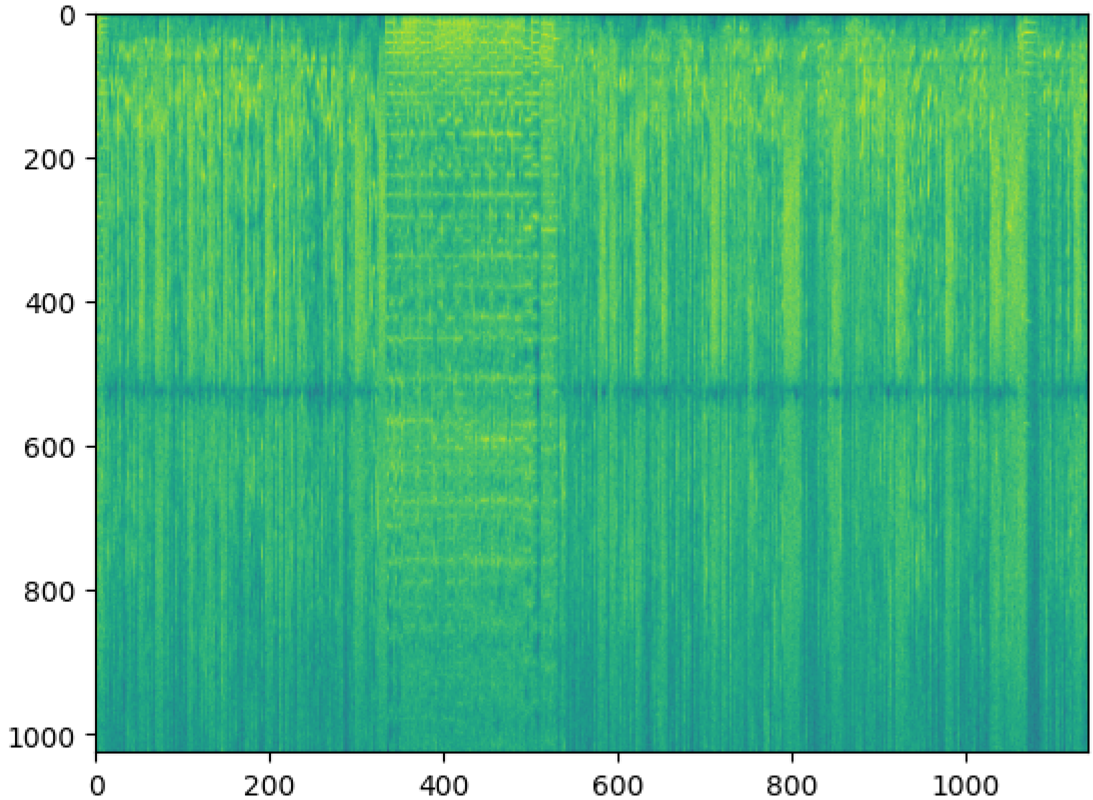
Problem 1

I Load the data in and take the stft. I then remove any columns of the spectrogram where the sum of the magnitude of its bins is less than 20000 based on what i observed. This gets rid of parts of audio where its silent that can mess with our clustering

In [ ]:
```
from scipy.io import wavfile
from scipy.signal import stft
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from scipy.ndimage import median_filter
from sklearn.mixture import GaussianMixture
from hmmlearn import hmm


np.random.seed(0)

file_path = 'friends.wav'
sr, audio_data = wavfile.read(file_path)

summedPowerThreshold = 20000
n_fft = 2048
overlap = 0
f, t, Zxx = stft(audio_data, fs=sr, nperseg=n_fft, noverlap=overlap)
summedPower = np.sum(np.abs(Zxx),axis=0) #incase i need later
filteredforPower = Zxx[:, summedPower > summedPowerThreshold]
log_spec = 10 * np.log10(np.abs(filteredforPower))
plt.imshow(log_spec, aspect='auto')

print(Zxx.shape,"<---Initial spec")
print(log_spec.shape, "<--Removed Low Energy columns")
```

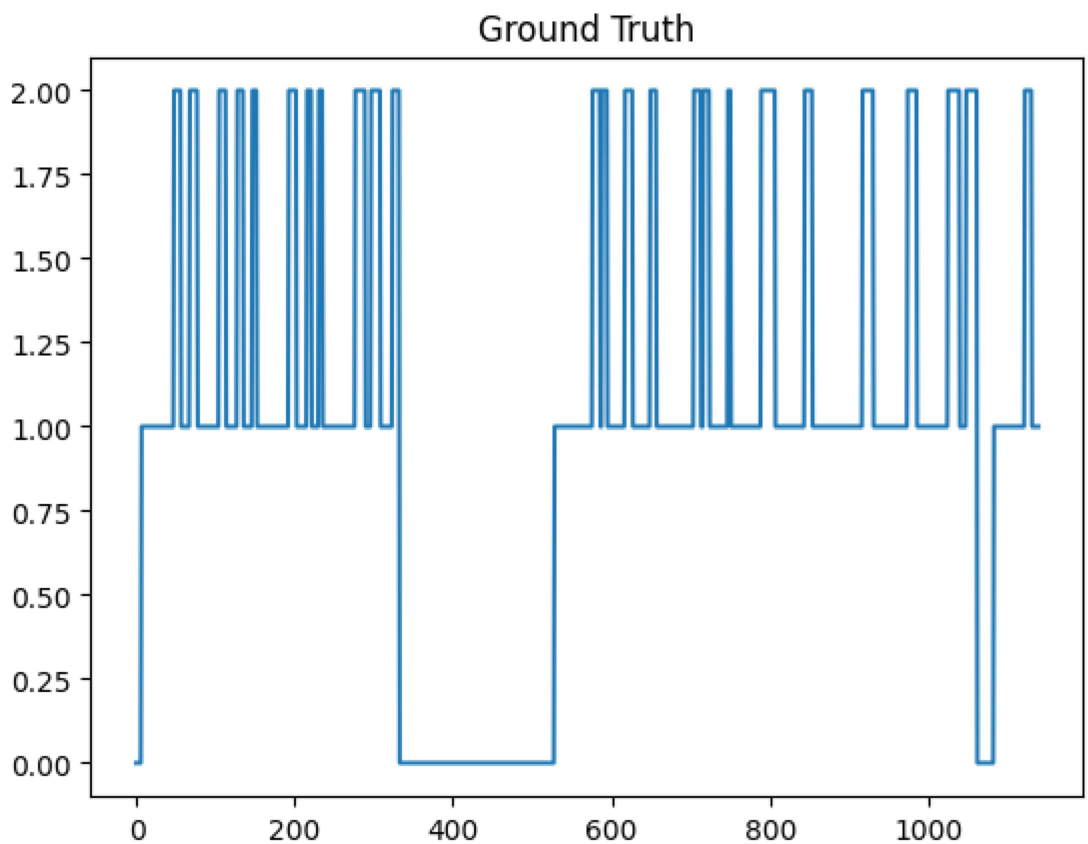
```
(1025, 1168) <---Initial spec
(1025, 1140) <--Removed Low Energy columns
```



Get labels for entire spectrogram then remove labels for the relatively silent bins that were in the spectrogram

In [ ]:
```
## Getting labels

labels_file_path = 'friendsLabels.txt'
with open(labels_file_path, 'r') as file:
    labels_data = file.readlines()

N = Zxx.shape[1]
MaxTime = 298.711700
fullLabels = np.full(N, -2, dtype=int)

for line in labels_data:
    start_time, end_time, label = line.strip().split('\t')
    start_idx = int(float(start_time)/298.711700 * N)
    end_idx =  int(float(end_time)/298.711700 * N)
    if label == 'music':
        fullLabels[start_idx:end_idx] = 0
    elif label == 'speech':
        fullLabels[start_idx:end_idx] = 1
    elif label == 'laughter':
        fullLabels[start_idx:end_idx] = 2

groundTruthLabels = fullLabels[summedPower > summedPowerThreshold]
plt.plot(groundTruthLabels)
plt.title('Ground Truth')
```

Out[ ]:
```
Text(0.5, 1.0, 'Ground Truth')
```



In [ ]:
```
def pca(data): #takes input data as columns and expects it to be 0 mean (add mean back
    covariance_matrix = np.dot(data,data.T)/(data.shape[1])
    eigvals, eigvecs = np.linalg.eigh(covariance_matrix)
    eigvals = eigvals[::-1]
    eigvecs = eigvecs[:,::-1]

    return (eigvecs,eigvals)

def pcaReduce(data,n_components): #return eigvals and reduced data, and mean it subtra
    mean = np.mean(data,axis=1).reshape(-1,1)
    vecs, vals = pca(data - mean)
    reduced = vecs.T[:n_components] @ (data - mean)
    return reduced,vals,mean
```

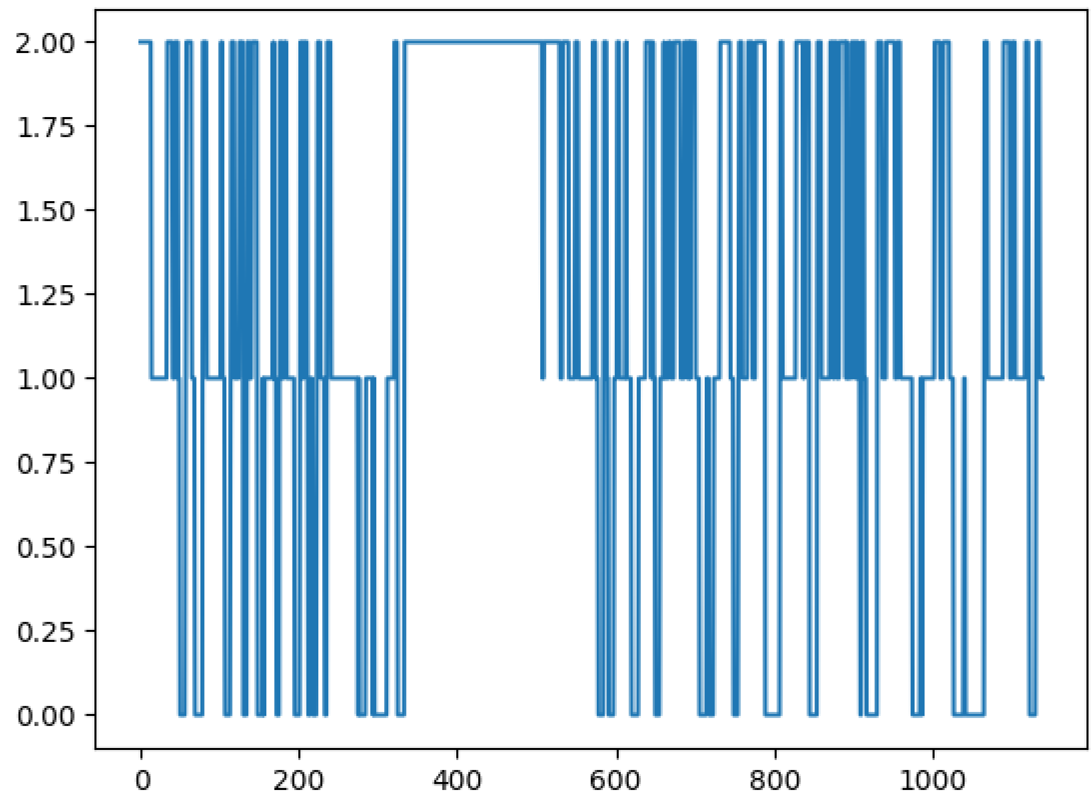
In [ ]:
```
dataP1,eigvalsP1, meanP1 = pcaReduce(log_spec,4) #get reduced data
```

Kmean

In [ ]:
```
n_clusters = 3

kmeans = KMeans(n_clusters=n_clusters, n_init=1)
kmeans.fit(dataP1.T)

# Get cluster labels and centroids
labels_knn = kmeans.labels_
centroids = kmeans.cluster_centers_

filtered_labels_knn = median_filter(labels_knn, size=4, mode='reflect')

plt.plot(filtered_labels_knn)
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x1ff1d560040>]
```



GMM

In [ ]:

```
gmm = GaussianMixture(n_components=n_clusters,n_init=1)
gmm.fit(dataP1.T)


labels_gmm = gmm.predict(dataP1.T)

filtered_labels_gmm = median_filter(labels_gmm, size=4, mode='reflect')

# plt.plot(filtered_labels_gmm)
```

HMM

In [ ]:

```
model = hmm.GaussianHMM(n_components=3, covariance_type="full")


model.startprob_ = np.array([1/3, 1/3, 1/3])
# model.transmat_ = np.array([[.9, 0.05, .05],
#                              [0.05, 0.9, 0.05],
#                              [0.05, 0.05, .9]])
# model.transmat_ = np.array([[.95, 0.025, .025],
#                              [0.025, 0.95, 0.025],
#                              [0.025, 0.025, .95]])
model.transmat_ = np.array([[.99, 0.005, .005],
                            [0.005, 0.99, 0.005],
                            [0.005, 0.005, .99]])

model.means_ = gmm.means_
```

```
model.covars_ = gmm.covariances_

logprob, labels_hmm = model.decode(dataP1.T)
print("Log Probability of the Most Likely Path:", logprob)
```

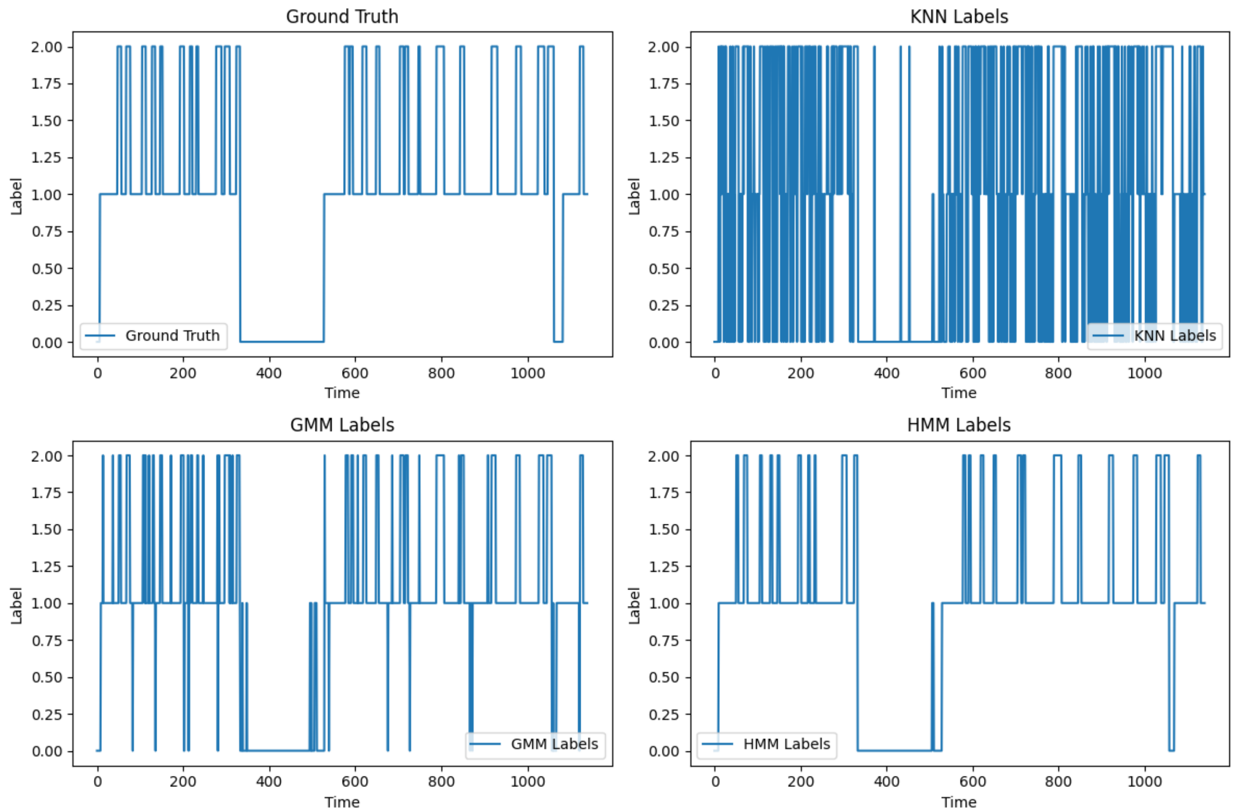
```
Log Probability of the Most Likely Path: -22523.809011804762
```

RESULTS

In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(12, 8))

plt.subplot(221)
plt.plot(groundTruthLabels, label='Ground Truth')
plt.title('Ground Truth')
plt.xlabel('Time')
plt.ylabel('Label')
plt.legend()

plt.subplot(222) ### I know for this random seed that I need to arange the labels as s
rearrangedKNNlabels = np.zeros_like(labels_knn)
rearrangedKNNlabels[labels_knn == 2] = 0
rearrangedKNNlabels[labels_knn == 1] = 1
rearrangedKNNlabels[labels_knn == 0] = 2
print("KNNaccuracy: ", np.sum(rearrangedKNNlabels == groundTruthLabels)/groundTruthLab

plt.plot( rearrangedKNNlabels, label='KNN Labels')
plt.title('KNN Labels')
plt.xlabel('Time')
plt.ylabel('Label')
plt.legend()

print("GMMaccuracy: ", np.sum(labels_gmm == groundTruthLabels)/groundTruthLabels.size)

plt.subplot(223)
plt.plot( labels_gmm, label='GMM Labels')
plt.title('GMM Labels')
plt.xlabel('Time')
plt.ylabel('Label')
plt.legend()

print("HMMaccuracy: ", np.sum(labels_hmm == groundTruthLabels)/groundTruthLabels.size)

plt.subplot(224)
plt.plot( labels_hmm, label='HMM Labels')
plt.title('HMM Labels')
plt.xlabel('Time')
plt.ylabel('Label')
plt.legend()

plt.tight_layout()
plt.show()
```

```
KNNaccuracy:  0.625438596491228
GMMaccuracy:  0.8842105263157894
HMMaccuracy:  0.9114035087719298
```



EXPLANATION

KNN: This performs the worst as we are constrained to euclidian distance as our only metric of seperation. Does not account for different variances on different features. We also don't consider the samples as related in time.

GMM: Performs significantly better than KNN as we allow different variances/covariances for/between our different features when clustering. We have the added benefits of learning gaussian distributions to describe each of our classes. We still don't consider the samples as related in time.

HMM: Performs the best as we get all the benefits of a GMM while now considering transition probabilities between states. In qualitative terms, we now consider the fact the audio is not likely to quickly switch between music/laughter/speech and that if one of these is occuring the most likely state for the next column of our spectrogram is that it will continue. Even though the percentage correct is only 2-3% greater than GMMs, the labels make much more sense due to these considerations.

In [ ]:
```
from scipy.io import wavfile
from scipy.signal import stft
import matplotlib.pyplot as plt
import numpy as np
# import librosa as librosa
from hmmlearn import hmm
import os
from sklearn.model_selection import train_test_split
np.random.seed(1)
```

In [ ]:
```
def load_audio_data(folder_path, num_recordings_per_digit=20):
    data = []
    labels = []


    for digit in range(10):
        digit_count = 0
        # maxlen = 0
        for filename in os.listdir(folder_path):
            if filename.endswith(".wav") and filename.startswith('digit'+str(digit)):
                file_path = os.path.join(folder_path, filename)

                _ , audio_data = wavfile.read(file_path)

                data.append(audio_data)
                labels.append(digit)

                digit_count += 1
                if digit_count == num_recordings_per_digit:
                    break

    return data, np.array(labels)

folder_path = './digitRecordings/'
audio_data, labels = load_audio_data(folder_path)

n_fft = 1024
noverlap  = 512
# n_mfcc = 13
for i in range(len(audio_data)):
    _,_,  audio_data[i] = stft(audio_data[i], nperseg=n_fft, noverlap=noverlap)
    audio_data[i] = 10*np.log(np.abs(audio_data[i]) + 1e-14)
    # audio_data[i] = librosa.feature.mfcc(audio_data[i].astype(float),sr = 10,n_me=n_

print("Audio Data Shape:", len(audio_data))
print("Labels:", labels)
```

```
Audio Data Shape: 200
Labels: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9]
```

In [ ]:
```
def pca(data): #takes input data as columns and expects it to be 0 mean (add mean back
    covariance_matrix = np.dot(data,data.T)/(data.shape[1])
    eigvals, eigvecs = np.linalg.eigh(covariance_matrix)
```

```python
    eigvals = eigvals[::-1]
    eigvecs = eigvecs[:,::-1]

    return (eigvecs,eigvals)

#takes data with rows as features, columns as samples
def pcaReduce(data,n_components): #return eigvals and reduced data, and mean it subtra
    mean = np.mean(data,axis=1).reshape(-1,1)
    vecs, vals = pca(data - mean)
    PCAmat = vecs.T[:n_components]
    reduced = PCAmat @ (data - mean)
    return reduced,vals,mean, PCAmat
```

In [ ]:
```python
X_train, X_test, y_train, y_test = train_test_split(audio_data, labels, test_size=.25,
sort_indices = np.argsort(y_train)

# Sort X_train and y_train by the labels
X_train = [X_train[i] for i in sort_indices]
y_train = y_train[sort_indices]

# Verify the shapes of the resulting sets
print("X_train_sorted shape:", len(X_train))
print("y_train_sorted shape:", y_train.shape)
```

```
X_train_sorted shape: 150
y_train_sorted shape: (150,)
```

In [ ]:
```python
nPCA = 5
reducedTrain,eigvals,PCAmean,PCAmat  = pcaReduce(np.hstack(X_train),nPCA)
temp  = []
startidx =0
for i in range(len(X_train)):
    temp.append(reducedTrain[:,startidx:startidx+X_train[i].shape[1]])
    startidx += X_train[i].shape[1]
reducedTrain=temp
```

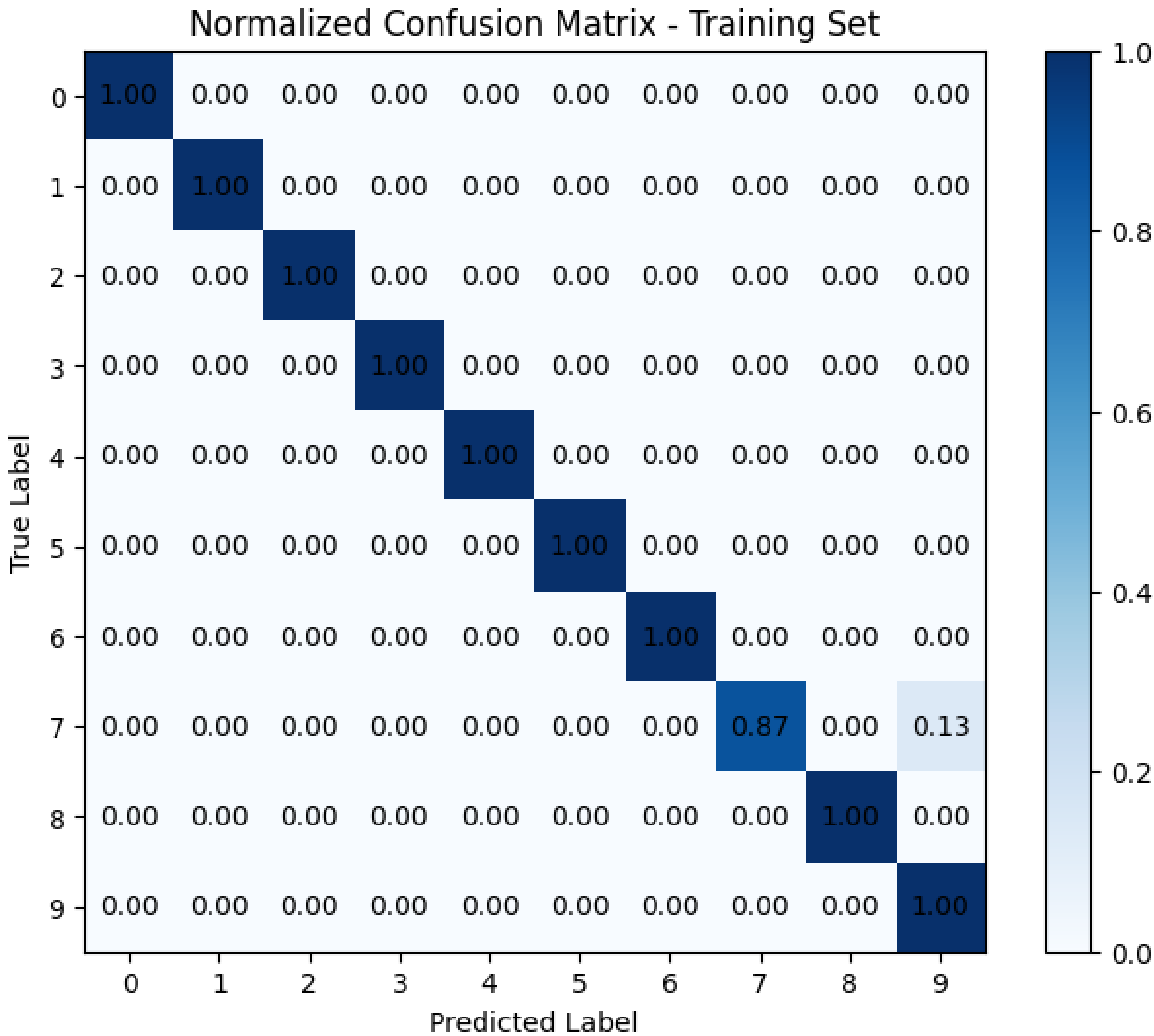
In [ ]:
```python
def fitHmm(spectrogramList, n_states, n_gaussians):
    model = hmm.GMMHMM(n_components=n_states, n_mix=n_gaussians, covariance_type='diag

    X = np.hstack(spectrogramList)

    lengths = [spec.shape[1] for spec in spectrogramList]
    X = X.T

    model.fit(X, lengths=lengths)
    return model

def evaluatefromHmms(model_list, spectrogram):
    scores = np.zeros(len(model_list))
    for i in range(len(model_list)):
        scores[i], _ = model_list[i].decode(spectrogram.T)
    return np.argmax(np.exp(scores))
```

In [ ]:
```
digitModels = []
n_states = 5
n_gaussians =1
for i in range(10):
    # print(i)
    digitModels.append(fitHmm(reducedTrain[i*15:(i+1)*15],n_states=n_states, n_gaussia
```

In [ ]:
```
%%capture

trainConfMat = np.zeros((10,10))
testConfMat = np.zeros((10,10))

for i in range(len(X_train)):
    yhat = evaluatefromHmms(digitModels,PCAmat @(X_train[i] - PCAmean))
    y = y_train[i]
    trainConfMat[y,yhat] +=1

for i in range(len(X_test)):
    yhat = evaluatefromHmms(digitModels,PCAmat @(X_test[i] - PCAmean))
    y = y_test[i]
    testConfMat[y,yhat] +=1
```

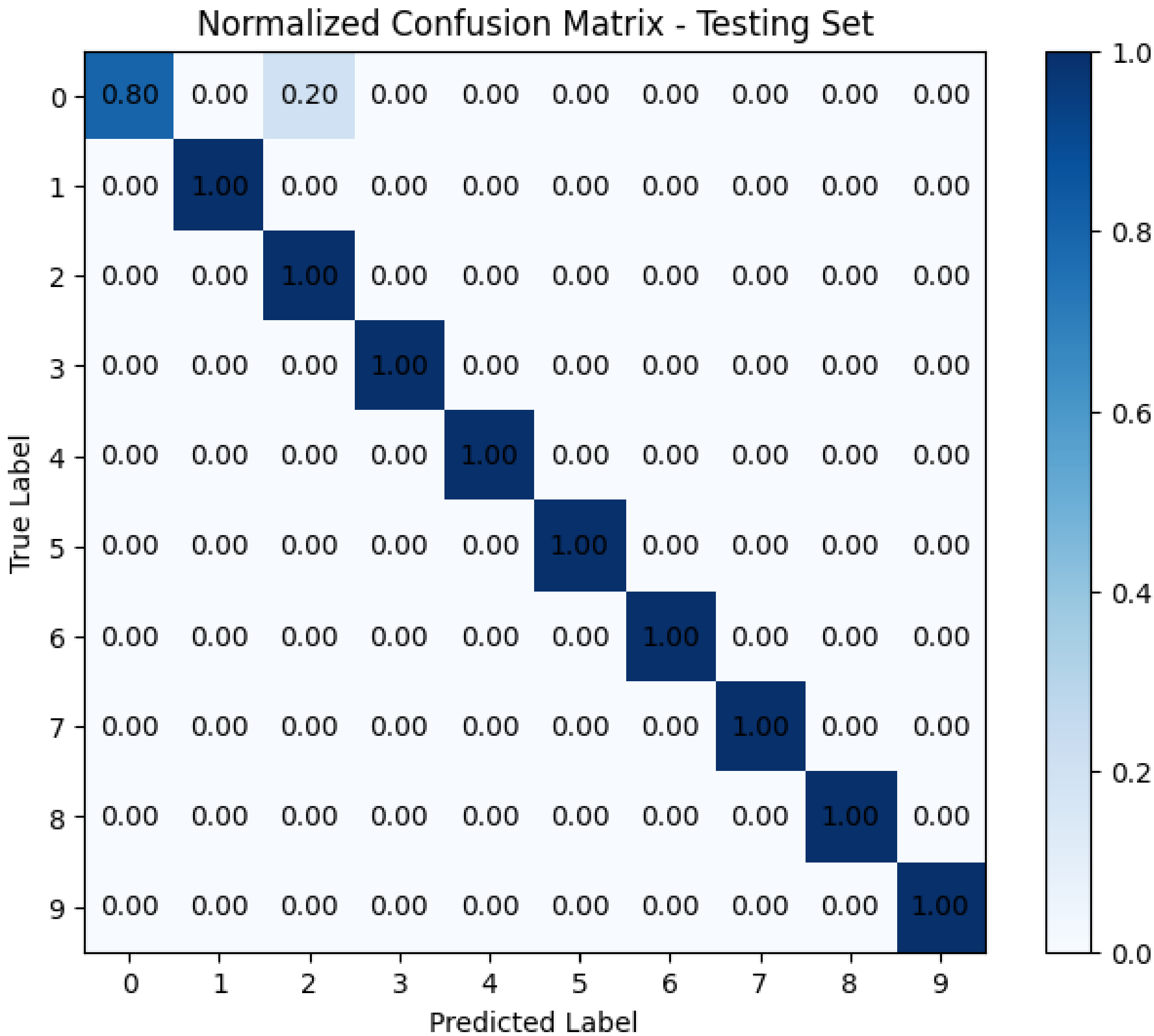
In [ ]:
```
trainConfMatNormalized = trainConfMat.astype('float') / trainConfMat.sum(axis=1)[:, np
plt.figure(figsize=(8, 6))
cax = plt.imshow(trainConfMatNormalized, cmap="Blues")
for i in range(trainConfMatNormalized.shape[0]):
    for j in range(trainConfMatNormalized.shape[1]):
        plt.text(j, i, f'{trainConfMatNormalized[i, j]:.2f}', ha='center', va='center'

plt.xticks(np.arange(trainConfMatNormalized.shape[1]))
plt.yticks(np.arange(trainConfMatNormalized.shape[0]))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Normalized Confusion Matrix - Training Set')

cbar = plt.colorbar(cax)
print("Using", PCAmat.shape[0], " PCA components and ", n_states, "HMM states per mode

plt.show()
```

```
Using 5  PCA components and  5 HMM states per model
```




```
In [ ]: testConfMatNormalized = testConfMat.astype('float') / testConfMat.sum(axis=1)[:, np.ne
plt.figure(figsize=(8, 6))
cax = plt.imshow(testConfMatNormalized, cmap="Blues")
for i in range(testConfMatNormalized.shape[0]):
    for j in range(testConfMatNormalized.shape[1]):
        plt.text(j, i, f'{testConfMatNormalized[i, j]:.2f}', ha='center', va='center',

plt.xticks(np.arange(testConfMatNormalized.shape[1]))
plt.yticks(np.arange(testConfMatNormalized.shape[0]))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Normalized Confusion Matrix - Testing Set')

print("Using", PCAmat.shape[0], " PCA components and ", n_states, "HMM states per mode

cbar = plt.colorbar(cax)

plt.show()
```

```
Using 5  PCA components and  5 HMM states per model
```



Normalized Confusion Matrix - Testing Set
True Label
Predicted Label
0.80 0.00 0.20 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00 0.00
0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 1.00
0 1 2 3 4 5 6 7 8 9
1.0 0.8 0.6 0.4 0.2 0.0

In [ ]:

```
In [ ]: import os
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        np.random.seed(1)
```

```
In [ ]: data = []
        labelsIncluded = [1,2,3,4]
        path = './UCI_HAR_Dataset/test/Inertial_Signals/'
        xaccel = pd.read_csv(path + 'total_acc_x_test.txt', sep=" ",  header=None, skipinitial
        yaccel = pd.read_csv(path + 'total_acc_y_test.txt', sep=" ",  header=None, skipinitial
        zaccel = pd.read_csv(path + 'total_acc_z_test.txt', sep=" ",  header=None, skipinitial
        labels = pd.read_csv('./UCI_HAR_Dataset/test/y_test.txt', sep=" ",  header=None, skipi
        mask = np.isin(labels.flatten(), labelsIncluded)

        xaccel= xaccel[mask,:]
        yaccel= yaccel[mask,:]
        zaccel= zaccel[mask,:]
        labels = labels[mask,:].flatten()

        totalAccel = np.dstack((xaccel,yaccel,zaccel))
        X_train, X_test, y_train, y_test = train_test_split(totalAccel, labels, test_size=0.5)
        print(X_train.shape, y_train.shape)
```

```
(939, 128, 3) (939,)
```

```
In [ ]: Xtrain1 = X_train[y_train == 1,:,:]
        Xtrain2 = X_train[y_train == 2,:,:]
        Xtrain3 = X_train[y_train == 3,:,:]
        Xtrain4 = X_train[y_train == 4,:,:]
        Xtrain1 = np.vstack(Xtrain1)
        Xtrain2 = np.vstack(Xtrain2)
        Xtrain3 = np.vstack(Xtrain3)
        Xtrain4 = np.vstack(Xtrain4)
        print(Xtrain1.shape,Xtrain2.shape, Xtrain3.shape,Xtrain4.shape)
```

```
(33792, 3) (29824, 3) (26624, 3) (29952, 3)
```

VAR training

```
In [ ]: #Plan: Use Matrix compact form and do Psuedo inverse (https://handwiki.org/wiki/Genera

        def getVARWeights(dataMat, N=4): #dataMat.shape = (timestamps, Dimensions). Returns a
            Y = np.zeros((dataMat.shape[1], dataMat.shape[0]-N))
            Z = np.zeros((1+ N *dataMat.shape[1] ,dataMat.shape[0]-N))
            Z[0,:] = 1
            for i in range(N,dataMat.shape[0]):
                Y[:,i-N] = dataMat[i]
            for i in range(0,N):
                for j in range(0,dataMat.shape[0]-N):
                    rowIdx = 1 + i*3
                    Z[rowIdx:rowIdx + dataMat.shape[1],j] = dataMat[N-1-i+j,:]
            toReturn = Y @ np.linalg.pinv(Z)
            return toReturn
        def predictSamples(dataMat, WeightMat, N = 4): # returns the samples that we can predi
            Z = np.zeros((1+ N *dataMat.shape[1] ,dataMat.shape[0]-N))
```

```
    Z[0,:] = 1
    for i in range(0,N):
        for j in range(0,dataMat.shape[0]-N):
            rowIdx = 1 + i*3
            Z[rowIdx:rowIdx + dataMat.shape[1],j] = dataMat[N-1-i+j,:]
    return (WeightMat @ Z).T

def predictLabel(dataMat, allWeights, N = 4): #Assumes Data is shape (timestamps, Dime
    groundTruth = dataMat[N:,:]
    errors = np.zeros(len(allWeights))
    for i in range(len(allWeights)):
        predicted = predictSamples(dataMat,allWeights[i],N)
        # errors[i] = np.multiply(predicted,groundTruth)
        errors[i] = np.sum(np.square(groundTruth - predictSamples(dataMat,allWeights[i
    return np.argmin(errors)

Norder = 10
WeightMatAct1 = getVARWeights(Xtrain1, N=Norder)
WeightMatAct2 = getVARWeights(Xtrain2,N=Norder)
WeightMatAct3 = getVARWeights(Xtrain3,N = Norder)
WeightMatAct4 = getVARWeights(Xtrain4, N = Norder)

allWeights = [WeightMatAct1,WeightMatAct2,WeightMatAct3,WeightMatAct4]
```

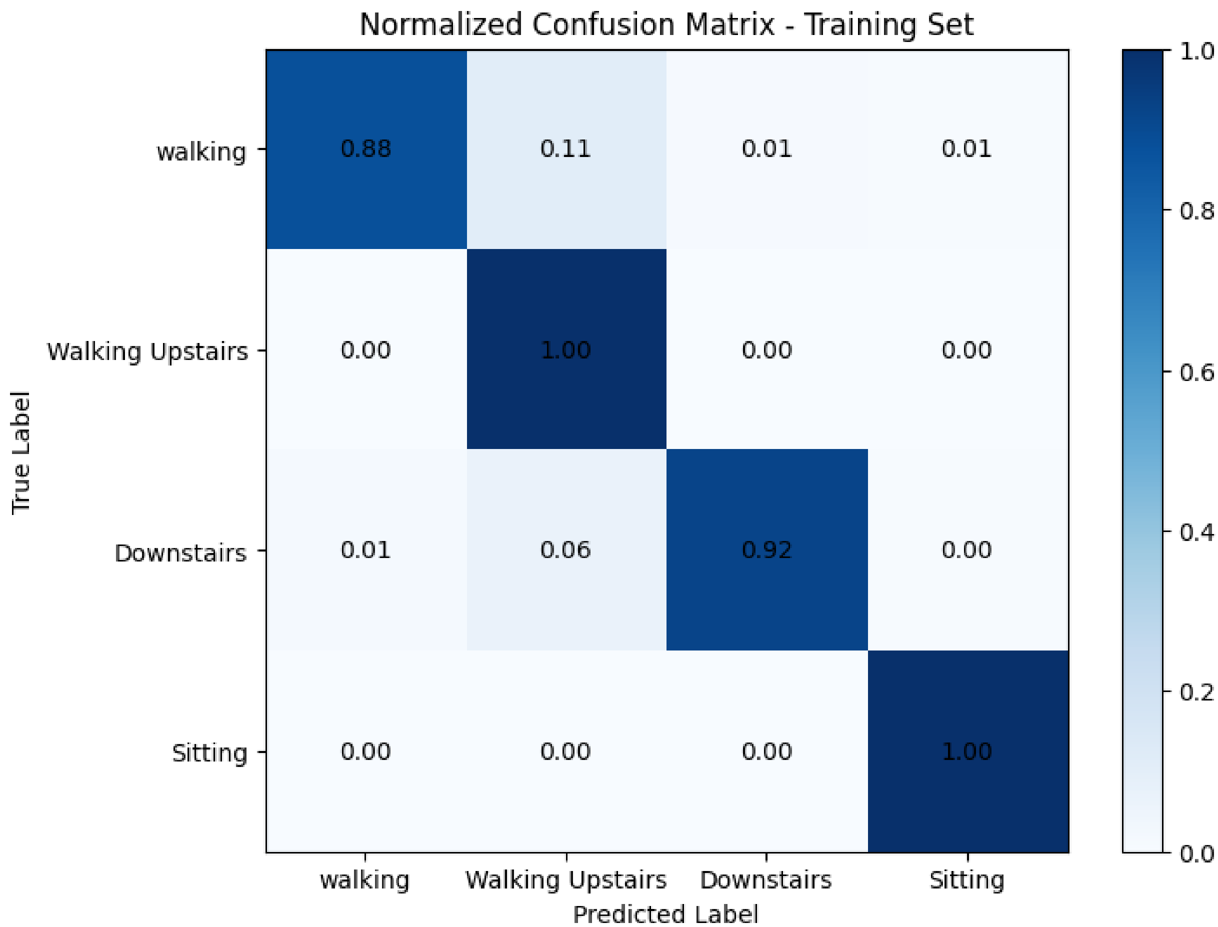
In [ ]:
```
# print(y_train[1])
# GTvisual = X_train[0,Norder:,2]
# prediction = predictSamples(X_train[0],WeightMatAct3,Norder)[:,2]
# plt.plot(GTvisual)
# plt.plot(prediction)
```

In [ ]:
```
trainConfMat = np.zeros((4,4))
testConfMat = np.zeros((4,4))

for i in range(X_train.shape[0]):
    yhat = predictLabel(X_train[i], allWeights, N=Norder)
    y = y_train[i]
    trainConfMat[yhat,y-1] +=1

for i in range(X_test.shape[0]):
    yhat = predictLabel(X_test[i], allWeights, N=Norder)
    y = y_test[i]
    testConfMat[yhat,y-1] +=1
```

In [ ]:
```
trainConfMatNormalized = trainConfMat.astype('float') / trainConfMat.sum(axis=1)[:, np
plt.figure(figsize=(8, 6))
cax = plt.imshow(trainConfMatNormalized, cmap="Blues")
for i in range(trainConfMatNormalized.shape[0]):
    for j in range(trainConfMatNormalized.shape[1]):
        plt.text(j, i, f'{trainConfMatNormalized[i, j]:.2f}', ha='center', va='center'

plt.xticks(np.arange(trainConfMatNormalized.shape[1]), ["walking", "Walking Upstairs",
plt.yticks(np.arange(trainConfMatNormalized.shape[0]), ["walking", "Walking Upstairs",
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Normalized Confusion Matrix - Training Set')

cbar = plt.colorbar(cax)
print("Using", Norder, "past samples to predict")
```

```
plt.show()
```

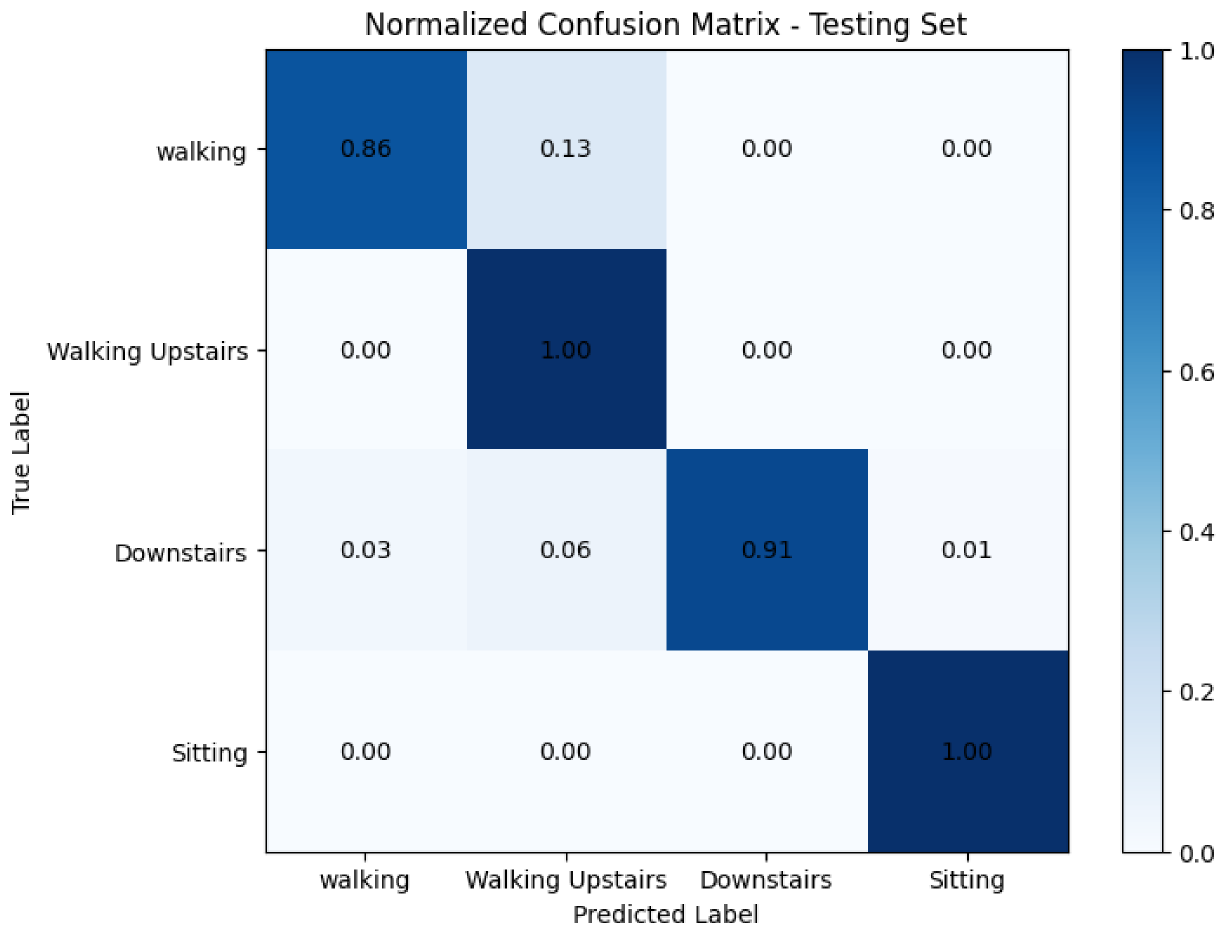
```
Using 10 past samples to predict
```



In [ ]:

```
testConfMatNormalized = testConfMat.astype('float') / testConfMat.sum(axis=1)[:, np.ne
plt.figure(figsize=(8, 6))
cax = plt.imshow(testConfMatNormalized, cmap="Blues")
for i in range(testConfMatNormalized.shape[0]):
    for j in range(testConfMatNormalized.shape[1]):
        plt.text(j, i, f'{testConfMatNormalized[i, j]:.2f}', ha='center', va='center',

plt.xticks(np.arange(testConfMatNormalized.shape[1]), ["walking", "Walking Upstairs",
plt.yticks(np.arange(testConfMatNormalized.shape[0]), ["walking", "Walking Upstairs",
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Normalized Confusion Matrix - Testing Set')

print("Using", Norder, "past samples to predict")

cbar = plt.colorbar(cax)

plt.show()
```

```
Using 10 past samples to predict
```



Normalized Confusion Matrix - Testing Set
True Label
Predicted Label
walking
Walking Upstairs
Downstairs
Sitting
0.86
0.13
0.00
0.00
0.00
1.00
0.00
0.00
0.03
0.06
0.91
0.01
0.00
0.00
0.00
1.00
0.0
0.2
0.4
0.6
0.8
1.0

In [ ]: