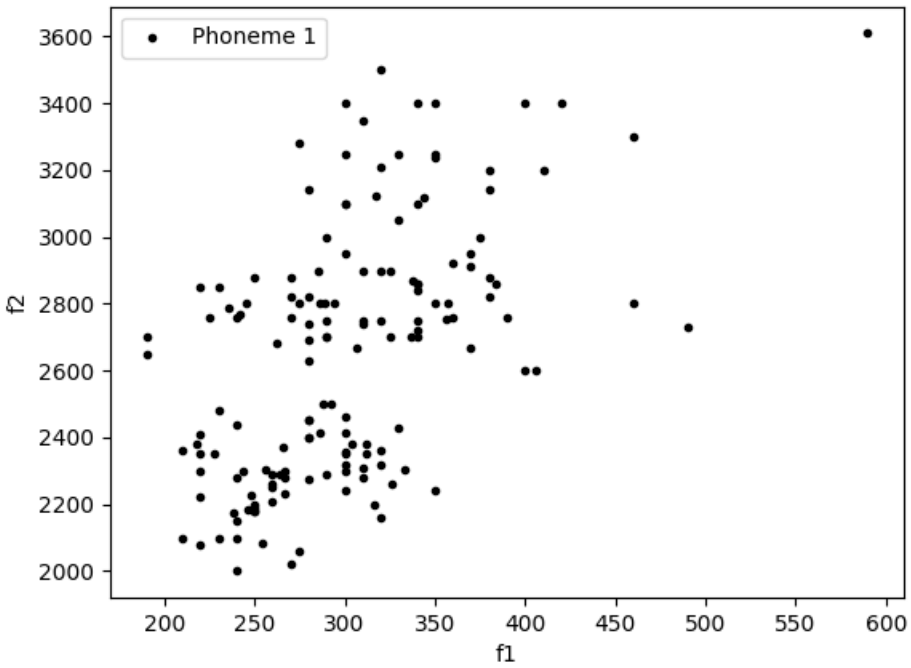
Include in your report the corresponding lines of your code and the plot.

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




```
# Create array containing only samples that belong to phoneme 1
X_phoneme_1 = np.zeros((np.sum(phoneme_id==1), 2))
# print(len(X_phoneme_1))
X_phoneme_1 = X_full[phoneme_id==p_id,:]
# print(X_full[phoneme_id==p_id,:])
```

Neeraj Vashistha
190573735



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:

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).

Generate a dataset stored in the variable “X_phoneme_1”, that contains only the F1 and F2 forthe first phoneme.

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.

Run task_2.py on the dataset using K=6

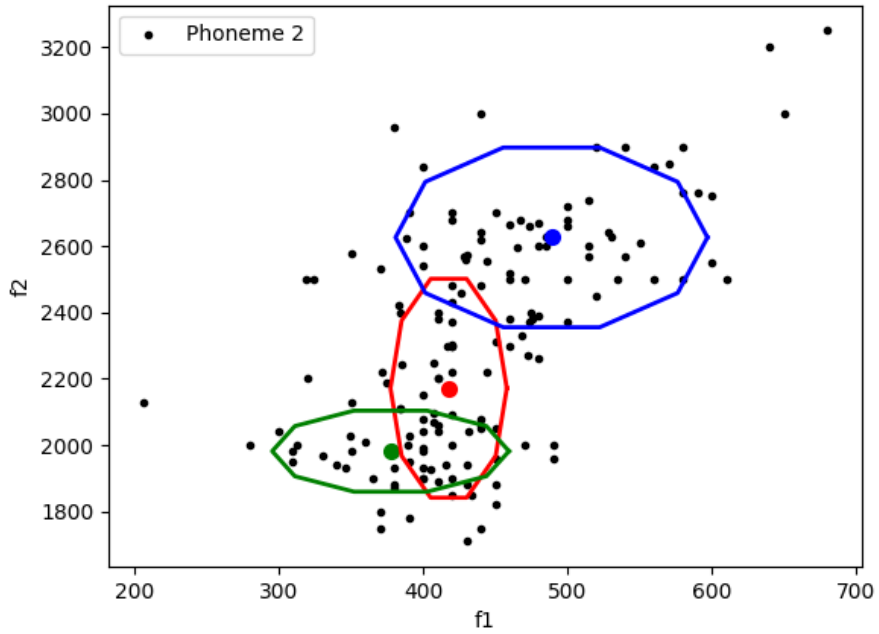
Repeat steps 2-4 for the second phoneme

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

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

X_phoneme = X_full[phoneme_id==p_id,:]
K=3 phoneme = 2
```

Neeraj Vashistha
190573735



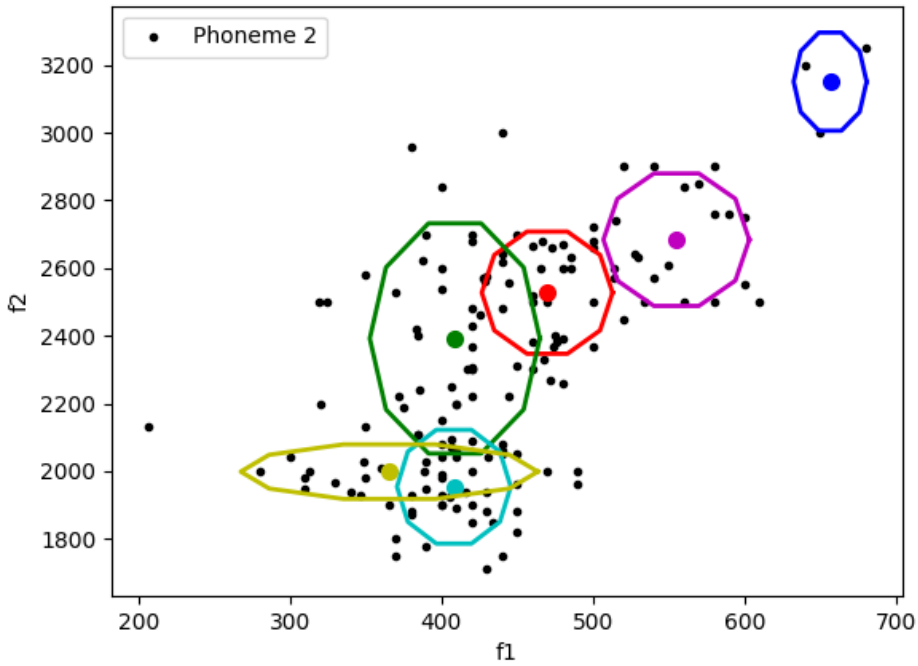

```
Implemented GMM | Mean values
[ 417.69122 2171.6367 ]
[ 377.5102 1981.9363]
[ 488.86087 2626.4846 ]

Implemented GMM | Covariances
[[  810.23843575     0.        ]
[    0.         60194.75964545]]
[[3356.75091241    0.        ]
[   0.         8259.18087656]]
[[ 5825.77784373     0.        ]
[    0.         40611.43172655]]

Implemented GMM | Weights
[0.34403585 0.23510038 0.42086377]
```

K=6 phoneme = 2

Neeraj Vashistha
190573735



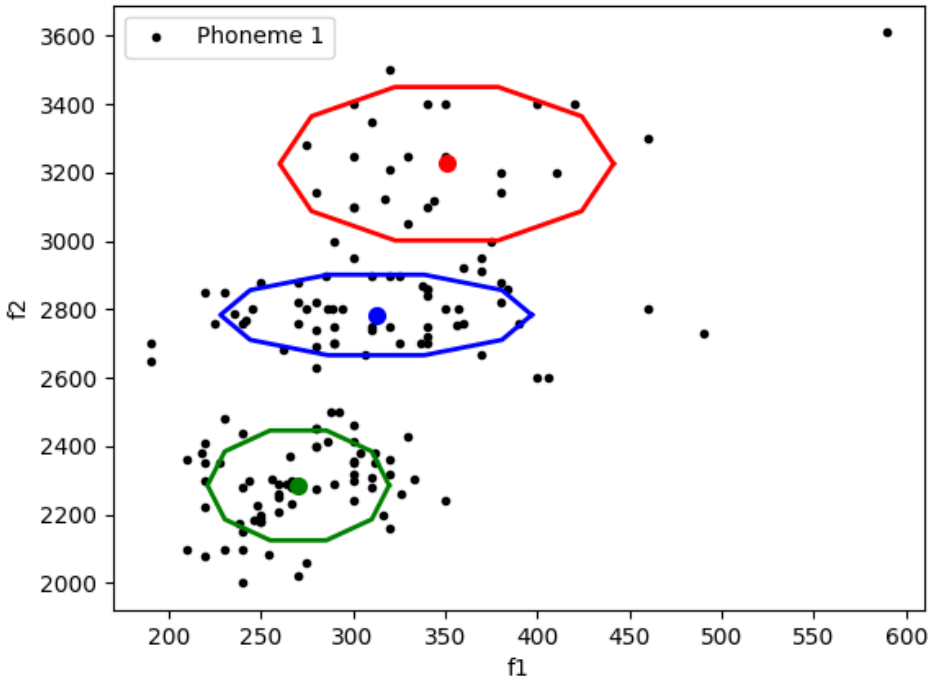

```
Implemented GMM | Mean values
[ 469.86063 2527.2544 ]
[ 408.8784 2392.1135]
[ 656.6973 3150.6448]
[ 408.2564 1954.1306]
[ 555.1772 2683.716 ]
[ 365.60007 1998.5745 ]

Implemented GMM | Covariances
[[  932.43882736     0.        ]
[    0.         17975.71002635]]
[[ 1576.3784046      0.        ]
[    0.         63880.38007491]]
[[  289.96457955     0.        ]
[    0.         11620.5571611 ]]
[[  705.28324742     0.        ]
[    0.         15600.2947933 ]]
[[ 1159.62076083     0.        ]
[    0.         21140.44309384]]
[[4807.82190949    0.        ]
[   0.         3586.81186569]]

Implemented GMM | Weights
[0.19756236 0.29787194 0.01964919 0.22699474 0.12504335 0.13287842]
```

K=3 phoneme = 1

Neeraj Vashistha
190573735



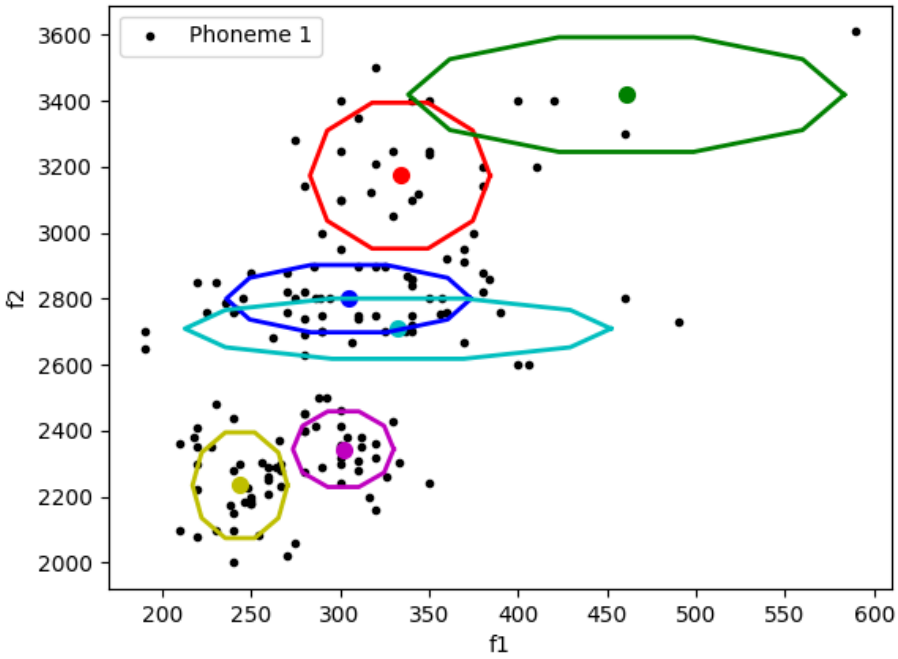

```
Implemented GMM | Mean values
[ 350.8446 3226.339 ]
[ 270.3952 2285.4653]
[ 312.59125 2783.898  ]

Implemented GMM | Covariances
[[ 4102.87457976     0.         ]
[    0.         27829.5891043 ]]
[[ 1213.73843377     0.         ]
[    0.         14278.42070277]]
[[3562.59750605    0.         ]
[   0.         7657.84496186]]

Implemented GMM | Weights
[0.18386048 0.43514434 0.38099518]
```

K=6 phoneme = 1

Neeraj Vashistha
190573735




```
Implemented GMM | Mean values
[ 333.7469 3173.4844]
[ 460.8253 3419.2847]
[ 304.88425 2800.4736 ]
[ 332.5451 2709.5947]
[ 301.87903 2344.0007 ]
[ 243.85162 2234.5955 ]

Implemented GMM | Covariances
[[ 1279.79506622     0.        ]
[    0.         27050.37231273]]
[[ 7499.99782834     0.        ]
[    0.         16701.5888991 ]]
[[2363.94787867    0.        ]
[   0.         5769.65915574]]
[[7166.83319288    0.        ]
[   0.         4620.64129664]]
[[ 401.29360816    0.        ]
[   0.         7257.0601022 ]]
[[  356.18880996     0.        ]
[    0.         14197.23859805]]

Implemented GMM | Weights
[0.17294541 0.025002   0.27878276 0.08896935 0.19876708 0.23553341]
```

Task 3:

Neeraj Vashistha
190573735

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; θ 1), where θ 1 are the parameters of the MoG learnt for the first phoneme, with p(x; θ 2), where θ 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

```
X_phonemes_1_2 = np.zeros((np.sum(phoneme_id==2)+np.sum(phoneme_id==1), 2))
X_phonemes_1_2 = np.concatenate((X_full[phoneme_id==1,:],X_full[phoneme_id==2,:]), axis=0)
# # X_phonemes_1_2 =
y_true =
np.concatenate((np.zeros((np.sum(phoneme_id==1))),np.ones((np.sum(phoneme_id==2)))),axis =0)
def load_data(k):
    # File that contain trained model
    data_npy_phoneme_01 = 'data/GMM_params_phoneme_01_k_0'+str(k)+'.npy'
    data_npy_phoneme_02 = 'data/GMM_params_phoneme_02_k_0'+str(k)+'.npy'


    # Loading data from .npy file
    model_phoneme_01 = np.ndarray.tolist(np.load(data_npy_phoneme_01, allow_pickle=True))
    model_phoneme_02 = np.ndarray.tolist(np.load(data_npy_phoneme_02, allow_pickle=True))
    return model_phoneme_01, model_phoneme_02


def get_pred(X, k, model1_weights, model2_weights):
    predClass = []
    N = X.shape[0]
    Z01 = np.zeros((N,k)) # shape Nxk
    Z02 = np.zeros((N,k)) # shape Nxk

    # get predictions on X from model1
    Z01 = get_predictions(model1_weights['mu'], model1_weights['s'], model1_weights['p'], X)
    # Z01 = normalize(Z01, axis=1, norm='l1')
    Z01 = Z01.astype(np.float32)
    Z01Sum = np.sum(Z01,axis=1)

    # get predictions on X from model2
    Z02 = get_predictions(model2_weights['mu'], model2_weights['s'], model2_weights['p'], X)
    # Z02 = normalize(Z02, axis=1, norm='l1')
    Z02 = Z02.astype(np.float32)
    Z02Sum = np.sum(Z02,axis=1)

    # if sum of probabilities of any model is less than other mark it
    for z1,z2 in zip(Z01Sum,Z02Sum):
        if z1 > z2:
            predClass.append(0.0)
        else:
```

```
            predClass.append(1.0)

    y_pred = np.array(predClass)
    return y_pred

model_phoneme_01, model_phoneme_02 = load_data(k)
y_pred = get_pred(X, k, model_phoneme_01, model_phoneme_02)

# print(y_pred)
accuracy = accuracy_score(y_true,y_pred)
#########################
Output
#########################
              precision    recall  f1-score   support

   phoneme 1       0.94      0.96      0.95       152
   phoneme 2       0.96      0.94      0.95       152

    accuracy                           0.95       304
   macro avg       0.95      0.95      0.95       304
weighted avg       0.95      0.95      0.95       304

confusion matrix:
[[146   6]
[  9 143]]
Accuracy using GMMs with 3 components: 95.07%
Mis-classification error using GMMs with 3 components: 4.93%
#########################
Output
#########################
              precision    recall  f1-score   support

   phoneme 1       0.95      0.96      0.96       152
   phoneme 2       0.96      0.95      0.96       152

    accuracy                           0.96       304
   macro avg       0.96      0.96      0.96       304
weighted avg       0.96      0.96      0.96       304

confusion matrix:
[[146   6]
[  7 145]]
Accuracy using GMMs with 6 components: 95.72%
Mis-classification error using GMMs with 6 components: 4.28%
```

Form the above we can see that, misclassification error rates are almost identical for both 3 components and 6 components approximately 4%, also we conclude that both the models have high accuracy of around 95%. This means that, if we have to discriminate between phonemes 1 and 2 and compare both of these models, then both the models are able to classify similar data-points in their respective clusters and outside the clusters. The sum of probabilities for any data-point to lie in one of the clusters for a model is compared with the other, if the model is able to successfully cluster then it would return 1 else a value less than 1. For points which are very far away such as outliers are not classified into any clusters

for such data-points we see probabilities less than 1, and from above mis-classification error rate we confirm that there are around 4% of such points.

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.

2Display the classification matrix.

Include the lines of code in your report, comment them, and display the classification matrix.

```
# Create a custom grid of shape N_f1 x N_f2
custom_grid = np.zeros((N_f1,N_f2,2))
# The grid will span all the values of (f1, f2) pairs, between [min_f1, max_f1] on f1 axis, and between [min_f2, max_f2] on f2 axis
# Then, classify each point [i.e., each (f1, f2) pair] of that grid, to either phoneme 1, or phoneme 2, using the two trained GMMs
for i in range(N_f1):
    for j in range(N_f2):
        custom_grid[i][j] = [int(i + min_f1),(j + min_f2)]

# Do predictions, using GMM trained on phoneme 1, on custom grid
# Do predictions, using GMM trained on phoneme 2, on custom grid


def get_predicted_value(k,gird_val):
    # Loading data for phoneme 1
    path_01 = 'data/GMM_params_phoneme_01_k_0' + str(k) + '.npy'
    data_phoneme_01 = np.load(path_01, allow_pickle=True)
    data_phoneme_01 = np.ndarray.tolist(data_phoneme_01)
    # Loading data for phoneme 2
    path_02 = 'data/GMM_params_phoneme_02_k_0' + str(k) + '.npy'
    data_phoneme_02 = np.load(path_02, allow_pickle=True)
    data_phoneme_02 = np.ndarray.tolist(data_phoneme_02)

    Z_phoneme_01 = get_predictions(data_phoneme_01['mu'], data_phoneme_01['s'], data_phoneme_01['p'], gird_val)
    Z_phoneme_01_sum = np.sum(Z_phoneme_01,axis=1)
    Z_phoneme_02 = get_predictions(data_phoneme_02['mu'], data_phoneme_02['s'], data_phoneme_02['p'], gird_val)
    Z_phoneme_02_sum = np.sum(Z_phoneme_02,axis=1)
```

```
    predClass = []
    # if sum of probabilities of any model is less than other mark it
    for z1,z2 in zip(Z_phoneme_01_sum,Z_phoneme_02_sum):
        if z1 > z2:
            predClass.append(0.0)
        else:
            predClass.append(1.0)

    y_pred = np.array(predClass)
    return y_pred
# Compare these predictions, to classify each point of the grid
# Store these prediction in a 2D numpy array named "M", of shape N_f2 x N_f1 (the first
dimension is f2 so that we keep f2 in the vertical axis of the plot)
M = np.ndarray(shape=(N_f2, N_f1))
for i in range(0,N_f2):
    M[i,:] = get_predicted_value(3, custom_grid[:,i])
# M should contain "0.0" in the points that belong to phoneme 1 and "1.0" in the points that
belong to phoneme 2
```