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 [ ]:

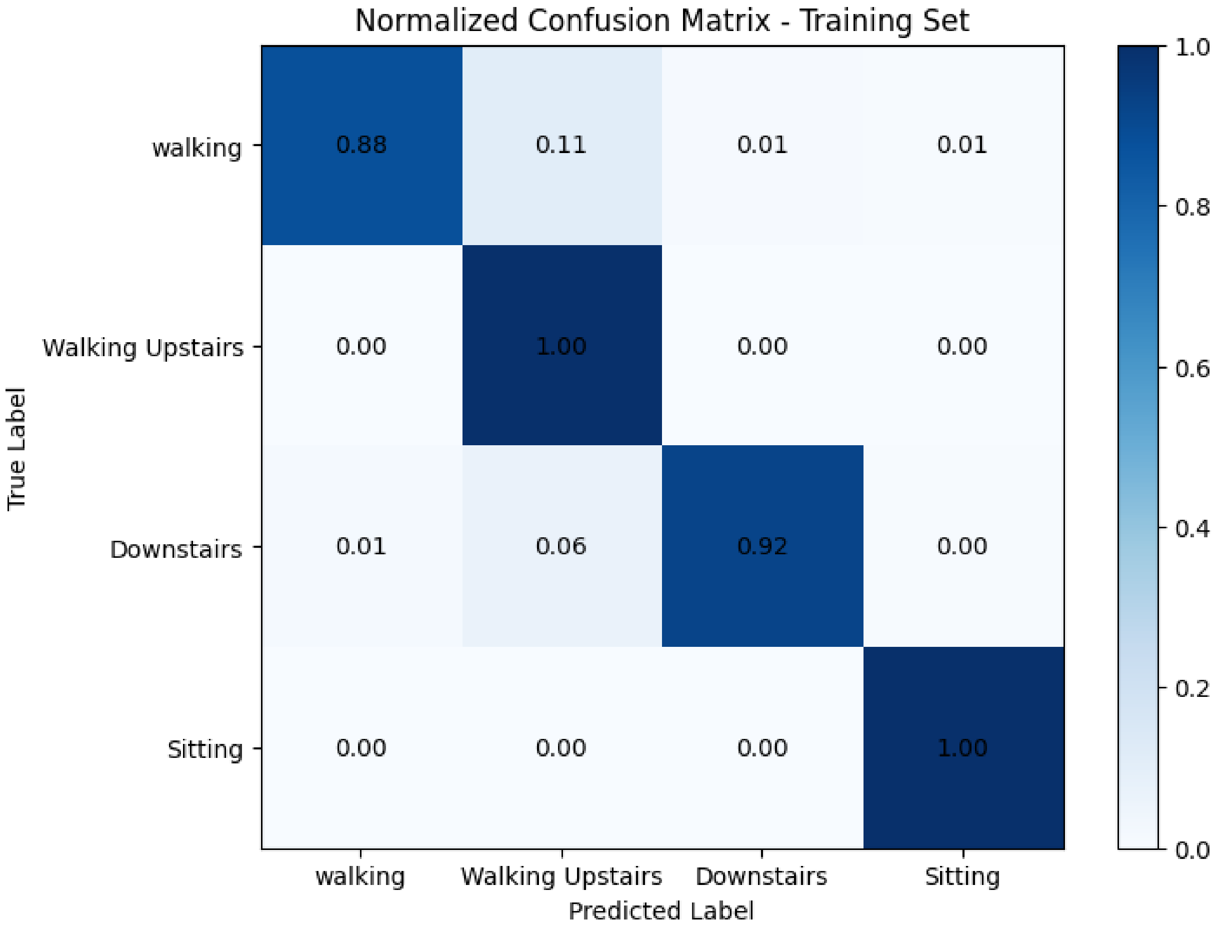
```
trainConfMat = np.zeros((4,4))
testConfMat = np.zeros((4,4))

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()
```

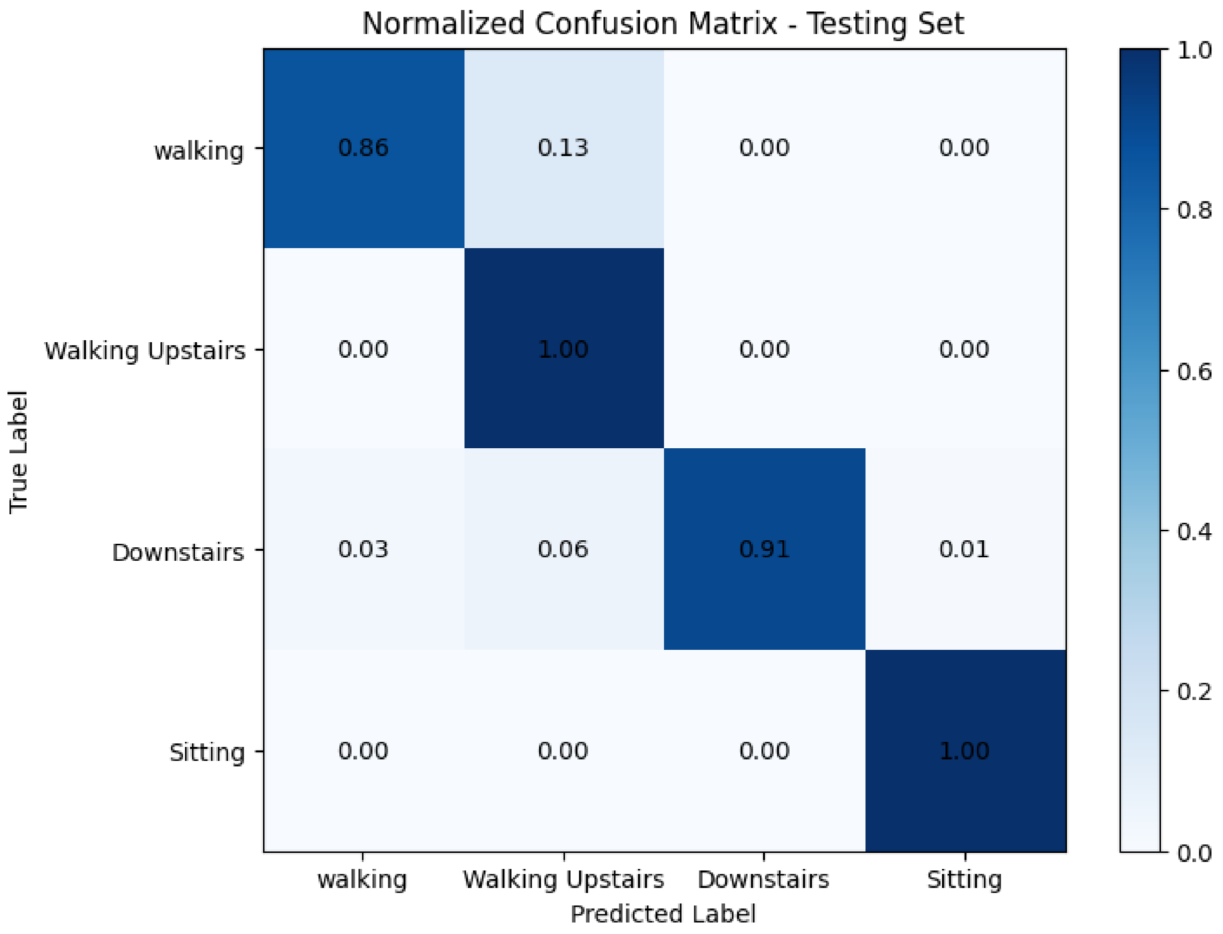
```
Using 10 past samples to predict
```



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
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
walking
Walking Upstairs
Downstairs
Sitting
Predicted Label
1.0
0.8
0.6
0.4
0.2
0.0

In [ ]: