HumanActivityRecognition

This project is to build a model that predicts the human activities such as Walking, Walking_Upstairs, Walking_Downstairs, Sitting, Standing or Laying.

This dataset is collected from 30 persons(referred as subjects in this dataset), performing different activities with a smartphone to their waists. The data is recorded with the help of sensors (accelerometer and Gyroscope) in that smartphone. This experiment was video recorded to label the data manually.

How data was recorded

By using the sensors(Gyroscope and accelerometer) in a smartphone, they have captured '3-axial linear acceleration'(_tAcc-XYZ_) from accelerometer and '3-axial angular velocity' (_tGyro-XYZ_) from Gyroscope with several variations.

> prefix 't' in those metrics denotes time.

> suffix 'XYZ' represents 3-axial signals in X , Y, and Z directions.

Feature names

1. These sensor signals are preprocessed by applying noise filters and then sampled in fixed-width windows(sliding windows) of 2.56 seconds each with 50% overlap. ie., each window has 128 readings.
2. From Each window, a feature vector was obtianed by calculating variables from the time and frequency domain.

> In our dataset, each datapoint represents a window with different readings

3. The accelertion signal was saperated into Body and Gravity acceleration signals(***tBodyAcc-XYZ*** and ***tGravityAcc-XYZ***) using some low pass filter with corner frequecy of 0.3Hz.
4. After that, the body linear acceleration and angular velocity were derived in time to obtian *jerk signals* (***tBodyAccJerk-XYZ*** and ***tBodyGyroJerk-XYZ***).
5. The magnitude of these 3-dimensional signals were calculated using the Euclidian norm. This magnitudes are represented as features with names like *tBodyAccMag_*, *_tGravityAccMag_*, *_tBodyAccJerkMag_*, *_tBodyGyroMag* and *tBodyGyroJerkMag*.
6. Finally, We've got frequency domain signals from some of the available signals by applying a FFT (Fast Fourier Transform). These signals obtained were labeled with ***prefix 'f'*** just like original signals with ***prefix 't'***. These signals are labeled as ***fBodyAcc-XYZ***, ***fBodyGyroMag*** etc.,.
7. These are the signals that we got so far.

- tBodyAcc-XYZ
- tGravityAcc-XYZ
- tBodyAccJerk-XYZ
- tBodyGyro-XYZ
- tBodyGyroJerk-XYZ
- tBodyAccMag
- tGravityAccMag
- tBodyAccJerkMag
- tBodyGyroMag
- tBodyGyroJerkMag
- fBodyAcc-XYZ
- fBodyAccJerk-XYZ
- fBodyGyro-XYZ
- fBodyAccMag

- fBodyAccJerkMag
- fBodyGyroMag
- fBodyGyroJerkMag

8. We can esitmate some set of variables from the above signals. ie., We will estimate the following properties on each and every signal that we recoreded so far.

   - ***mean()***: Mean value
   - ***std()***: Standard deviation
   - ***mad()***: Median absolute deviation
   - ***max()***: Largest value in array
   - ***min()***: Smallest value in array
   - ***sma()***: Signal magnitude area
   - ***energy()***: Energy measure. Sum of the squares divided by the number of values.
   - ***iqr()***: Interquartile range
   - ***entropy()***: Signal entropy
   - ***arCoeff()***: Autorregresion coefficients with Burg order equal to 4
   - ***correlation()***: correlation coefficient between two signals
   - ***maxInds()***: index of the frequency component with largest magnitude
   - ***meanFreq()***: Weighted average of the frequency components to obtain a mean frequency
   - ***skewness()***: skewness of the frequency domain signal
   - ***kurtosis()***: kurtosis of the frequency domain signal
   - ***bandsEnergy()***: Energy of a frequency interval within the 64 bins of the FFT of each window.
   - ***angle()***: Angle between to vectors.

9. We can obtain some other vectors by taking the average of signals in a single window sample. These are used on the angle() variable' `

   - gravityMean
   - tBodyAccMean
   - tBodyAccJerkMean
   - tBodyGyroMean
   - tBodyGyroJerkMean

Y_Labels(Encoded)

- In the dataset, Y_labels are represented as numbers from 1 to 6 as their identifiers.
  - WALKING as **1**
  - WALKING_UPSTAIRS as **2**
  - WALKING_DOWNSTAIRS as **3**
  - SITTING as **4**
  - STANDING as **5**
  - LAYING as **6**

Train and test data were saperated

- The readings from ***70%*** of the volunteers were taken as ***trianing data*** and remaining ***30%*** subjects recordings were taken for ***test data***

Data

- All the data is present in 'UCI_HAR_dataset/' folder in present working directory.
  - Feature names are present in 'UCI_HAR_dataset/features.txt'
  - ***Train Data***
    - 'UCI_HAR_dataset/train/X_train.txt'

- 'UCI_HAR_dataset/train/subject_train.txt'
- 'UCI_HAR_dataset/train/y_train.txt'

- ***Test Data***
  - 'UCI_HAR_dataset/test/X_test.txt'
  - 'UCI_HAR_dataset/test/subject_test.txt'
  - 'UCI_HAR_dataset/test/y_test.txt'

Data Size :

> 27 MB

Quick overview of the dataset :

- Accelerometer and Gyroscope readings are taken from 30 volunteers(referred as subjects) while performing the following 6 Activities.

  1. Walking
  2. WalkingUpstairs
  3. WalkingDownstairs
  4. Standing
  5. Sitting
  6. Lying.

- Readings are divided into a window of 2.56 seconds with 50% overlapping.
- Accelerometer readings are divided into gravity acceleration and body acceleration readings, which has x,y and z components each.
- Gyroscope readings are the measure of angular velocities which has x,y and z components.
- Jerk signals are calculated for BodyAcceleration readings.
- Fourier Transforms are made on the above time readings to obtain frequency readings.
- Now, on all the base signal readings., mean, max, mad, sma, arcoefficient, engerybands,entropy etc., are calculated for each window.
- We get a feature vector of 561 features and these features are given in the dataset.
- Each window of readings is a datapoint of 561 features.

Problem Framework

- 30 subjects(volunteers) data is randomly split to 70%(21) test and 30%(7) train data.
- Each datapoint corresponds one of the 6 Activities.

Problem Statement

- Given a new datapoint we have to predict the Activity

Get the handcrafted features

In [9]:

```
#### importing libraries
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import linear_model
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from datetime import datetime
from sklearn.metrics import confusion_matrix
```

In [2]:

```


# get the features from the file features.txt
features = list()
with open('UCI_HAR_Dataset/features.txt') as f:
    features = [line.split()[1] for line in f.readlines()]
print('No of Features: {}'.format(len(features)))

```

```
No of Features: 561
```

Obtain the train data

In [3]:

```
# get the data from txt files to pandas dataffame
X_train = pd.read_csv('UCI_HAR_dataset/train/X_train.txt', delim_whitespace=True, header=None, names=features)

# add subject column to the dataframe
X_train['subject'] = pd.read_csv('UCI_HAR_dataset/train/subject_train.txt', header=None, squeeze=True)

y_train = pd.read_csv('UCI_HAR_dataset/train/y_train.txt', names=['Activity'], squeeze=True)
y_train_labels = y_train.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS',\
                       4:'SITTING', 5:'STANDING',6:'LAYING'})

# put all columns in a single dataframe
train = X_train
train['Activity'] = y_train
train['ActivityName'] = y_train_labels
train.sample()
```

Out[3]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	angle(tBodyAccMean,gravity)	angle(tBodyAccJerkMean),gravityMean)	angle(tBodyGy
1264	0.353527	-0.036623	-0.093492	-0.296752	0.187732	-0.387994	-0.36065	0.218166	-0.395419	-0.192086	...	-0.727454	0.2369	

1 rows × 564 columns

In [4]:

```
train.shape
```

Out[4]:

```
(7352, 564)
```

Obtain the test data

In [5]:
```
# get the data from txt files to pandas dataffame
X_test = pd.read_csv('UCI_HAR_dataset/test/X_test.txt', delim_whitespace=True, header=None, names=features)

# add subject column to the dataframe
X_test['subject'] = pd.read_csv('UCI_HAR_dataset/test/subject_test.txt', header=None, squeeze=True)

# get y labels from the txt file
y_test = pd.read_csv('UCI_HAR_dataset/test/y_test.txt', names=['Activity'], squeeze=True)
y_test_labels = y_test.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS',\
                       4:'SITTING', 5:'STANDING',6:'LAYING'})


# put all columns in a single dataframe
test = X_test
test['Activity'] = y_test
test['ActivityName'] = y_test_labels
test.sample()
```

Out[5]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	angle(tBodyAccMean,gravity)	angle(tBodyAccJerkMean),gravityMean)	angle(tBodyGy
2560	0.303065	0.002404	-0.084307	-0.012065	0.391696	-0.224547	-0.098529	0.468728	-0.214721	0.177449	...	0.026147	-0.111198	

1 rows × 564 columns

In [6]:
```
test.shape
```

Out[6]:
```
(2947, 564)
```

Data Cleaning

1. Check for Duplicates

In [7]:
```
print('No of duplicates in train: {}'.format(sum(train.duplicated())))
print('No of duplicates in test : {}'.format(sum(test.duplicated())))
```

```
No of duplicates in train: 0
No of duplicates in test : 0
```

2. Checking for NaN/null values

In [8]:
```
print('We have {} NaN/Null values in train'.format(train.isnull().values.sum()))
print('We have {} NaN/Null values in test'.format(test.isnull().values.sum()))
```

```
We have 0 NaN/Null values in train
We have 0 NaN/Null values in test
```

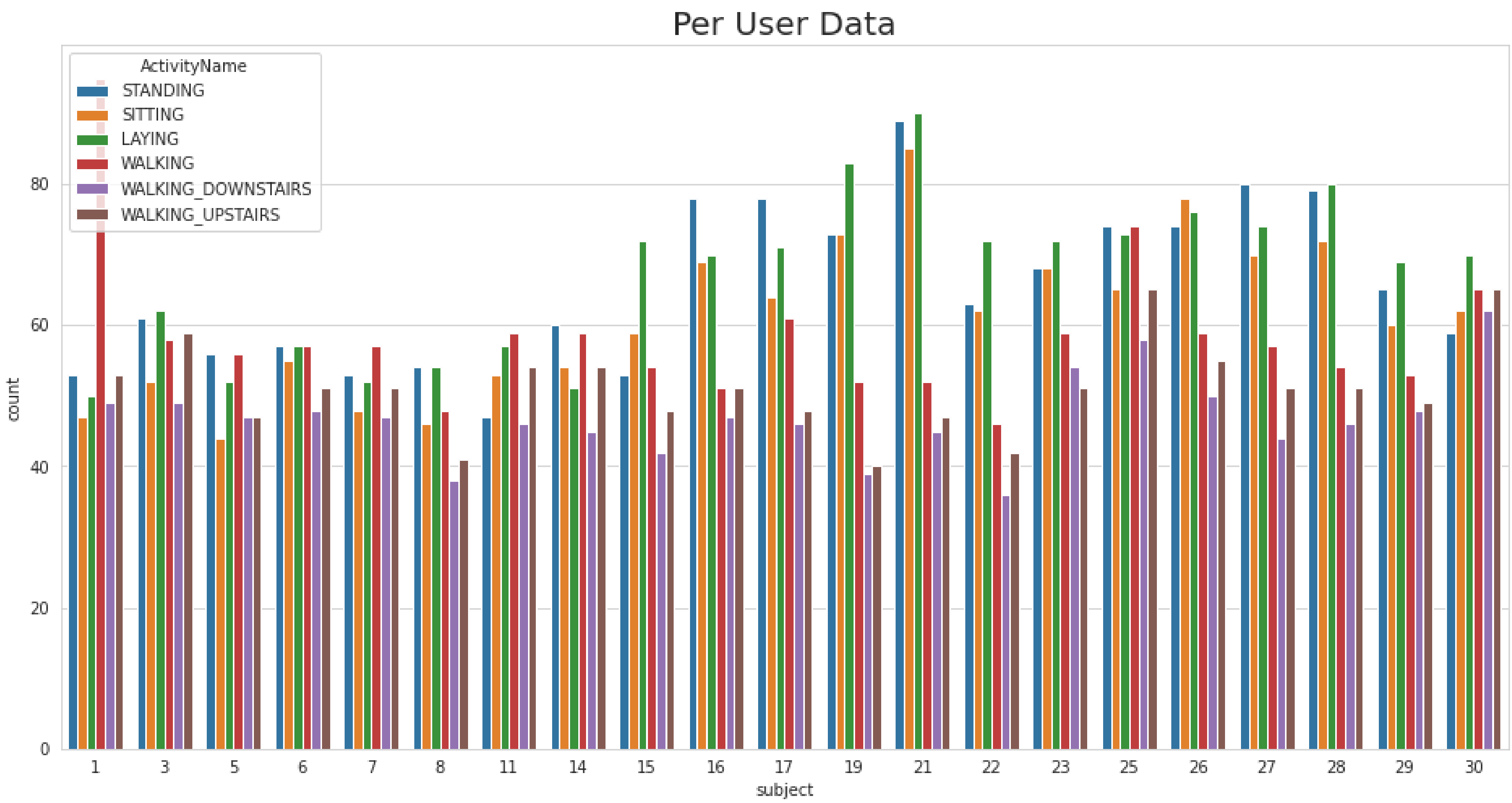
3. Check for data imbalance

In [9]:

```


sns.set_style('whitegrid')
plt.rcParams['font.family'] = 'Dejavu Sans'
```

In [10]:

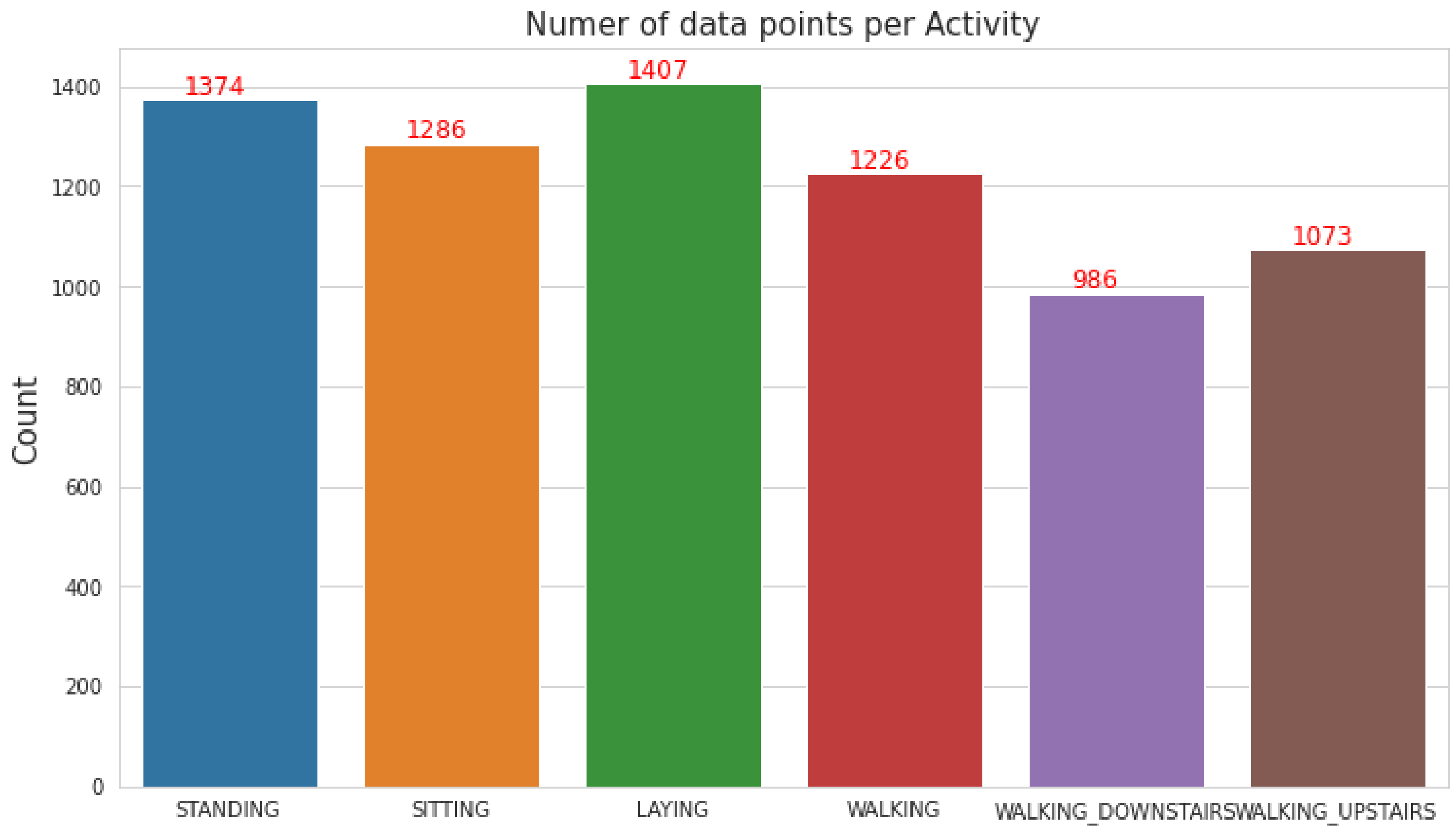
```
plt.figure(figsize=(16,8))
plt.title('Per User Data', fontsize=20)
sns.countplot(x='subject',hue='ActivityName', data = train)
plt.show()

```



We have got almost same number of reading from all the subjects

In [11]:

```
# https://www.listendata.com/2019/06/matplotlib-tutorial-learn-plot-python.html
# https://jovian.ai/rajkap/project
fig = plt.figure(figsize = (9, 5))
ax = fig.add_axes([0,0,1,1])
ax.set_title("Numer of data points per Activity", fontsize = 15)
sns.countplot(x = "ActivityName", data = train)
for i in ax.patches:
    ax.text(x = i.get_x() + 0.2, y = i.get_height()+10, s = str(i.get_height()), fontsize = 12, color = "red")
plt.xlabel("")
plt.ylabel("Count", fontsize = 15)
plt.show()
```



Observation

Our data is well balanced.

4. Changing feature names

In [12]:
```
columns = train.columns

# Removing '()' from column names
columns = columns.str.replace('[()]','')
columns = columns.str.replace('[-]', '')
columns = columns.str.replace('[,]','')

train.columns = columns
test.columns = columns

test.columns
```

Out[12]:
```
Index(['tBodyAccmeanX', 'tBodyAccmeanY', 'tBodyAccmeanZ', 'tBodyAccstdX',
       'tBodyAccstdY', 'tBodyAccstdZ', 'tBodyAccmadX', 'tBodyAccmadY',
       'tBodyAccmadZ', 'tBodyAccmaxX',
       ...
       'angletBodyAccMeangravity', 'angletBodyAccJerkMeangravityMean',
       'angletBodyGyroMeangravityMean', 'angletBodyGyroJerkMeangravityMean',
       'angleXgravityMean', 'angleYgravityMean', 'angleZgravityMean',
       'subject', 'Activity', 'ActivityName'],
      dtype='object', length=564)
```

5. Save this dataframe in a csv files

In [13]:
```
train.to_csv('UCI_HAR_Dataset/csv_files/train.csv', index=False)
test.to_csv('UCI_HAR_Dataset/csv_files/test.csv', index=False)
```

Exploratory Data Analysis

"***Without domain knowledge EDA has no meaning, without EDA a problem has no soul.***"

1. Featuring Engineering from Domain Knowledge

- **Static and Dynamic Activities**
  - In static activities (sit, stand, lie down) motion information will not be very useful.
  - In the dynamic activities (Walking, WalkingUpstairs,WalkingDownstairs) motion info will be significant.

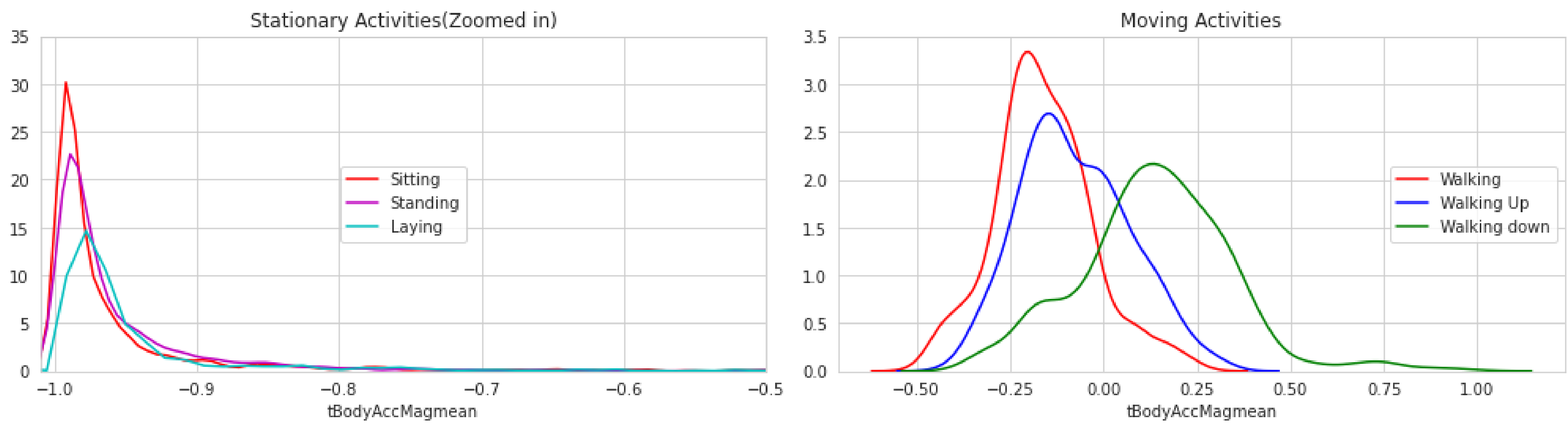
2. Stationary and Moving activities are completely different

In [14]:

```
sns.set_palette("Set1", desat=0.80)
facetgrid = sns.FacetGrid(train, hue='ActivityName', size=6,aspect=2)
facetgrid.map(sns.distplot,'tBodyAccMagmean', hist=False)\
    .add_legend()
plt.annotate("Stationary Activities (Sitting,Standing,Lying)", xy=(-0.956,23), xytext=(-0.7, 26), size=20,\
            va='center', ha='left',\
            arrowprops=dict(arrowstyle="simple",connectionstyle="arc3,rad=0.1"))

plt.annotate("Moving Activities ((Walking, Walking_Upstairs, Walking_Downstairs)", xy=(0,3), xytext=(0.2, 9), size=20,\
            va='center', ha='left',\
            arrowprops=dict(arrowstyle="simple",connectionstyle="arc3,rad=0.1"))
plt.show()
```

In [15]:

```python
# for plotting purposes taking datapoints of each activity to a different dataframe
df1 = train[train['Activity']==1]
df2 = train[train['Activity']==2]
df3 = train[train['Activity']==3]
df4 = train[train['Activity']==4]
df5 = train[train['Activity']==5]
df6 = train[train['Activity']==6]

plt.figure(figsize=(14,7))
plt.subplot(2,2,1)
plt.title('Stationary Activities(Zoomed in)')
sns.distplot(df4['tBodyAccMagmean'],color = 'r',hist = False, label = 'Sitting')
sns.distplot(df5['tBodyAccMagmean'],color = 'm',hist = False,label = 'Standing')
sns.distplot(df6['tBodyAccMagmean'],color = 'c',hist = False, label = 'Laying')
plt.axis([-1.01, -0.5, 0, 35])
plt.legend(loc='center')

plt.subplot(2,2,2)
plt.title('Moving Activities')
sns.distplot(df1['tBodyAccMagmean'],color = 'red',hist = False, label = 'Walking')
sns.distplot(df2['tBodyAccMagmean'],color = 'blue',hist = False,label = 'Walking Up')
sns.distplot(df3['tBodyAccMagmean'],color = 'green',hist = False, label = 'Walking down')
plt.legend(loc='center right')


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



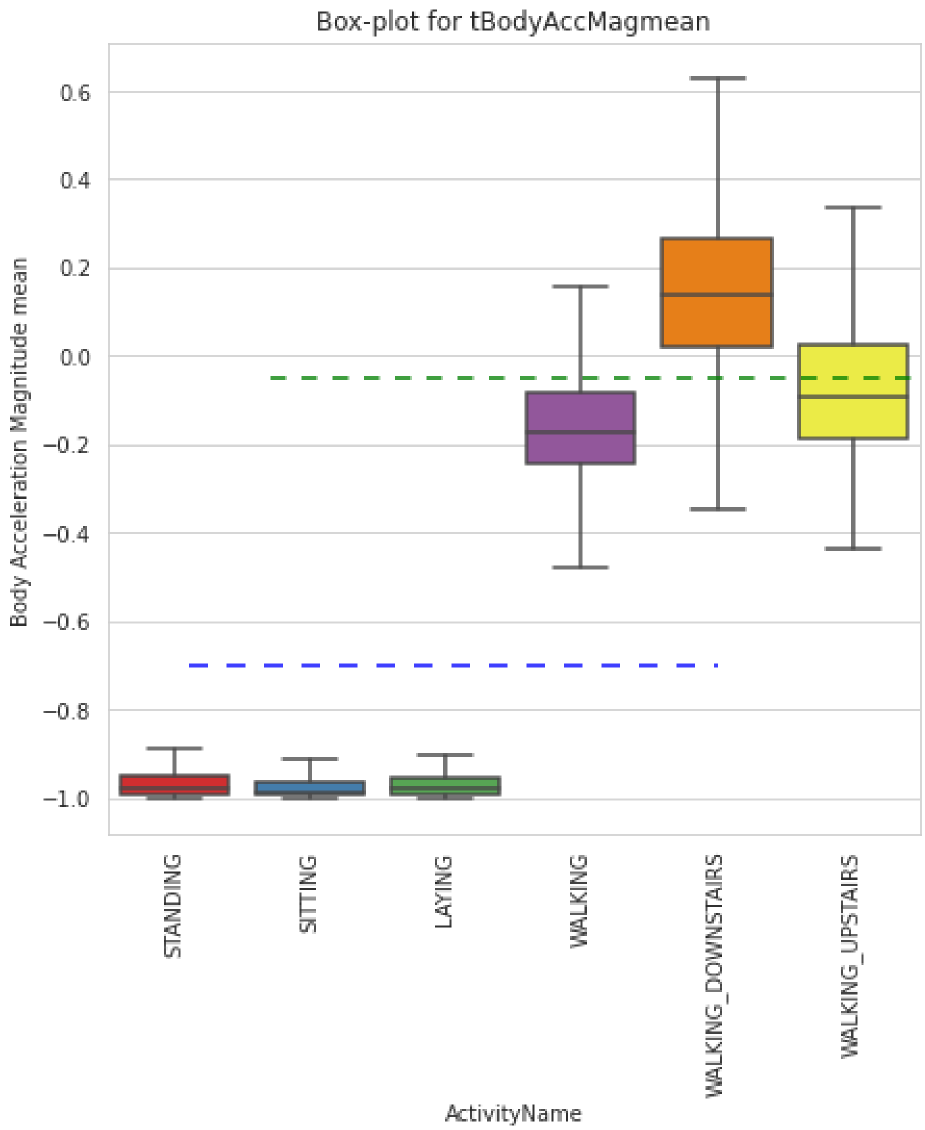
Observations :

- As we can see that the moving activities are well separated than the static activities.tbodyAccMagmean is the mean of the magnitude of body acceleration which is well separated for the dynamic nd static activities.

3. Magnitude of an acceleration can saperate it well

In [16]:

```
plt.figure(figsize=(7,7))
sns.boxplot(x='ActivityName', y='tBodyAccMagmean',data=train, showfliers=False, saturation=1)
plt.ylabel('Body Acceleration Magnitude mean')
plt.axhline(y=-0.7, xmin=0.1, xmax=0.75,dashes=(6,6), c='b')
plt.axhline(y=-0.05, xmin=0.2, dashes=(5,5), c='g')
plt.title('Box-plot for tBodyAccMagmean')
plt.xticks(rotation=90)
plt.show()
```



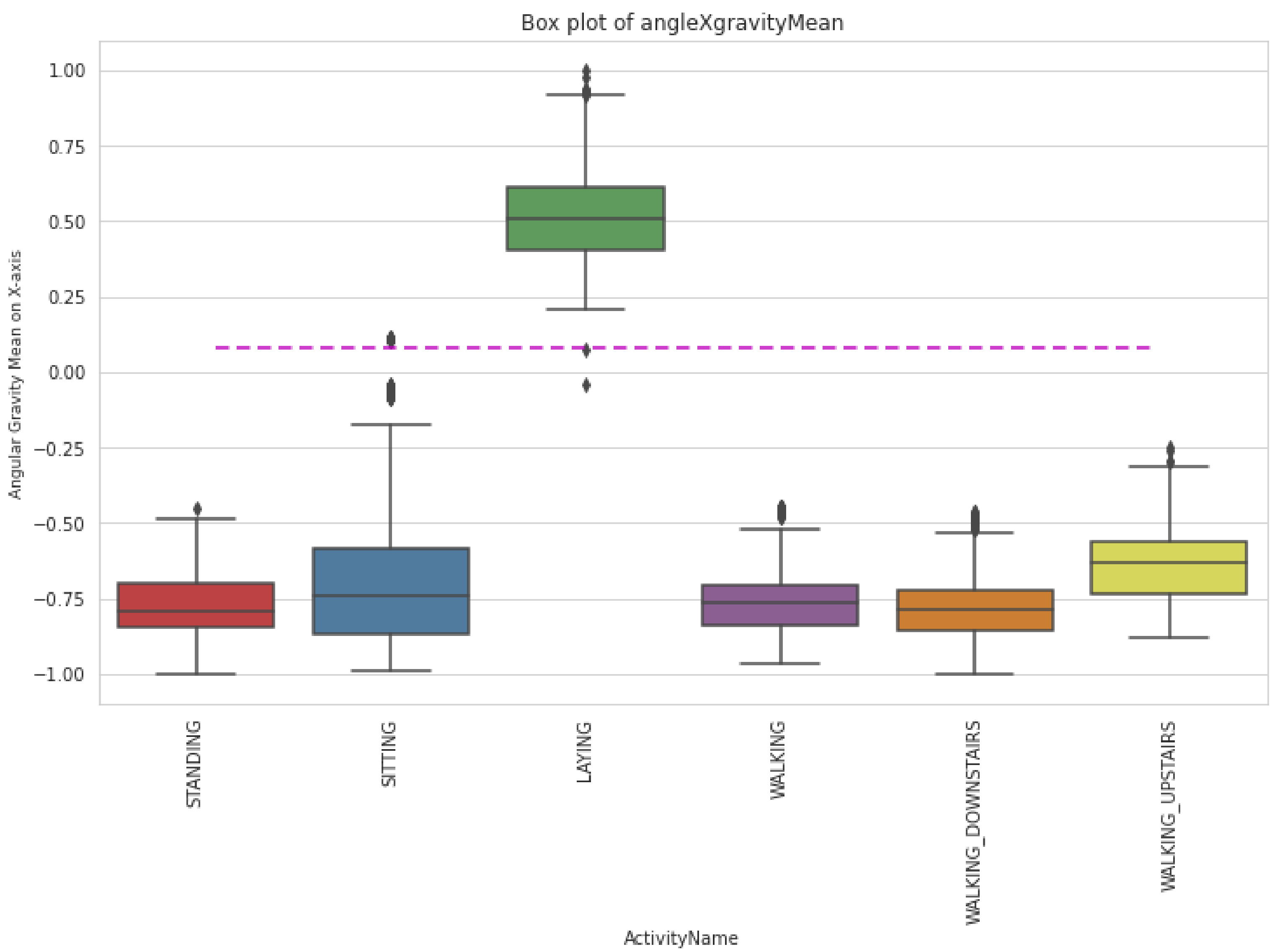
__ Observations__:

- If tAccMean is < -0.8 then the Activities are either Standing or Sitting or Laying.
- If tAccMean is > -0.6 then the Activities are either Walking or WalkingDownstairs or WalkingUpstairs.
- If tAccMean > 0.0 then the Activity is WalkingDownstairs.
- We can classify 75% the Acitivity labels with some errors.

4. Position of GravityAccelerationComponants is important

In [17]:

```
plt.figure(figsize = (12, 7))
sns.boxplot(x='ActivityName', y='angleXgravityMean', data=train)
plt.axhline(y=0.08, xmin=0.1, xmax=0.9,c='m',dashes=(5,3))
plt.title("Box plot of angleXgravityMean", fontsize = 12)
plt.ylabel("Angular Gravity Mean on X-axis", fontsize = 9)
plt.xticks(rotation = 90)
plt.show()
```

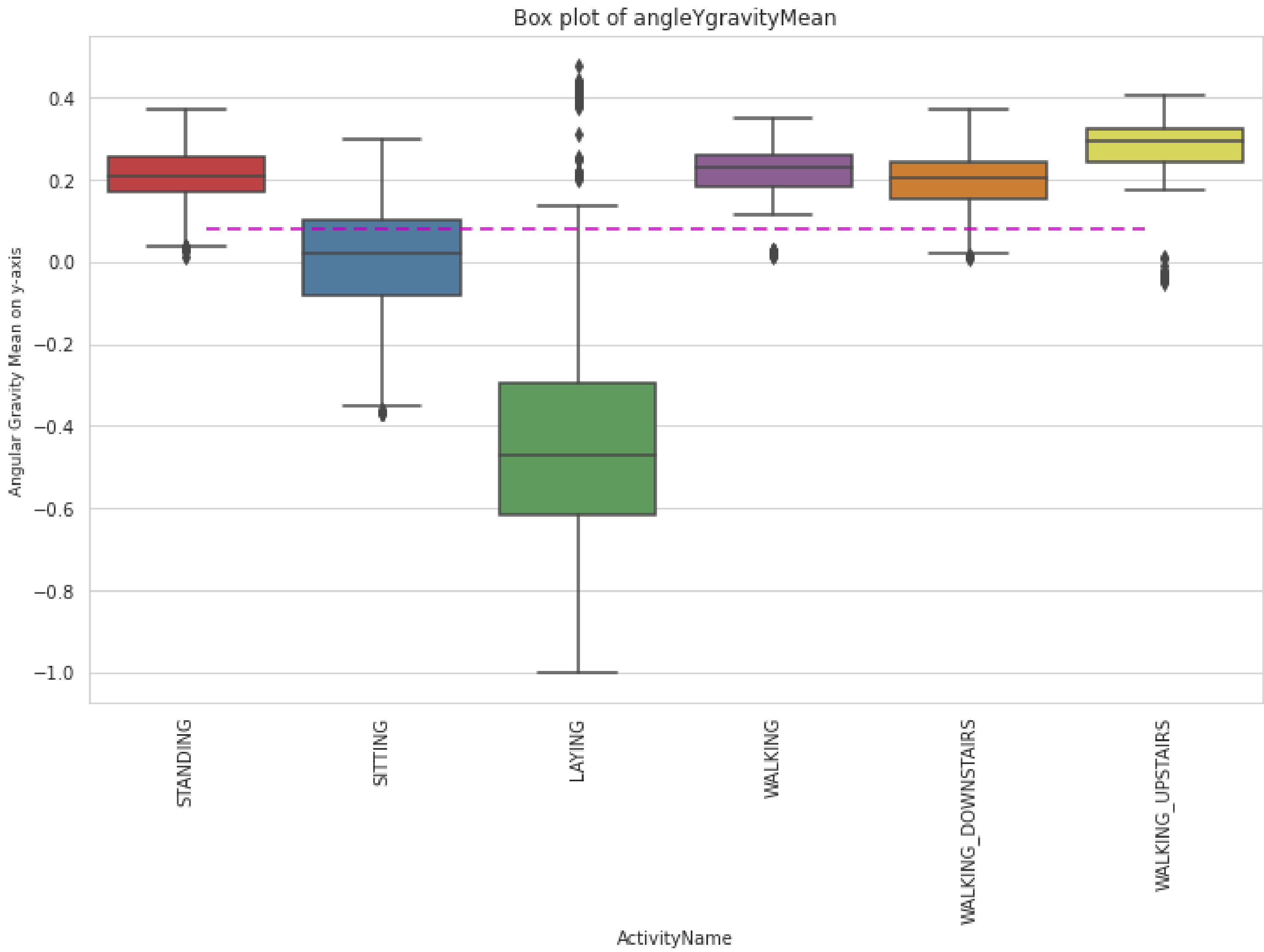


__ Observations__:

- If angleX,gravityMean > 0 then Activity is Laying.
- We can classify all datapoints belonging to Laying activity with just a single if else statement.

In [18]:

```
plt.figure(figsize = (12, 7))
sns.boxplot(x='ActivityName', y='angleYgravityMean', data=train)
plt.axhline(y=0.08, xmin=0.1, xmax=0.9,c='m',dashes=(5,3))
plt.title("Box plot of angleYgravityMean", fontsize = 12)
plt.ylabel("Angular Gravity Mean on y-axis", fontsize = 9)
plt.xticks(rotation = 90)
plt.show()
```



Apply t-sne on the data

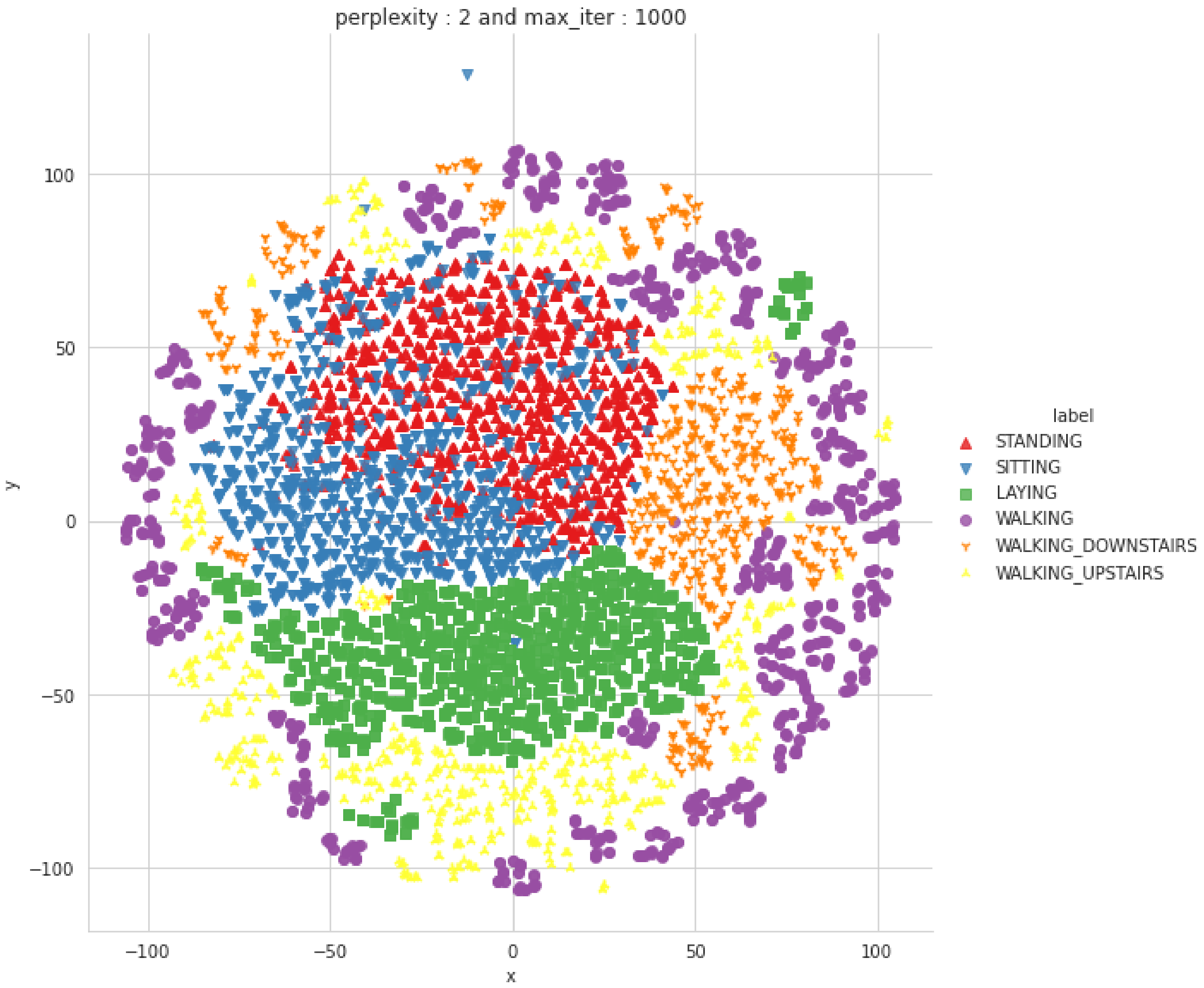
In [20]:

```
# performs t-sne with different perplexity values and their repective plots..

def show_tsne(perplexities,X_data, y_data, n_iter=1000, img_name_prefix='t-sne'):

        # perform t-sne
        print('\nperforming tsne with perplexity {} and with {} iterations at max'.format(perplexity, n_iter))
        lower_dim_x = TSNE(n_components=2, perplexity=perplexity,verbose=2).fit_transform(X_data)
        print('Done..')

        # prepare the data for seaborn
        print('Creating plot for this t-sne visualization..')
        df = pd.DataFrame({'x':lower_dim_x[:,0], 'y':lower_dim_x[:,1] ,'label':y_data})

        # draw the plot in appropriate place in the grid
        sns.lmplot(data=df, x='x', y='y', hue='label', fit_reg=False, size=8,\
                   palette="Set1",markers=['^','v','s','o', '1','2'])
        plt.title("perplexity : {} and max_iter : {}".format(perplexity, n_iter))
        img_name = img_name_prefix + '_perp_{}_iter_{}.png'.format(perplexity, n_iter)
        print('saving this plot as image in present working directory...')
        plt.savefig(img_name)
        plt.show()
        print('Done')

```

In [21]:

```
X_pre_tsne = train.drop(['subject', 'Activity','ActivityName'], axis=1)
y_pre_tsne = train['ActivityName']
perplexities =[2,5,10,20,50]

for index,perplexity in enumerate(perplexities):
    show_tsne(perplexity,X_data = X_pre_tsne,y_data=y_pre_tsne)
```

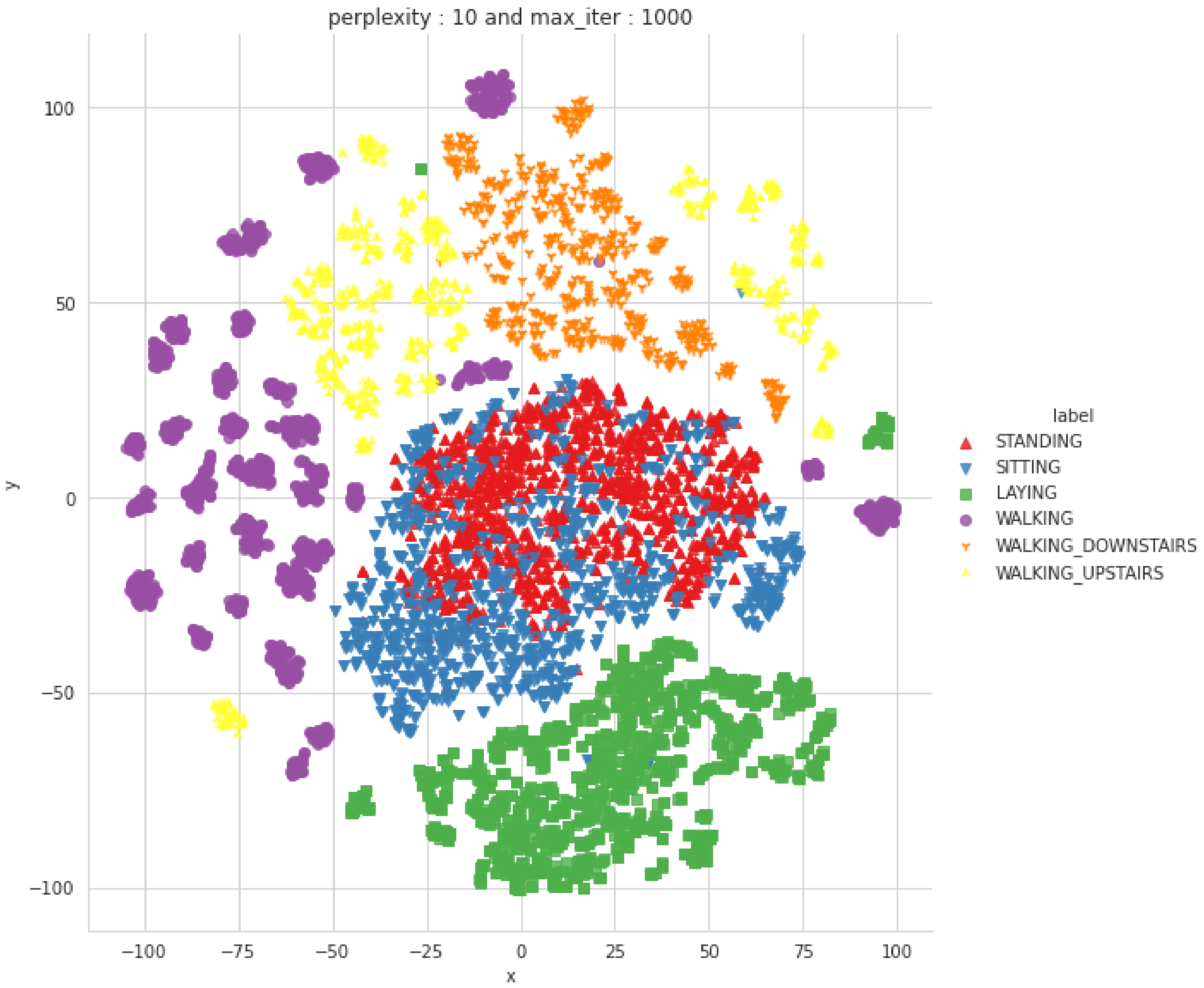
```
performing tsne with perplexity 2 and with 1000 iterations at max
[t-SNE] Computing 7 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.011s...
[t-SNE] Computed neighbors for 7352 samples in 3.485s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 0.635854
[t-SNE] Computed conditional probabilities in 0.061s
[t-SNE] Iteration 50: error = 124.7481995, gradient norm = 0.0241895 (50 iterations in 15.548s)
[t-SNE] Iteration 100: error = 107.1610794, gradient norm = 0.0287191 (50 iterations in 8.234s)
[t-SNE] Iteration 150: error = 100.7689590, gradient norm = 0.0195086 (50 iterations in 9.815s)
[t-SNE] Iteration 200: error = 97.4031982, gradient norm = 0.0188301 (50 iterations in 8.449s)
[t-SNE] Iteration 250: error = 95.1223755, gradient norm = 0.0143584 (50 iterations in 8.244s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 95.122375
[t-SNE] Iteration 300: error = 4.1157804, gradient norm = 0.0015629 (50 iterations in 5.464s)
[t-SNE] Iteration 350: error = 3.2080805, gradient norm = 0.0010077 (50 iterations in 4.559s)
[t-SNE] Iteration 400: error = 2.7786548, gradient norm = 0.0007186 (50 iterations in 5.337s)
[t-SNE] Iteration 450: error = 2.5147719, gradient norm = 0.0005656 (50 iterations in 5.052s)
[t-SNE] Iteration 500: error = 2.3308914, gradient norm = 0.0004854 (50 iterations in 4.040s)
[t-SNE] Iteration 550: error = 2.1930585, gradient norm = 0.0004143 (50 iterations in 6.483s)
[t-SNE] Iteration 600: error = 2.0837677, gradient norm = 0.0003667 (50 iterations in 3.388s)
[t-SNE] Iteration 650: error = 1.9938438, gradient norm = 0.0003352 (50 iterations in 3.929s)
[t-SNE] Iteration 700: error = 1.9183609, gradient norm = 0.0002997 (50 iterations in 5.978s)
[t-SNE] Iteration 750: error = 1.8535386, gradient norm = 0.0002770 (50 iterations in 8.058s)
[t-SNE] Iteration 800: error = 1.7969873, gradient norm = 0.0002558 (50 iterations in 3.695s)
[t-SNE] Iteration 850: error = 1.7469370, gradient norm = 0.0002387 (50 iterations in 2.996s)
[t-SNE] Iteration 900: error = 1.7023004, gradient norm = 0.0002266 (50 iterations in 4.374s)
[t-SNE] Iteration 950: error = 1.6621580, gradient norm = 0.0002098 (50 iterations in 3.225s)
[t-SNE] Iteration 1000: error = 1.6260192, gradient norm = 0.0001997 (50 iterations in 6.591s)
[t-SNE] KL divergence after 1000 iterations: 1.626019
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```


perplexity : 2 and max_iter : 1000
100
50
0
−50
−100
y
−100
−50
0
50
100
x
label
STANDING
SITTING
LAYING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS

```
Done

performing tsne with perplexity 5 and with 1000 iterations at max
[t-SNE] Computing 16 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.027s...
[t-SNE] Computed neighbors for 7352 samples in 3.124s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 0.961265
[t-SNE] Computed conditional probabilities in 0.110s
[t-SNE] Iteration 50: error = 113.8799591, gradient norm = 0.0222652 (50 iterations in 6.202s)
[t-SNE] Iteration 100: error = 98.0514526, gradient norm = 0.0167063 (50 iterations in 4.281s)
[t-SNE] Iteration 150: error = 93.2212448, gradient norm = 0.0092625 (50 iterations in 8.665s)
[t-SNE] Iteration 200: error = 91.1227722, gradient norm = 0.0069251 (50 iterations in 6.720s)
[t-SNE] Iteration 250: error = 89.9272690, gradient norm = 0.0058169 (50 iterations in 4.582s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 89.927269
[t-SNE] Iteration 300: error = 3.5702732, gradient norm = 0.0014615 (50 iterations in 7.480s)
[t-SNE] Iteration 350: error = 2.8129020, gradient norm = 0.0007539 (50 iterations in 6.124s)
[t-SNE] Iteration 400: error = 2.4331491, gradient norm = 0.0005248 (50 iterations in 5.643s)
[t-SNE] Iteration 450: error = 2.2162507, gradient norm = 0.0004032 (50 iterations in 3.685s)
[t-SNE] Iteration 500: error = 2.0715027, gradient norm = 0.0003394 (50 iterations in 7.865s)
[t-SNE] Iteration 550: error = 1.9659839, gradient norm = 0.0002864 (50 iterations in 4.150s)
[t-SNE] Iteration 600: error = 1.8851894, gradient norm = 0.0002455 (50 iterations in 3.601s)
[t-SNE] Iteration 650: error = 1.8203343, gradient norm = 0.0002179 (50 iterations in 5.335s)
[t-SNE] Iteration 700: error = 1.7667348, gradient norm = 0.0001973 (50 iterations in 5.542s)
[t-SNE] Iteration 750: error = 1.7211698, gradient norm = 0.0001807 (50 iterations in 7.751s)
[t-SNE] Iteration 800: error = 1.6821179, gradient norm = 0.0001650 (50 iterations in 6.928s)
[t-SNE] Iteration 850: error = 1.6483618, gradient norm = 0.0001520 (50 iterations in 7.055s)
[t-SNE] Iteration 900: error = 1.6186459, gradient norm = 0.0001435 (50 iterations in 7.633s)
[t-SNE] Iteration 950: error = 1.5924865, gradient norm = 0.0001330 (50 iterations in 6.766s)
[t-SNE] Iteration 1000: error = 1.5688016, gradient norm = 0.0001266 (50 iterations in 5.753s)
[t-SNE] KL divergence after 1000 iterations: 1.568802
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```

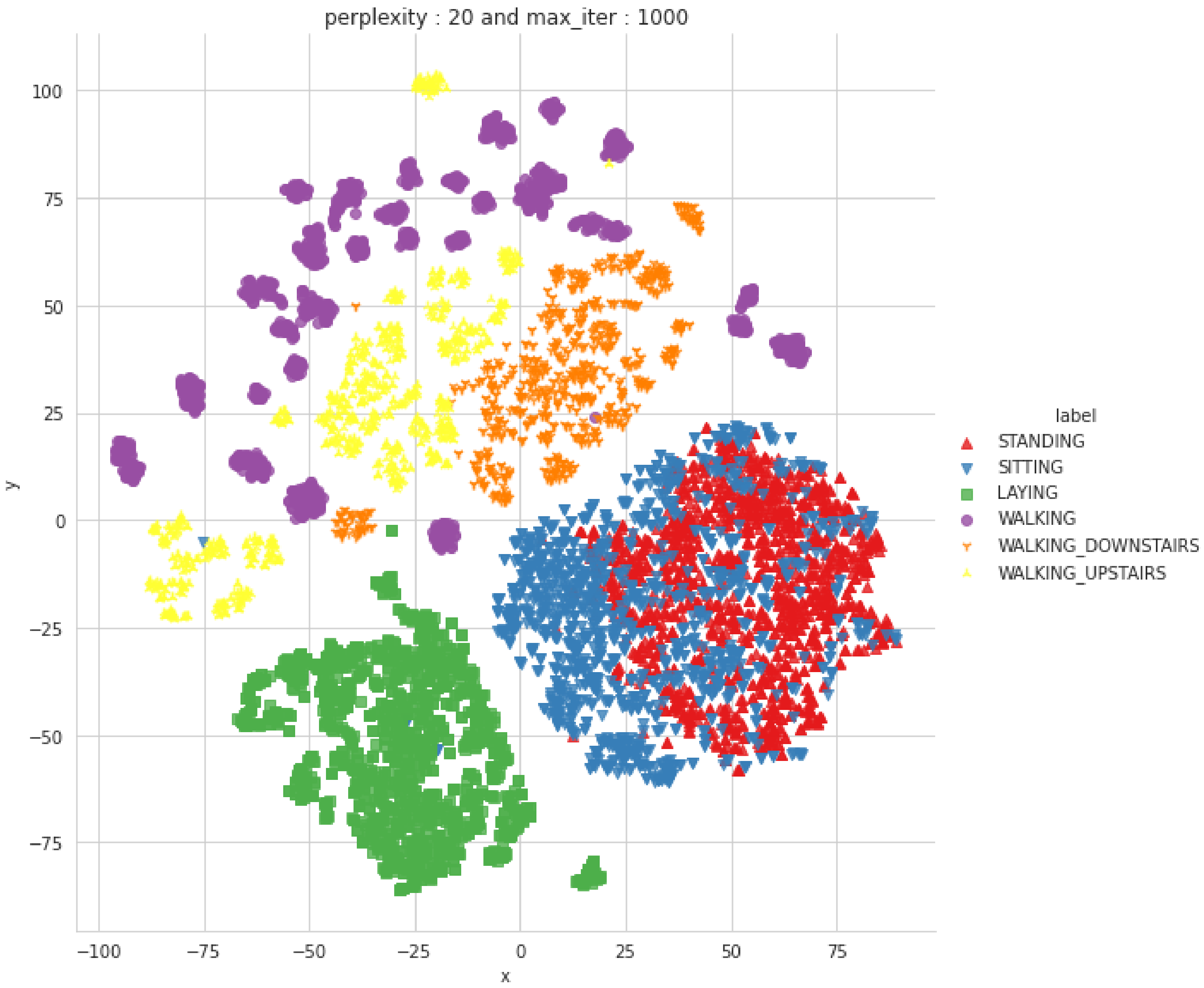

```
Done

performing tsne with perplexity 10 and with 1000 iterations at max
[t-SNE] Computing 31 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.012s...
[t-SNE] Computed neighbors for 7352 samples in 3.801s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
```

```
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.133828
[t-SNE] Computed conditional probabilities in 0.228s
[t-SNE] Iteration 50: error = 105.4448395, gradient norm = 0.0251430 (50 iterations in 7.548s)
[t-SNE] Iteration 100: error = 90.6870880, gradient norm = 0.0092564 (50 iterations in 9.102s)
[t-SNE] Iteration 150: error = 87.5686798, gradient norm = 0.0063727 (50 iterations in 7.636s)
[t-SNE] Iteration 200: error = 86.2863998, gradient norm = 0.0037143 (50 iterations in 7.474s)
[t-SNE] Iteration 250: error = 85.5652313, gradient norm = 0.0027280 (50 iterations in 6.162s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.565231
[t-SNE] Iteration 300: error = 3.1453919, gradient norm = 0.0013904 (50 iterations in 5.088s)
[t-SNE] Iteration 350: error = 2.5047314, gradient norm = 0.0006525 (50 iterations in 3.773s)
[t-SNE] Iteration 400: error = 2.1848981, gradient norm = 0.0004264 (50 iterations in 4.144s)
[t-SNE] Iteration 450: error = 2.0003052, gradient norm = 0.0003191 (50 iterations in 4.619s)
[t-SNE] Iteration 500: error = 1.8815640, gradient norm = 0.0002553 (50 iterations in 3.207s)
[t-SNE] Iteration 550: error = 1.7978530, gradient norm = 0.0002137 (50 iterations in 4.458s)
[t-SNE] Iteration 600: error = 1.7349135, gradient norm = 0.0001830 (50 iterations in 5.898s)
[t-SNE] Iteration 650: error = 1.6852351, gradient norm = 0.0001608 (50 iterations in 3.832s)
[t-SNE] Iteration 700: error = 1.6455531, gradient norm = 0.0001450 (50 iterations in 5.450s)
[t-SNE] Iteration 750: error = 1.6125048, gradient norm = 0.0001309 (50 iterations in 6.087s)
[t-SNE] Iteration 800: error = 1.5848947, gradient norm = 0.0001207 (50 iterations in 5.435s)
[t-SNE] Iteration 850: error = 1.5612959, gradient norm = 0.0001112 (50 iterations in 5.172s)
[t-SNE] Iteration 900: error = 1.5406760, gradient norm = 0.0001038 (50 iterations in 5.930s)
[t-SNE] Iteration 950: error = 1.5226164, gradient norm = 0.0000960 (50 iterations in 3.116s)
[t-SNE] Iteration 1000: error = 1.5068566, gradient norm = 0.0000927 (50 iterations in 4.112s)
[t-SNE] KL divergence after 1000 iterations: 1.506857
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```



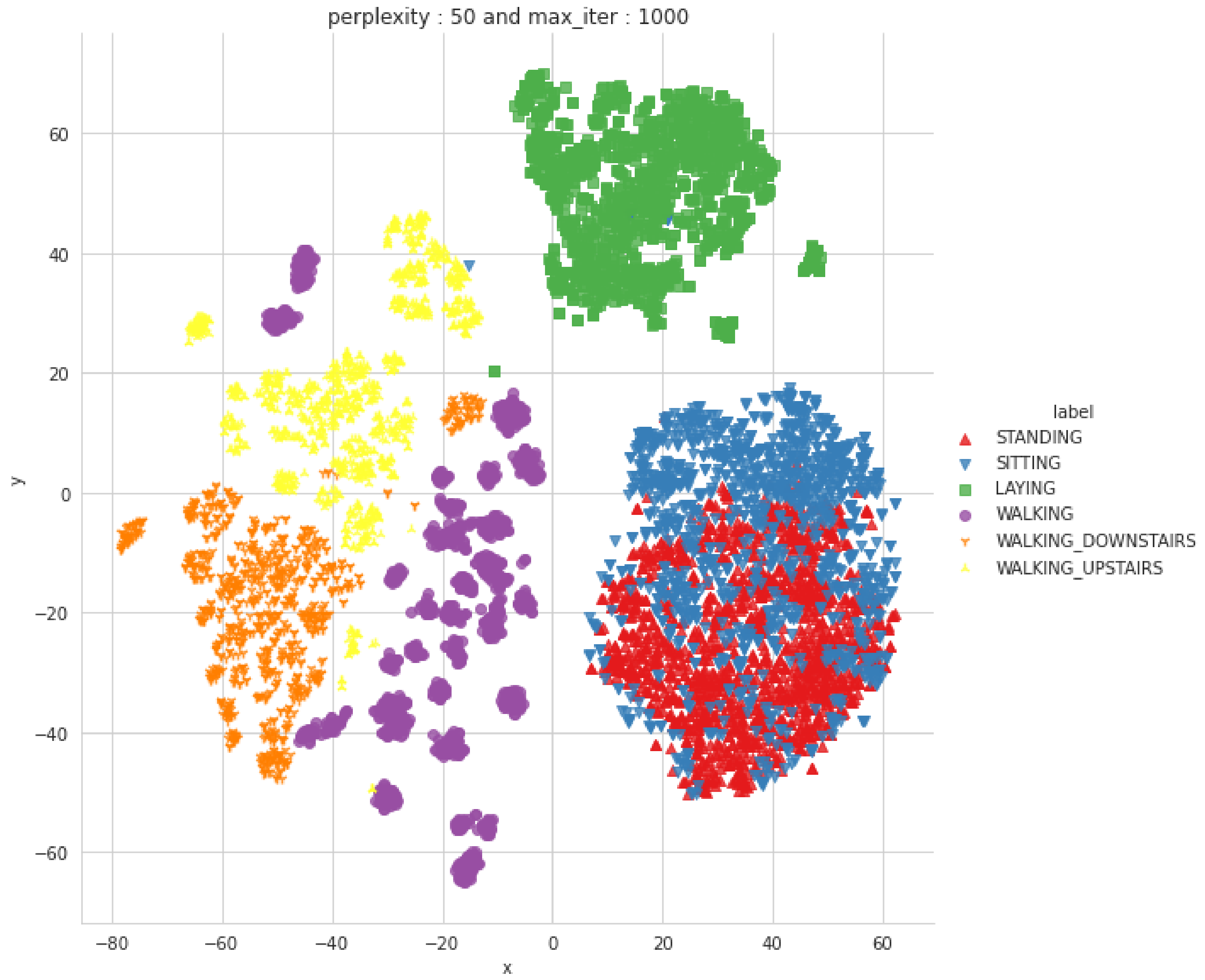

```
Done

performing tsne with perplexity 20 and with 1000 iterations at max
[t-SNE] Computing 61 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.009s...
[t-SNE] Computed neighbors for 7352 samples in 3.672s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.274335
```

```
[t-SNE] Computed conditional probabilities in 0.419s
[t-SNE] Iteration 50: error = 97.5241547, gradient norm = 0.0189147 (50 iterations in 5.782s)
[t-SNE] Iteration 100: error = 84.0375824, gradient norm = 0.0068614 (50 iterations in 3.715s)
[t-SNE] Iteration 150: error = 81.9335175, gradient norm = 0.0049976 (50 iterations in 4.355s)
[t-SNE] Iteration 200: error = 81.1859741, gradient norm = 0.0026055 (50 iterations in 4.134s)
[t-SNE] Iteration 250: error = 80.8040237, gradient norm = 0.0021426 (50 iterations in 3.676s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 80.804024
[t-SNE] Iteration 300: error = 2.6995583, gradient norm = 0.0013018 (50 iterations in 4.197s)
[t-SNE] Iteration 350: error = 2.1656680, gradient norm = 0.0005773 (50 iterations in 4.495s)
[t-SNE] Iteration 400: error = 1.9159158, gradient norm = 0.0003462 (50 iterations in 3.933s)
[t-SNE] Iteration 450: error = 1.7694576, gradient norm = 0.0002484 (50 iterations in 4.117s)
[t-SNE] Iteration 500: error = 1.6754386, gradient norm = 0.0001923 (50 iterations in 4.450s)
[t-SNE] Iteration 550: error = 1.6107479, gradient norm = 0.0001589 (50 iterations in 4.851s)
[t-SNE] Iteration 600: error = 1.5641252, gradient norm = 0.0001353 (50 iterations in 4.009s)
[t-SNE] Iteration 650: error = 1.5289489, gradient norm = 0.0001221 (50 iterations in 3.679s)
[t-SNE] Iteration 700: error = 1.5018971, gradient norm = 0.0001081 (50 iterations in 5.355s)
[t-SNE] Iteration 750: error = 1.4807789, gradient norm = 0.0000976 (50 iterations in 5.341s)
[t-SNE] Iteration 800: error = 1.4634119, gradient norm = 0.0000900 (50 iterations in 4.690s)
[t-SNE] Iteration 850: error = 1.4488745, gradient norm = 0.0000845 (50 iterations in 3.863s)
[t-SNE] Iteration 900: error = 1.4368867, gradient norm = 0.0000816 (50 iterations in 4.684s)
[t-SNE] Iteration 950: error = 1.4270021, gradient norm = 0.0000770 (50 iterations in 4.339s)
[t-SNE] Iteration 1000: error = 1.4184731, gradient norm = 0.0000730 (50 iterations in 3.715s)
[t-SNE] KL divergence after 1000 iterations: 1.418473
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```




```
Done

performing tsne with perplexity 50 and with 1000 iterations at max
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.008s...
[t-SNE] Computed neighbors for 7352 samples in 3.365s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.437672
[t-SNE] Computed conditional probabilities in 1.011s
[t-SNE] Iteration 50: error = 86.4537201, gradient norm = 0.0218618 (50 iterations in 7.428s)
[t-SNE] Iteration 100: error = 75.5895767, gradient norm = 0.0042839 (50 iterations in 6.650s)
[t-SNE] Iteration 150: error = 74.6346283, gradient norm = 0.0024554 (50 iterations in 6.863s)
[t-SNE] Iteration 200: error = 74.2932053, gradient norm = 0.0017026 (50 iterations in 4.487s)
[t-SNE] Iteration 250: error = 74.1198044, gradient norm = 0.0011326 (50 iterations in 6.315s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 74.119804
```

```
[t-SNE] Iteration 300: error = 2.1550534, gradient norm = 0.0011979 (50 iterations in 4.857s)
[t-SNE] Iteration 350: error = 1.7579156, gradient norm = 0.0004841 (50 iterations in 4.639s)
[t-SNE] Iteration 400: error = 1.5889353, gradient norm = 0.0002822 (50 iterations in 6.949s)
[t-SNE] Iteration 450: error = 1.4947137, gradient norm = 0.0001911 (50 iterations in 7.407s)
[t-SNE] Iteration 500: error = 1.4355956, gradient norm = 0.0001418 (50 iterations in 5.642s)
[t-SNE] Iteration 550: error = 1.3943672, gradient norm = 0.0001147 (50 iterations in 6.741s)
[t-SNE] Iteration 600: error = 1.3650140, gradient norm = 0.0000953 (50 iterations in 6.386s)
[t-SNE] Iteration 650: error = 1.3437411, gradient norm = 0.0000842 (50 iterations in 3.995s)
[t-SNE] Iteration 700: error = 1.3287430, gradient norm = 0.0000771 (50 iterations in 4.570s)
[t-SNE] Iteration 750: error = 1.3181075, gradient norm = 0.0000705 (50 iterations in 4.657s)
[t-SNE] Iteration 800: error = 1.3099051, gradient norm = 0.0000670 (50 iterations in 4.524s)
[t-SNE] Iteration 850: error = 1.3029939, gradient norm = 0.0000626 (50 iterations in 4.796s)
[t-SNE] Iteration 900: error = 1.2973118, gradient norm = 0.0000604 (50 iterations in 4.114s)
[t-SNE] Iteration 950: error = 1.2927370, gradient norm = 0.0000600 (50 iterations in 3.880s)
[t-SNE] Iteration 1000: error = 1.2890576, gradient norm = 0.0000534 (50 iterations in 4.365s)
[t-SNE] KL divergence after 1000 iterations: 1.289058
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```



```
Done
```

Observations :

- We can see that as we increase our perplexities all the lables except standing and sitting positions are separted well.
- Standing and sitting position being a dynamic label overlaps with one another despite perplexity alters.

2. Building Statistical models and training with the domain engineered features

In [22]:
```
# get X_train and y_train from csv files
X_train = train.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_train = train['ActivityName']
# get X_test and y_test from test csv file
X_test = test.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_test = test['ActivityName']
X_train.shape, y_train.shape,X_test.shape, y_test.shape
```

Out[22]:
```
((7352, 561), (7352,), (2947, 561), (2947,))
```

In [26]:

```python
def plot_confusion_matrix(y_test,y_pred):
    conf_mat = confusion_matrix_(y_test, y_pred)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j
    A =(((conf_mat.T)/(conf_mat.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that column

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                           [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                           [3/7, 4/7]]
    # sum of row elements = 1

    B =(conf_mat/conf_mat.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]

    labels = ["WALKING", "WALKING_UPSTAIRS", "WALKING_DOWNSTAIRS", "SITTING", "STANDING", "LYING"]
    # representing A in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(conf_mat, annot=True, cmap="YlOrBr", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.xticks(rotation = 90)
    plt.show()

    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(B, annot=True, cmap="YlOrBr", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.xticks(rotation = 90)
    plt.show()

    # representing B in heatmap format
    print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(A, annot=True, cmap="YlOrBr", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.xticks(rotation = 90)
    plt.show()

```

In [24]:

```
### craete a dataframe to store model scores
df = pd.DataFrame(columns=['Model_name','Accuracy%'])

def save_performances(model_name,accuracy):
    global df
    df = df.append(pd.DataFrame([[model_name,accuracy]],columns=['Model_name','Accuracy%']))
    #df = df.reset_index(drop=True,inplace=True)
```

In [25]:

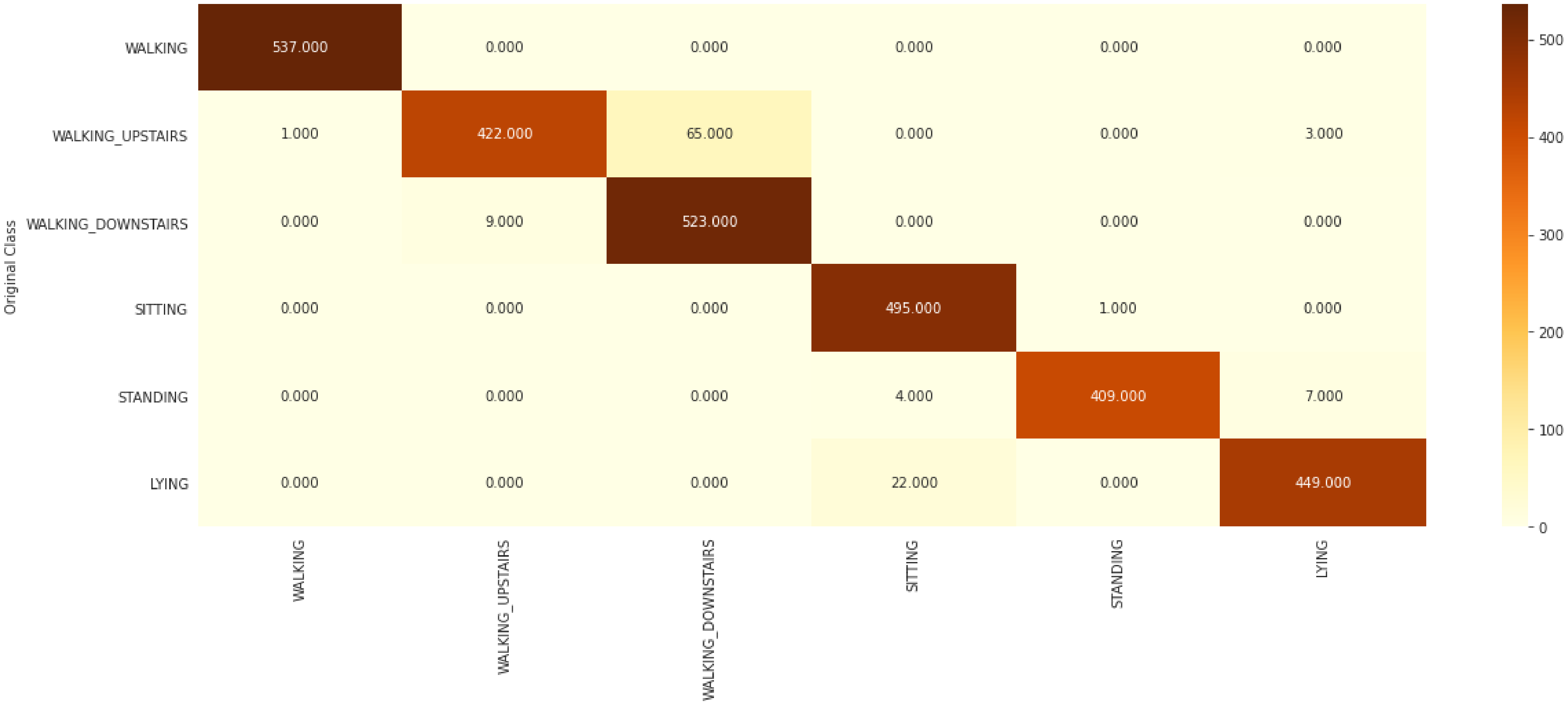
```
def run_model(model,train,ytrain,test,ytest,model_name):

    ##train data
    train_start = datetime.now()
    model.fit(train,ytrain)
    print('Time taken to train the model: ',datetime.now()- train_start)
    print('Done \n \n')

    ## print best params from grid search
    print('*'*20)
    print('Best-Params(Grid_search):  ')
    print('*'*20)
    print('params of best estimator: ',model.best_params_)

    print('*'*50)
    print('Best-Score(Grid_search):  ')
    print('*'*50)
    print('Score of best estimator: ',model.best_score_)
    print('/n/n')
    print('Training model: ',model)
    ## fit model with best params
    model = model.best_estimator_.fit(train,ytrain)
    ##predict data
    y_pred_tr = model.predict(train)
    y_pred_te = model.predict(test)
    train_acc = metrics.accuracy_score(y_true=ytrain, y_pred=y_pred_tr)
    test_acc = metrics.accuracy_score(y_true=ytest, y_pred=y_pred_te)

    ## calculate train accuracy
    print('*'*50)
    print('Accuracy:  ')
    print('*'*50)
    print('Train Accuracy:',train_acc,'Test Accuracy',test_acc)
    print('\n\n')
    ##print confusion matrix
    print('*'*50)
    print('Confusion Matrix :test')
    print('*'*50)
    plot_confusion_matrix(ytest,y_pred_te)
    ##store in df
    save_performances(model_name,test_acc)
```

2.1. Logistic Regression with Grid Search

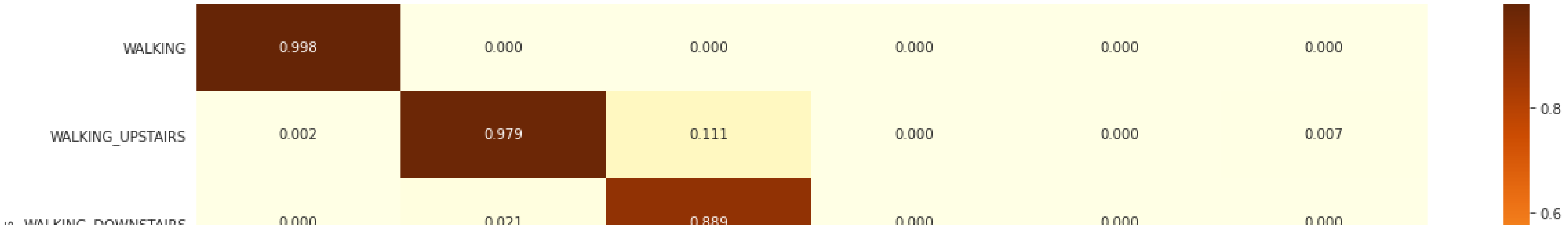
In [27]:

```
# start Grid search
parameters = {'C':[0.01, 0.1, 1, 10, 20, 30], 'penalty':['l2','l1']}
log_reg = linear_model.LogisticRegression(multi_class = "ovr")
log_reg_grid = GridSearchCV(log_reg, param_grid=parameters, cv=3, verbose=1, n_jobs=-1)
log_reg_grid_results =  run_model(log_reg_grid, X_train, y_train, X_test, y_test,'LogisticRegression')
```

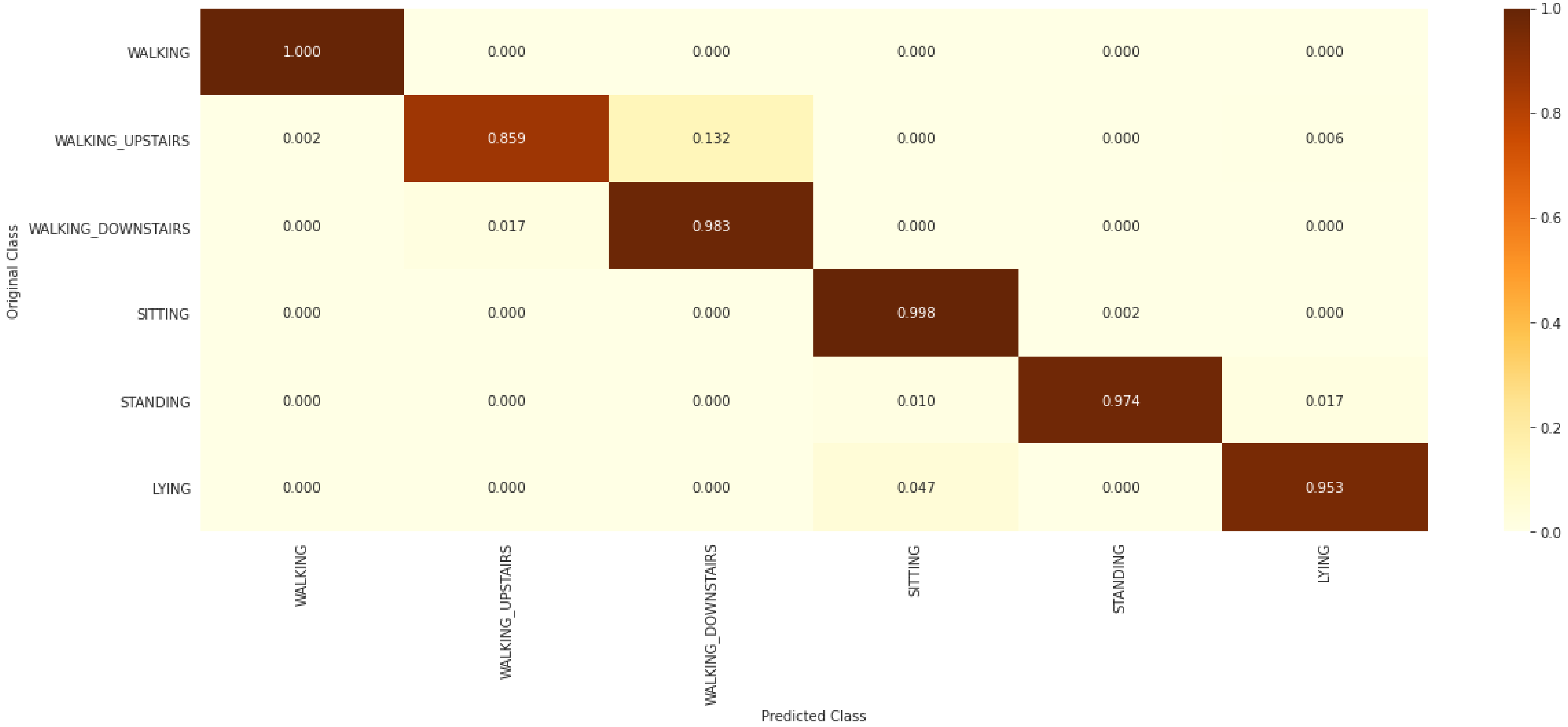
```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Time taken to train the model:  0:00:56.821896
Done


********************
Best-Params(Grid_search):
********************
params of best estimator:  {'C': 30, 'penalty': 'l2'}
**************************************************
Best-Score(Grid_search):
**************************************************
Score of best estimator:  0.9449151116995146
/n/n
Training model:  GridSearchCV(cv=3, estimator=LogisticRegression(multi_class='ovr'), n_jobs=-1,
             param_grid={'C': [0.01, 0.1, 1, 10, 20, 30],
                         'penalty': ['l2', 'l1']},
             verbose=1)
**************************************************
Accuracy:
**************************************************
Train Accuracy: 0.9944232861806311 Test Accuracy 0.9619952494061758



**************************************************
Confusion Matrix :test
**************************************************
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
------------------- Recall matrix (Row sum=1) -------------------
```



2.2 Linear SVC with GridSearch

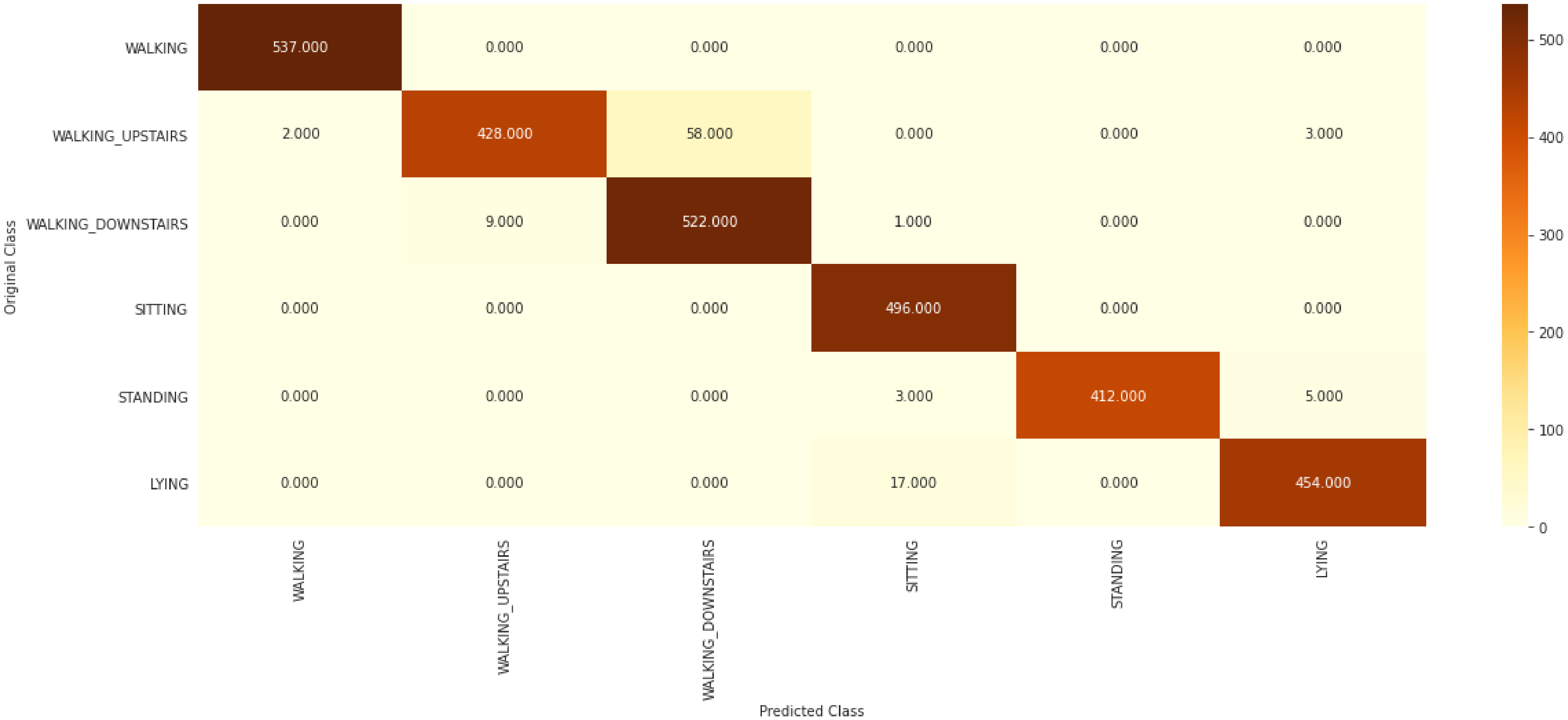
In [28]:

```
from sklearn.svm import LinearSVC
```

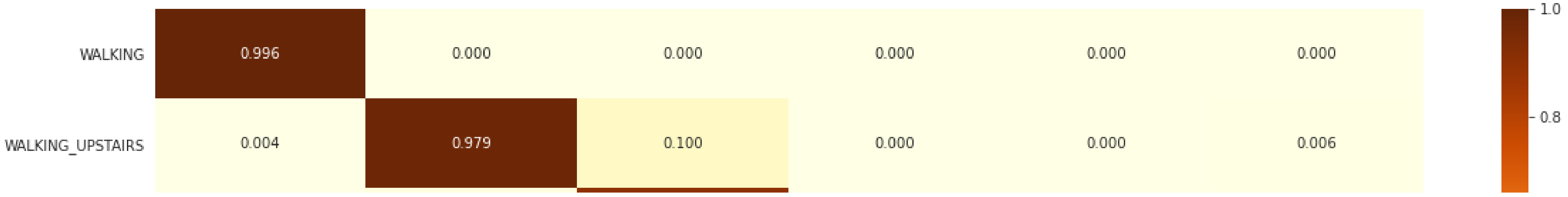
In [29]:

```python
parameters = {'C':[0.125, 0.5, 1, 2, 8, 16]}
lr_svc = LinearSVC(tol=0.00005)
lr_svc_grid = GridSearchCV(lr_svc, param_grid=parameters, n_jobs=-1, verbose=1)
lr_svc_grid_results = run_model(lr_svc_grid, X_train, y_train, X_test, y_test,'Linear SVC')
```

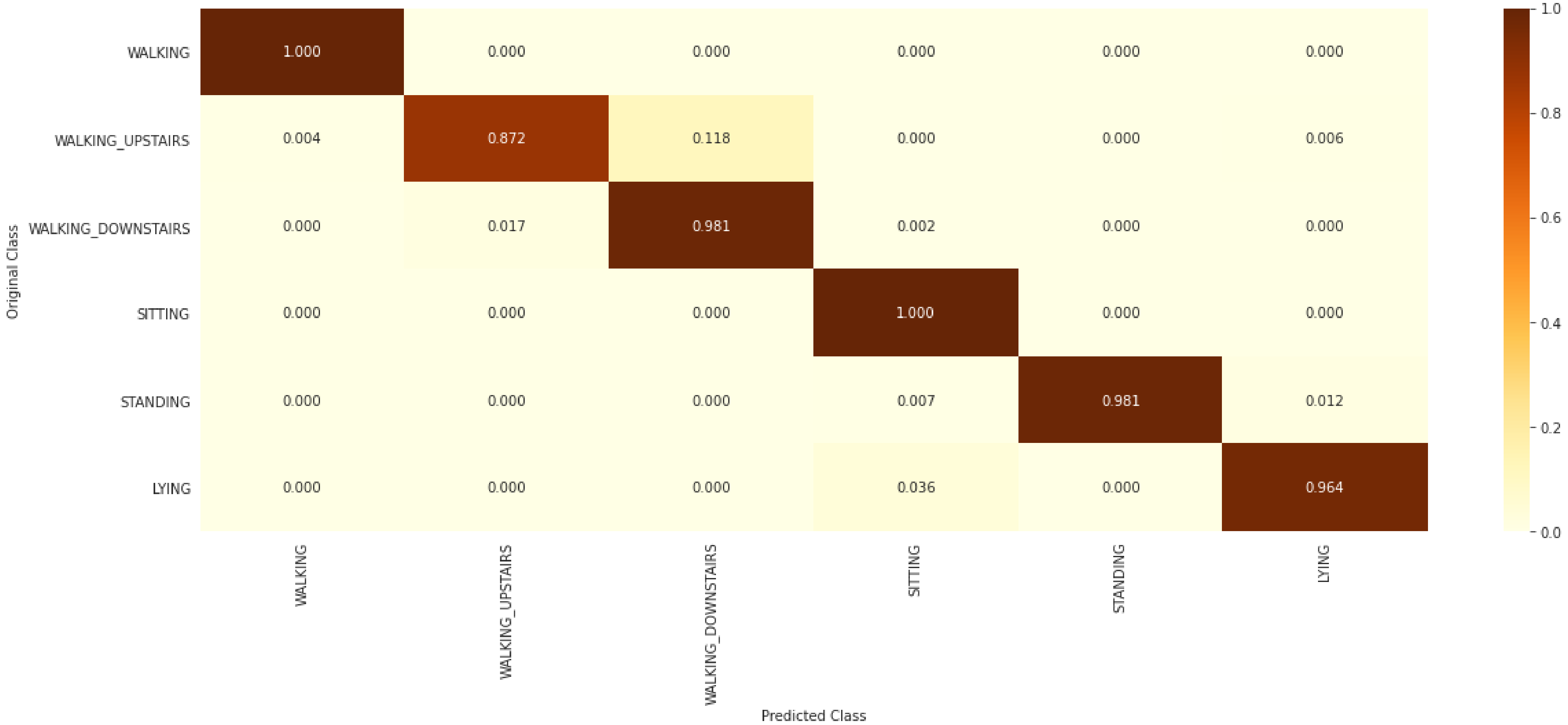
```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Time taken to train the model:  0:01:05.836804
Done


********************
Best-Params(Grid_search):
********************
params of best estimator:  {'C': 0.5}
**************************************************
Best-Score(Grid_search):
**************************************************
Score of best estimator:  0.9423363254207189
/n/n
Training model:  GridSearchCV(estimator=LinearSVC(tol=5e-05), n_jobs=-1,
             param_grid={'C': [0.125, 0.5, 1, 2, 8, 16]}, verbose=1)
**************************************************
Accuracy:
**************************************************
Train Accuracy: 0.9944232861806311 Test Accuracy 0.9667458432304038



**************************************************
Confusion Matrix :test
**************************************************
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```

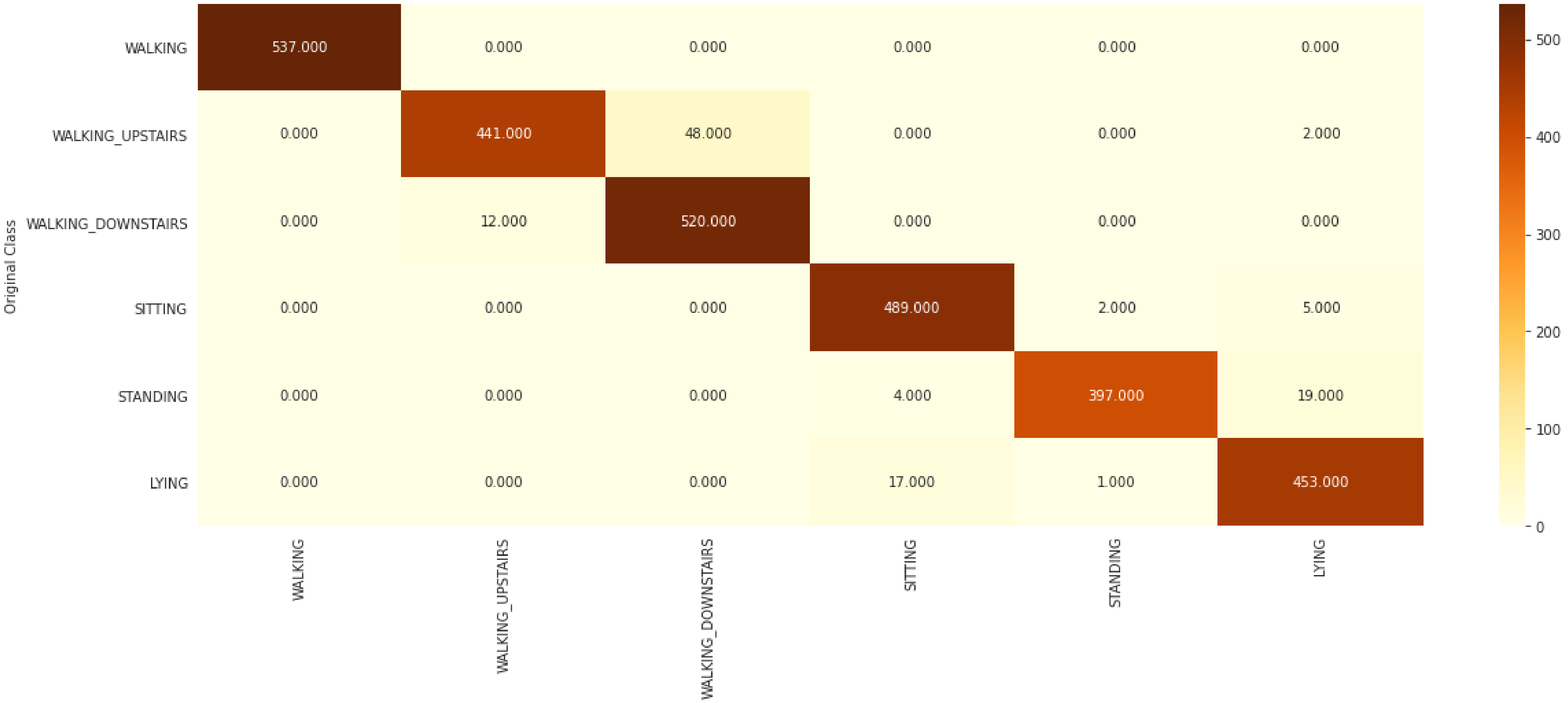


2.3. Kernel SVM with GridSearch

In [30]:

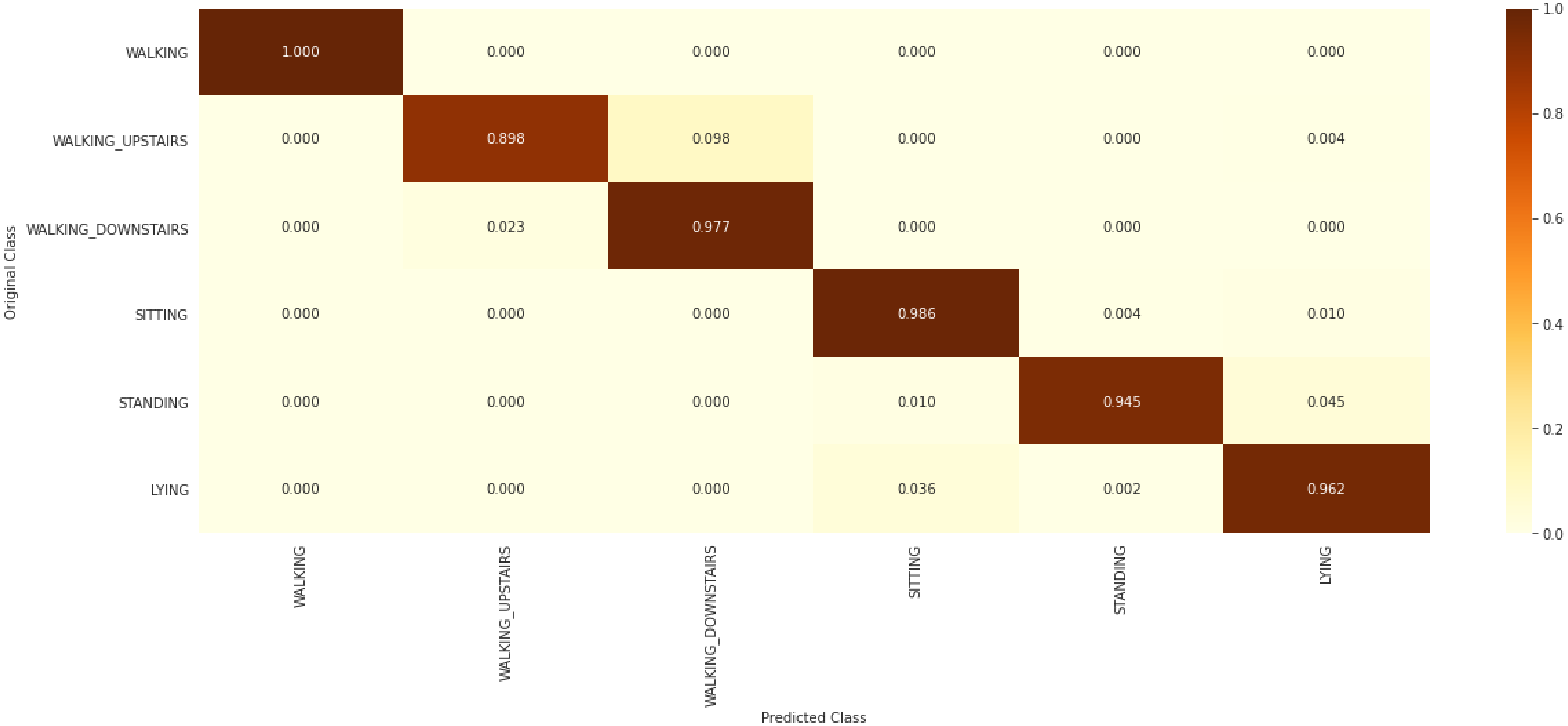
```
from sklearn.svm import SVC
parameters = {'C':[2,8,16],\
              'gamma': [ 0.0078125, 0.125, 2]}
rbf_svm = SVC(kernel='rbf')
rbf_svm_grid = GridSearchCV(rbf_svm,param_grid=parameters, n_jobs=-1)
rbf_svm_grid_results = run_model(rbf_svm_grid, X_train, y_train, X_test, y_test, 'Kernel SVM')
```

```
Time taken to train the model:  0:08:16.665605
Done


********************
Best-Params(Grid_search):
********************
params of best estimator:  {'C': 16, 'gamma': 0.0078125}
**************************************************
Best-Score(Grid_search):
**************************************************
Score of best estimator:  0.9447834551903698
/n/n
Training model:  GridSearchCV(estimator=SVC(), n_jobs=-1,
             param_grid={'C': [2, 8, 16], 'gamma': [0.0078125, 0.125, 2]})
**************************************************
Accuracy:
**************************************************
Train Accuracy: 0.9964635473340587 Test Accuracy 0.9626739056667798



**************************************************
Confusion Matrix :test
**************************************************
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```


```
-------------------- Recall matrix (Row sum=1) --------------------
```



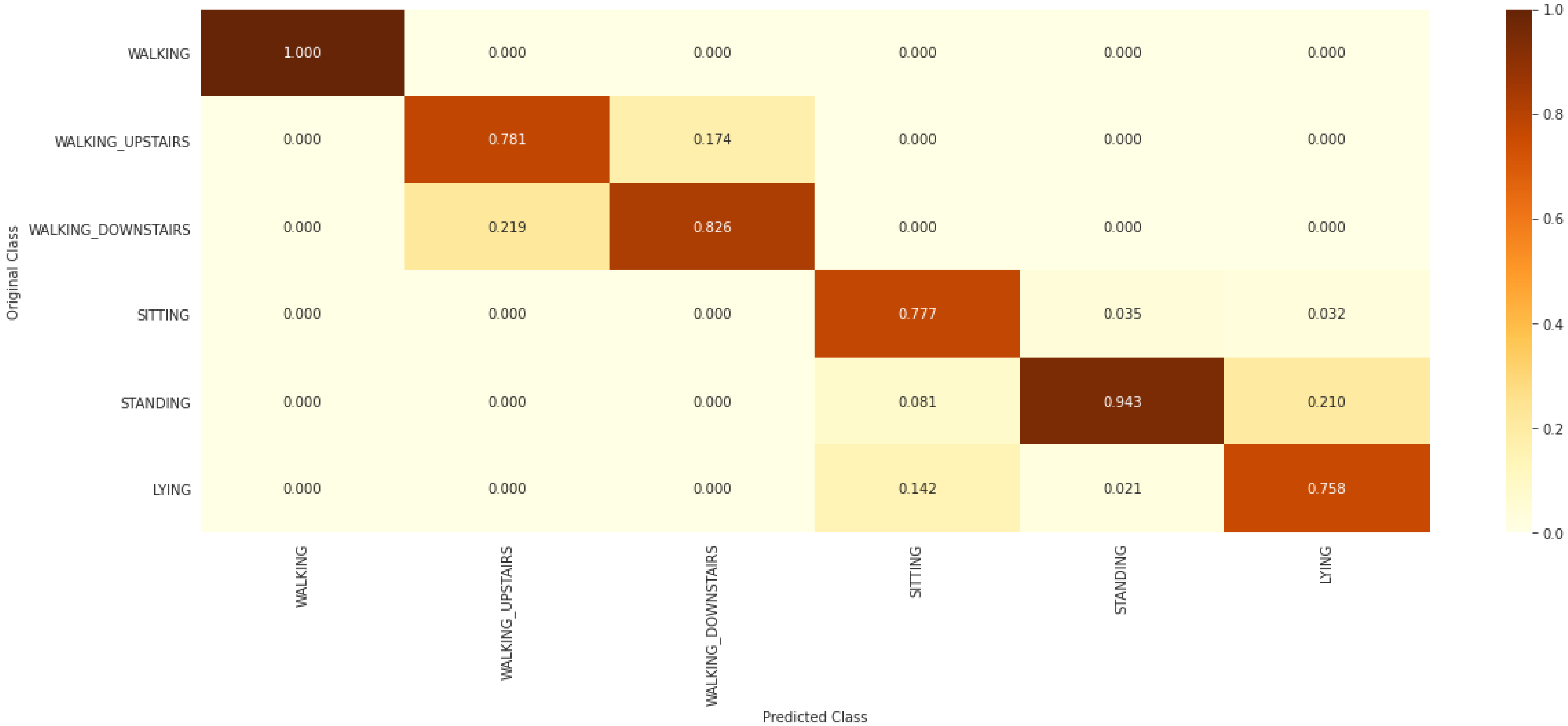
2.4. Decision Trees with GridSearchCV

In [31]:

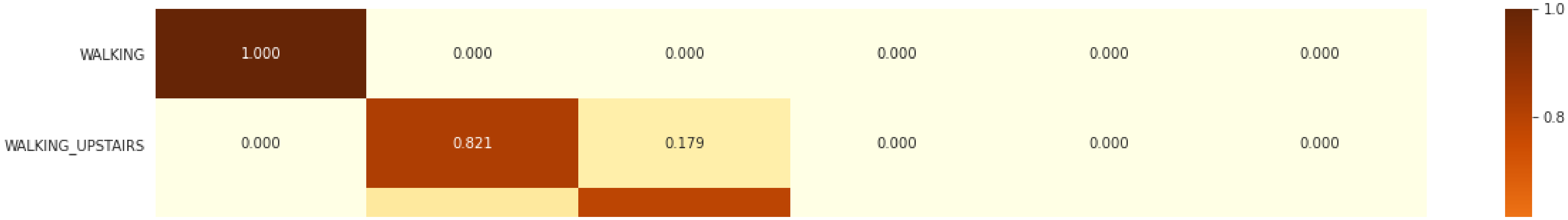
```
from sklearn.tree import DecisionTreeClassifier
parameters = {'max_depth':np.arange(3,10,2)}
dt = DecisionTreeClassifier()
dt_grid = GridSearchCV(dt,param_grid=parameters, n_jobs=-1)
dt_grid_results = run_model(dt_grid, X_train, y_train, X_test, y_test, 'DecisionTrees')
```

```
Time taken to train the model:  0:00:19.908109
Done


********************
Best-Params(Grid_search):
********************
params of best estimator:  {'max_depth': 5}
**************************************************
Best-Score(Grid_search):
**************************************************
Score of best estimator:  0.8514733371254689
/n/n
Training model:  GridSearchCV(estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'max_depth': array([3, 5, 7, 9])})
**************************************************
Accuracy:
**************************************************
Train Accuracy: 0.920429815016322 Test Accuracy 0.839497794367153



**************************************************
Confusion Matrix :test
**************************************************
-------------------- Confusion matrix --------------------
```


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```

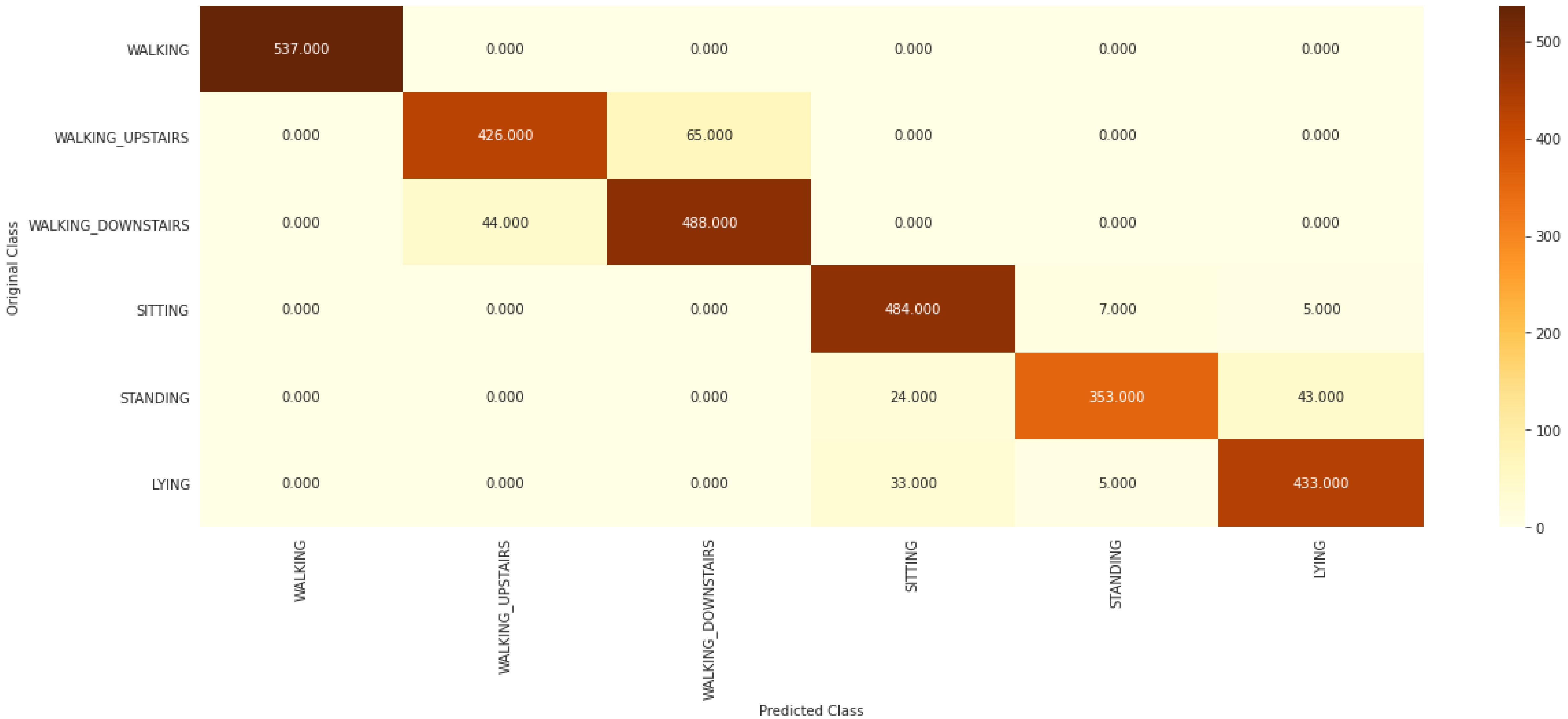


2.5. Random Forest Classifier with GridSearch

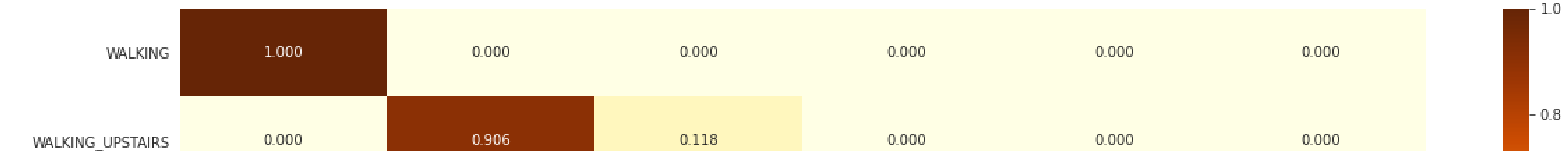
In [32]:

```
from sklearn.ensemble import RandomForestClassifier
params = {'n_estimators': np.arange(10,201,20), 'max_depth':np.arange(3,15,2)}
rfc = RandomForestClassifier()
rfc_grid = GridSearchCV(rfc, param_grid=params, n_jobs=-1)
rfc_grid_results = run_model(rfc_grid, X_train, y_train, X_test, y_test,'RandomForest')
```

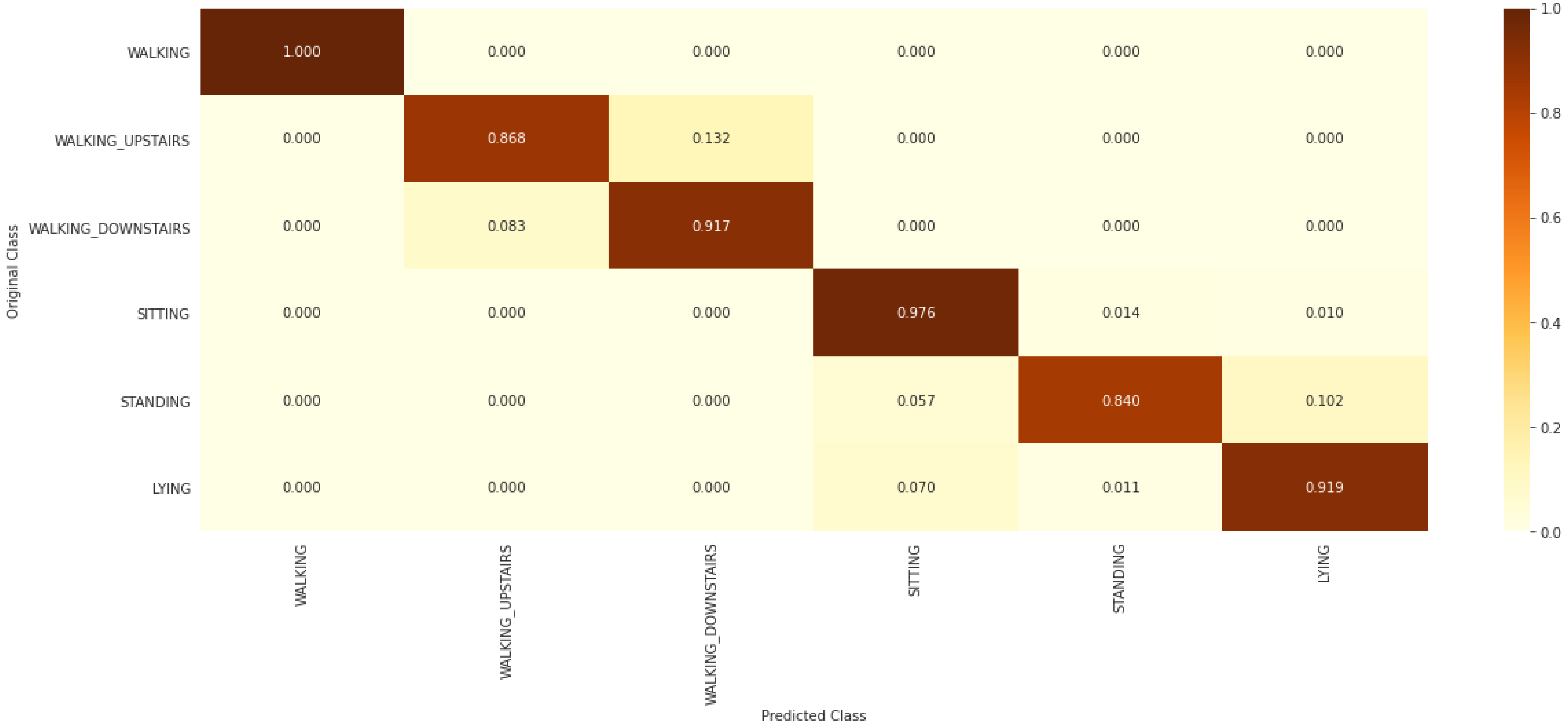
```
Time taken to train the model:  0:10:44.490748
Done


********************
Best-Params(Grid_search):
********************
params of best estimator:  {'max_depth': 13, 'n_estimators': 70}
**************************************************
Best-Score(Grid_search):
**************************************************
Score of best estimator:  0.9212499248509737
/n/n
Training model:  GridSearchCV(estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'max_depth': array([ 3,  5,  7,  9, 11, 13]),
                         'n_estimators': array([ 10,  30,  50,  70,  90, 110, 130, 150, 170, 190])})
**************************************************
Accuracy:
**************************************************
Train Accuracy: 0.999183895538629 Test Accuracy 0.9233118425517476



**************************************************
Confusion Matrix :test
**************************************************
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



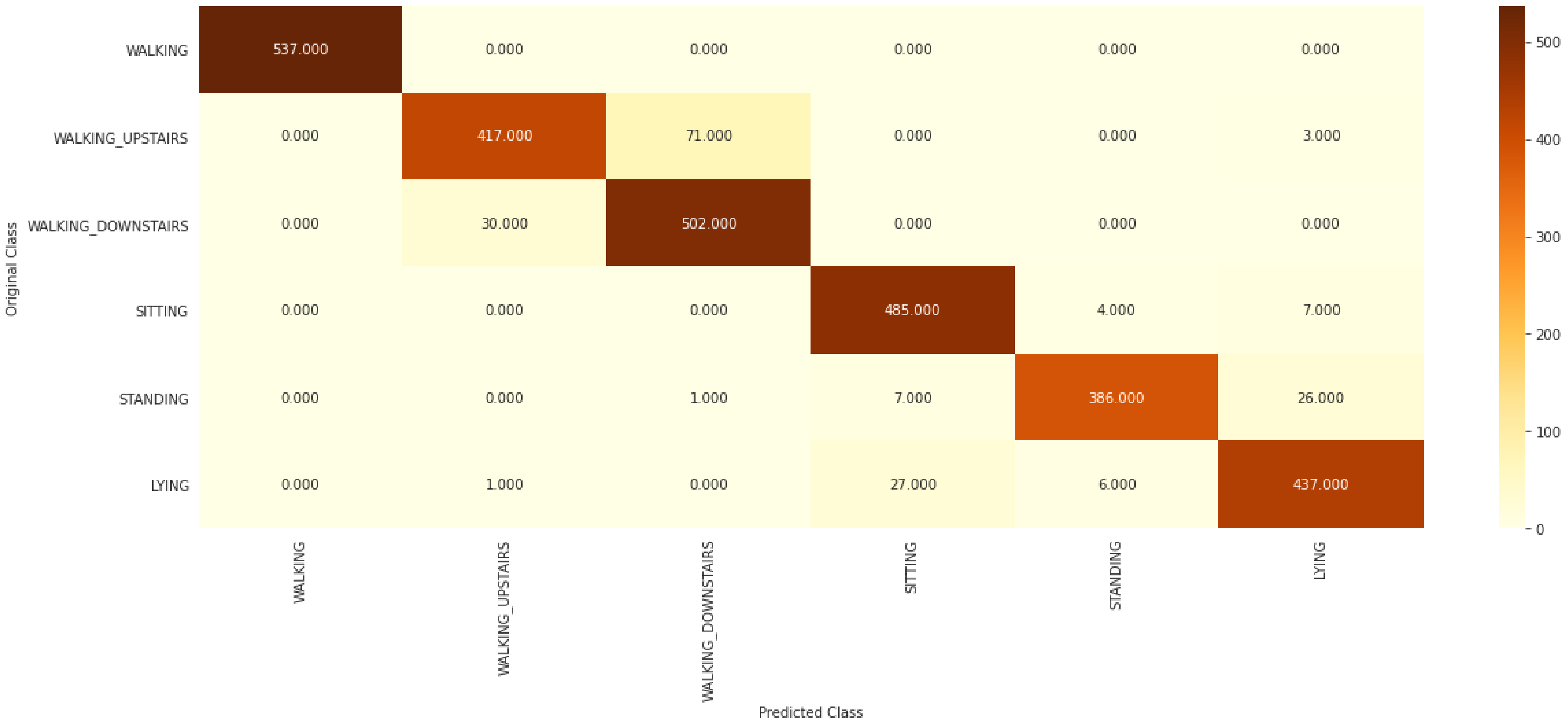
2.6. Gradient Boosted Decision Trees With GridSearch

In [33]:

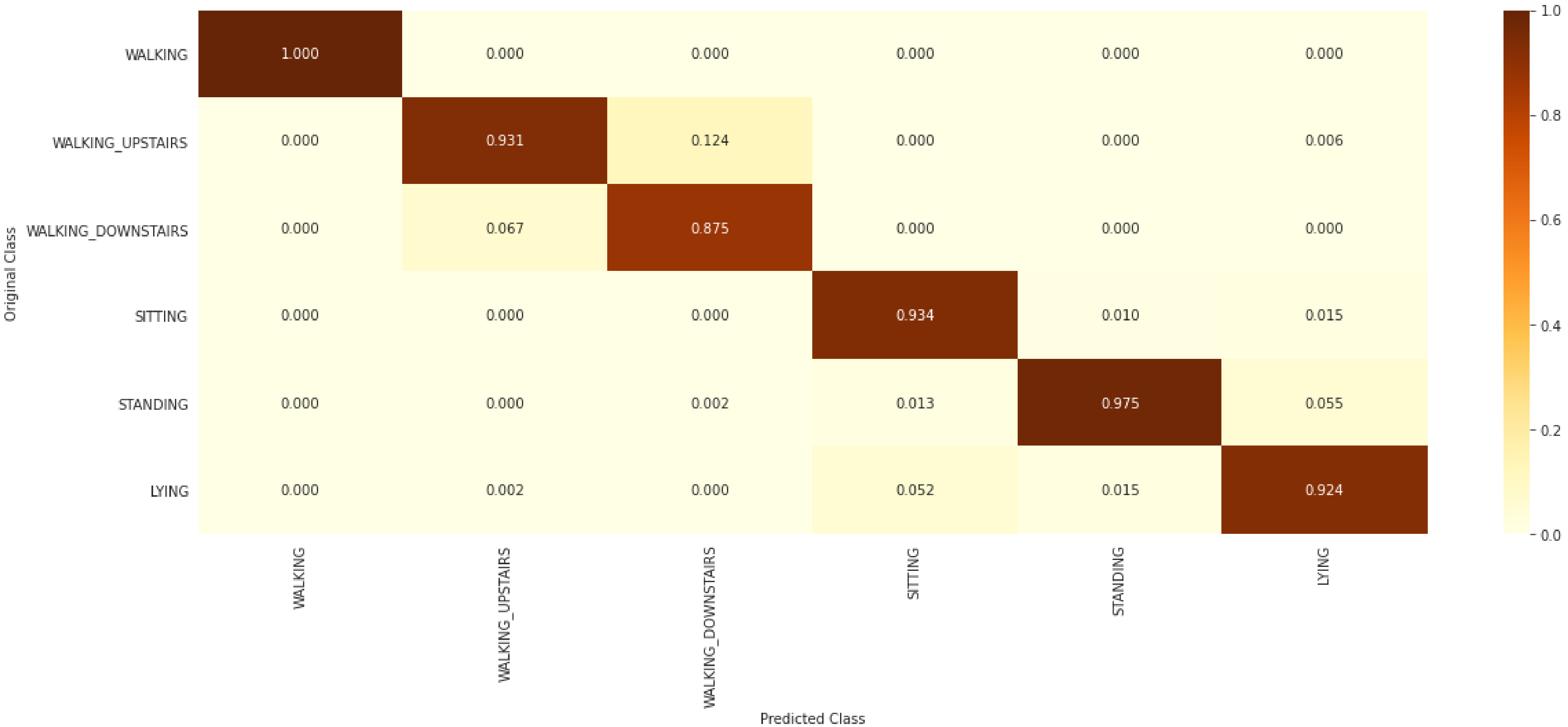
```python
from sklearn.ensemble import GradientBoostingClassifier
param_grid ={"n_estimators": [50, 100], "max_depth":[1, 3]}
gbdt = GradientBoostingClassifier()
gbdt_grid = GridSearchCV(gbdt, param_grid=param_grid,verbose=10)
gbdt_grid_results = run_model(gbdt_grid, X_train, y_train, X_test, y_test,'GradientBoosted DecisionTrees')
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
[CV 1/5; 1/4] START max_depth=1, n_estimators=50...............................
[CV 1/5; 1/4] END .............max_depth=1, n_estimators=50; total time= 1.5min
[CV 2/5; 1/4] START max_depth=1, n_estimators=50...............................
[CV 2/5; 1/4] END .............max_depth=1, n_estimators=50; total time= 1.5min
[CV 3/5; 1/4] START max_depth=1, n_estimators=50...............................
[CV 3/5; 1/4] END .............max_depth=1, n_estimators=50; total time= 1.6min
[CV 4/5; 1/4] START max_depth=1, n_estimators=50...............................
[CV 4/5; 1/4] END .............max_depth=1, n_estimators=50; total time= 1.6min
[CV 5/5; 1/4] START max_depth=1, n_estimators=50...............................
[CV 5/5; 1/4] END .............max_depth=1, n_estimators=50; total time= 1.8min
[CV 1/5; 2/4] START max_depth=1, n_estimators=100..............................
[CV 1/5; 2/4] END ............max_depth=1, n_estimators=100; total time= 3.1min
[CV 2/5; 2/4] START max_depth=1, n_estimators=100..............................
[CV 2/5; 2/4] END ............max_depth=1, n_estimators=100; total time= 3.1min
[CV 3/5; 2/4] START max_depth=1, n_estimators=100..............................
[CV 3/5; 2/4] END ............max_depth=1, n_estimators=100; total time= 3.1min
[CV 4/5; 2/4] START max_depth=1, n_estimators=100..............................
[CV 4/5; 2/4] END ............max_depth=1, n_estimators=100; total time= 3.0min
[CV 5/5; 2/4] START max_depth=1, n_estimators=100..............................
[CV 5/5; 2/4] END ............max_depth=1, n_estimators=100; total time= 3.1min
[CV 1/5; 3/4] START max_depth=3, n_estimators=50...............................
[CV 1/5; 3/4] END .............max_depth=3, n_estimators=50; total time= 4.4min
[CV 2/5; 3/4] START max_depth=3, n_estimators=50...............................
[CV 2/5; 3/4] END .............max_depth=3, n_estimators=50; total time= 4.4min
[CV 3/5; 3/4] START max_depth=3, n_estimators=50...............................
[CV 3/5; 3/4] END .............max_depth=3, n_estimators=50; total time= 4.5min
[CV 4/5; 3/4] START max_depth=3, n_estimators=50...............................
[CV 4/5; 3/4] END .............max_depth=3, n_estimators=50; total time= 4.4min
[CV 5/5; 3/4] START max_depth=3, n_estimators=50...............................
[CV 5/5; 3/4] END .............max_depth=3, n_estimators=50; total time= 4.3min
[CV 1/5; 4/4] START max_depth=3, n_estimators=100..............................
[CV 1/5; 4/4] END ............max_depth=3, n_estimators=100; total time= 8.5min
[CV 2/5; 4/4] START max_depth=3, n_estimators=100..............................
[CV 2/5; 4/4] END ............max_depth=3, n_estimators=100; total time= 8.5min
[CV 3/5; 4/4] START max_depth=3, n_estimators=100..............................
[CV 3/5; 4/4] END ............max_depth=3, n_estimators=100; total time= 8.3min
[CV 4/5; 4/4] START max_depth=3, n_estimators=100..............................
[CV 4/5; 4/4] END ............max_depth=3, n_estimators=100; total time= 8.2min
[CV 5/5; 4/4] START max_depth=3, n_estimators=100..............................
[CV 5/5; 4/4] END ............max_depth=3, n_estimators=100; total time= 8.2min
Time taken to train the model:  1:37:41.198286
Done


********************
Best-Params(Grid_search):
********************
params of best estimator:  {'max_depth': 3, 'n_estimators': 100}
**************************************************
Best-Score(Grid_search):
**************************************************
```

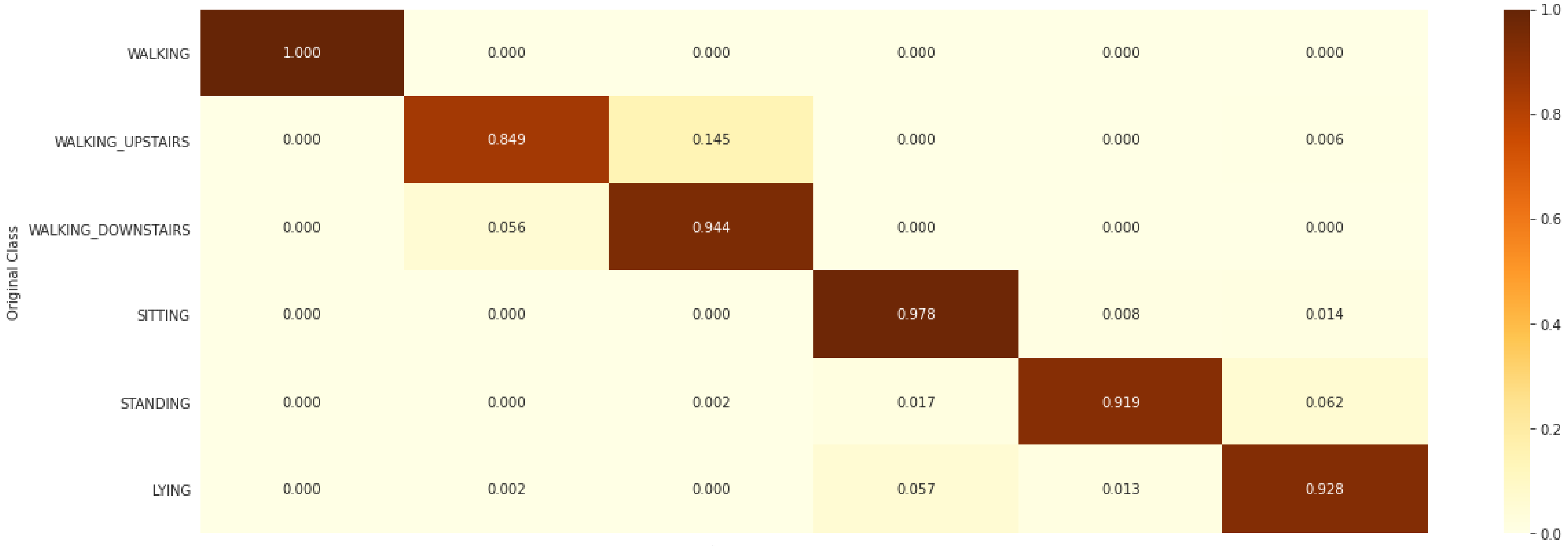
```
Score of best estimator:  0.9227464309993202
/n/n
Training model:  GridSearchCV(estimator=GradientBoostingClassifier(),
             param_grid={'max_depth': [1, 3], 'n_estimators': [50, 100]},
             verbose=10)
**************************************************
Accuracy:
**************************************************
Train Accuracy: 1.0 Test Accuracy 0.9379029521547336



**************************************************
Confusion Matrix :test
**************************************************
-------------------- Confusion matrix --------------------
```



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



2.7 Accuracy

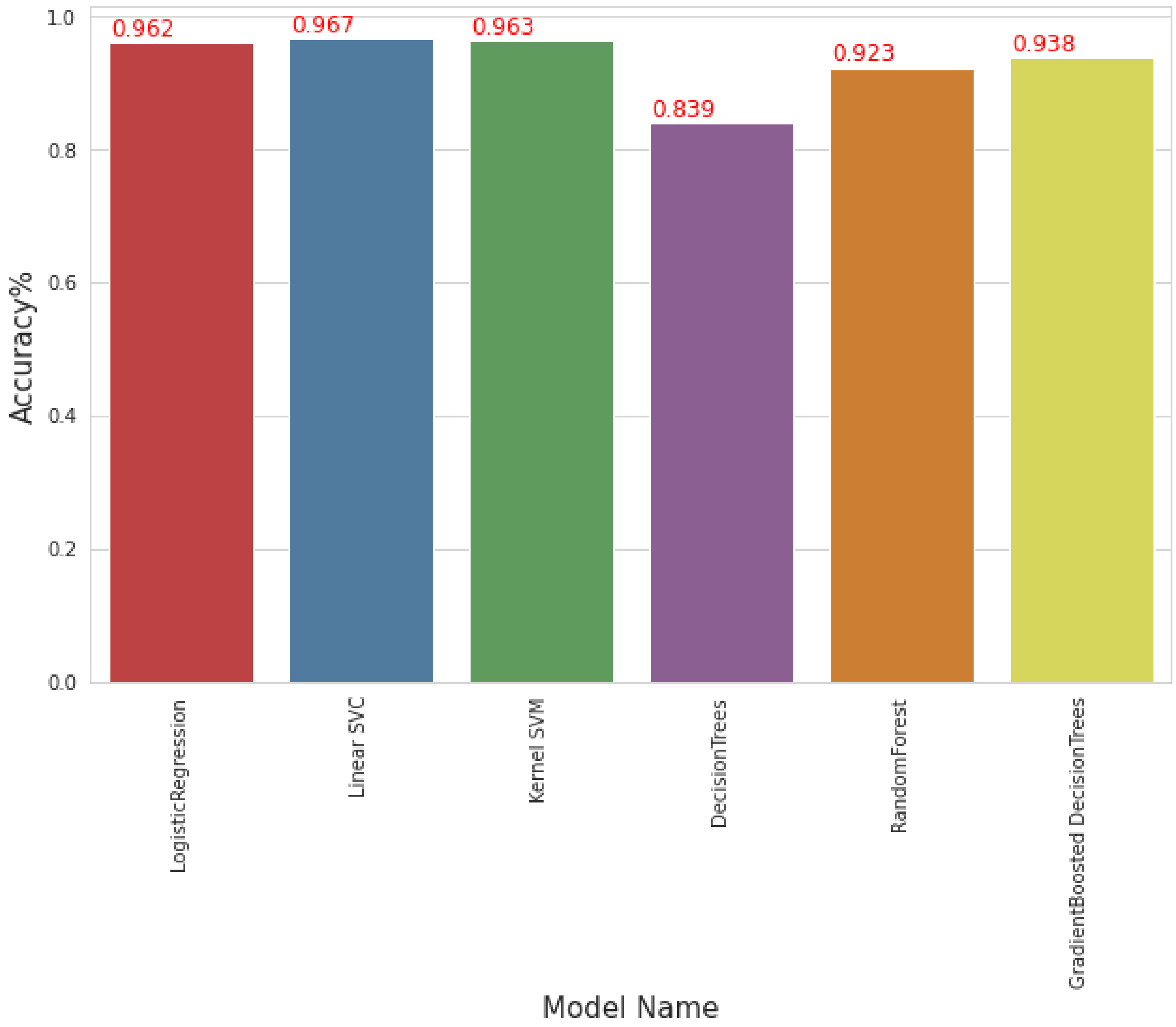
In [84]:

```
df['Accuracy%'] = round(df['Accuracy%'],3)
```

In [103]:

```
fig = plt.figure(figsize=(8,5))
ax = fig.add_axes([0,0,1,1])
sns.barplot(x='Model_name',y='Accuracy%',data=df,capsize=0.1)
for i in ax.patches:
    ax.text(x = i.get_x() + 0.01, y = i.get_height()+0.01, s = str(i.get_height()), fontsize = 12, color = "red")
plt.xticks(rotation=90)
plt.xlabel('Model Name',fontsize=15)
plt.ylabel('Accuracy%',fontsize=15)
```

Out[103]: `Text(0, 0.5, 'Accuracy%')`



- Observations :
  - We can observe that tree based models do not perform well in comparison to linear models

- Linear SVC gives the highest performance

3. Applying Deep Learning Models -LSTM(RNN)

In [1]:

```
# Activities are the class labels
# It is a 6 class classification
ACTIVITIES = {
    0: 'WALKING',
    1: 'WALKING_UPSTAIRS',
    2: 'WALKING_DOWNSTAIRS',
    3: 'SITTING',
    4: 'STANDING',
    5: 'LAYING',
}

# Utility function to print the confusion matrix
def confusion_matrix_(Y_true, Y_pred):
    Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_true, axis=1)])
    Y_pred = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_pred, axis=1)])

    return pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])
```

Data

In [2]:

```
# Data directory
DATADIR = 'UCI_HAR_Dataset'
```

In [3]:

```
# Raw data signals
# Signals are from Accelerometer and Gyroscope
# The signals are in x,y,z directions
# Sensor signals are filtered to have only body acceleration
# excluding the acceleration due to gravity
# Triaxial acceleration from the accelerometer is total acceleration
SIGNALS = [
    "body_acc_x",
    "body_acc_y",
    "body_acc_z",
    "body_gyro_x",
    "body_gyro_y",
    "body_gyro_z",
    "total_acc_x",
    "total_acc_y",
    "total_acc_z"
]
```

In [4]:
```
# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

# Utility function to load the load
def load_signals(subset):
    signals_data = []

    for signal in SIGNALS:
        filename = f'UCI_HAR_Dataset/{subset}/Inertial Signals/{signal}_{subset}.txt'
        signals_data.append(
            _read_csv(filename).values
        )

    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
    return np.transpose(signals_data, (1, 2, 0))
```

In [5]:
```

def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = f'UCI_HAR_Dataset/{subset}/y_{subset}.txt'
    y = _read_csv(filename)[0]

    return pd.get_dummies(y).values
```

In [6]:
```
def load_data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = load_signals('train'), load_signals('test')
    y_train, y_test = load_y('train'), load_y('test')

    return X_train, X_test, y_train, y_test
```

In [10]:
```
# Importing tensorflow
np.random.seed(42)
import tensorflow as tf
tf.random.set_seed(42)
```

In [11]:
```
# Configuring a session
session_conf = tf.compat.v1.ConfigProto(
    intra_op_parallelism_threads=1,
    inter_op_parallelism_threads=1
)
```

In [12]:
```
# Import Keras
from keras import backend as K
sess = tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(), config=session_conf)
K.set_session(sess)
```

In [13]:
```
# Importing libraries
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout
```

In [14]:
```
# Initializing parameters
epochs = 30
batch_size = 16
n_hidden = 32
```

In [15]:
```
# Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

In [16]:
```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

In [17]:
```
timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = _count_classes(Y_train)

print(timesteps)
print(input_dim)
print(len(X_train))
```

```
128
9
7352
```

- Defining the Architecture of LSTM

In [18]:

```
# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(n_hidden, input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model.add(Dropout(0.5))
# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='sigmoid'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm (LSTM)                  (None, 32)                5376      
_________________________________________________________________
dropout (Dropout)            (None, 32)                0         
_________________________________________________________________
dense (Dense)                (None, 6)                 198       
=================================================================
Total params: 5,574
Trainable params: 5,574
Non-trainable params: 0
_________________________________________________________________
```

In [19]:

```
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

In [20]:

```
# Training the model
model.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=30)
```

```
Epoch 1/30
460/460 [==============================] - 29s 31ms/step - loss: 1.3606 - accuracy: 0.4296 - val_loss: 0.9645 - val_accuracy: 0.6244
Epoch 2/30
460/460 [==============================] - 14s 31ms/step - loss: 0.8550 - accuracy: 0.6386 - val_loss: 0.7700 - val_accuracy: 0.6916
Epoch 3/30
460/460 [==============================] - 17s 36ms/step - loss: 0.6994 - accuracy: 0.6924 - val_loss: 0.6732 - val_accuracy: 0.7414
Epoch 4/30
460/460 [==============================] - 19s 41ms/step - loss: 0.6143 - accuracy: 0.7539 - val_loss: 0.6026 - val_accuracy: 0.8107
Epoch 5/30
460/460 [==============================] - 17s 38ms/step - loss: 0.4222 - accuracy: 0.8637 - val_loss: 0.5768 - val_accuracy: 0.8188
Epoch 6/30
460/460 [==============================] - 16s 35ms/step - loss: 0.3275 - accuracy: 0.8998 - val_loss: 0.4691 - val_accuracy: 0.8507
Epoch 7/30
460/460 [==============================] - 16s 35ms/step - loss: 0.2680 - accuracy: 0.9156 - val_loss: 0.3787 - val_accuracy: 0.8829
Epoch 8/30
460/460 [==============================] - 16s 35ms/step - loss: 0.2303 - accuracy: 0.9271 - val_loss: 0.4688 - val_accuracy: 0.8687
Epoch 9/30
460/460 [==============================] - 16s 35ms/step - loss: 0.2068 - accuracy: 0.9321 - val_loss: 0.4229 - val_accuracy: 0.8673
Epoch 10/30
460/460 [==============================] - 16s 35ms/step - loss: 0.2160 - accuracy: 0.9325 - val_loss: 0.4198 - val_accuracy: 0.8812
Epoch 11/30
460/460 [==============================] - 16s 35ms/step - loss: 0.2000 - accuracy: 0.9324 - val_loss: 0.3876 - val_accuracy: 0.8775
Epoch 12/30
460/460 [==============================] - 16s 35ms/step - loss: 0.2050 - accuracy: 0.9356 - val_loss: 0.4830 - val_accuracy: 0.8721
Epoch 13/30
460/460 [==============================] - 16s 35ms/step - loss: 0.1797 - accuracy: 0.9408 - val_loss: 0.4337 - val_accuracy: 0.8802
Epoch 14/30
460/460 [==============================] - 16s 35ms/step - loss: 0.1644 - accuracy: 0.9421 - val_loss: 0.3087 - val_accuracy: 0.8985
Epoch 15/30
460/460 [==============================] - 16s 36ms/step - loss: 0.1774 - accuracy: 0.9367 - val_loss: 0.3682 - val_accuracy: 0.9002
Epoch 16/30
460/460 [==============================] - 16s 36ms/step - loss: 0.1711 - accuracy: 0.9359 - val_loss: 0.3338 - val_accuracy: 0.8975
Epoch 17/30
460/460 [==============================] - 16s 36ms/step - loss: 0.1681 - accuracy: 0.9366 - val_loss: 0.3830 - val_accuracy: 0.8860
Epoch 18/30
460/460 [==============================] - 16s 35ms/step - loss: 0.1590 - accuracy: 0.9470 - val_loss: 0.3424 - val_accuracy: 0.9053
Epoch 19/30
460/460 [==============================] - 16s 35ms/step - loss: 0.1457 - accuracy: 0.9495 - val_loss: 0.3554 - val_accuracy: 0.8938
Epoch 20/30
460/460 [==============================] - 16s 36ms/step - loss: 0.1617 - accuracy: 0.9373 - val_loss: 0.2958 - val_accuracy: 0.8999
Epoch 21/30
460/460 [==============================] - 16s 35ms/step - loss: 0.1640 - accuracy: 0.9408 - val_loss: 0.2472 - val_accuracy: 0.9050
Epoch 22/30
460/460 [==============================] - 20s 44ms/step - loss: 0.1485 - accuracy: 0.9492 - val_loss: 0.4368 - val_accuracy: 0.8921
Epoch 23/30
460/460 [==============================] - 21s 45ms/step - loss: 0.1452 - accuracy: 0.9492 - val_loss: 0.5006 - val_accuracy: 0.8843
Epoch 24/30
460/460 [==============================] - 16s 35ms/step - loss: 0.1555 - accuracy: 0.9470 - val_loss: 0.4922 - val_accuracy: 0.8751
Epoch 25/30
460/460 [==============================] - 16s 36ms/step - loss: 0.1236 - accuracy: 0.9557 - val_loss: 0.4233 - val_accuracy: 0.8884
Epoch 26/30
```

```
460/460 [==============================] - 16s 35ms/step - loss: 0.1415 - accuracy: 0.9495 - val_loss: 0.3455 - val_accuracy: 0.9077
Epoch 27/30
460/460 [==============================] - 19s 41ms/step - loss: 0.1397 - accuracy: 0.9466 - val_loss: 0.5045 - val_accuracy: 0.8839
Epoch 28/30
460/460 [==============================] - 16s 36ms/step - loss: 0.1261 - accuracy: 0.9491 - val_loss: 0.5134 - val_accuracy: 0.8979
Epoch 29/30
460/460 [==============================] - 17s 37ms/step - loss: 0.1488 - accuracy: 0.9522 - val_loss: 0.4185 - val_accuracy: 0.9077
Epoch 30/30
460/460 [==============================] - 20s 44ms/step - loss: 0.1356 - accuracy: 0.9545 - val_loss: 0.4121 - val_accuracy: 0.8945
```

Out[20]: <keras.callbacks.History at 0x1c32c550358>

In [21]:

```
# Confusion Matrix
print(confusion_matrix_(Y_test, model.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0        27        0                   0
SITTING                  0      388       101        2                   0
STANDING                 0      102       427        1                   0
WALKING                  0        5         1      449                  18
WALKING_DOWNSTAIRS       0        0         0        0                 415
WALKING_UPSTAIRS         0        1         2        1                  20

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            0
STANDING                           2
WALKING                           23
WALKING_DOWNSTAIRS                 5
WALKING_UPSTAIRS                 447
```

In [22]:

```
score = model.evaluate(X_test, Y_test)
```

```
93/93 [==============================] - 1s 8ms/step - loss: 0.4121 - accuracy: 0.8945
```

In [23]:

```
score
```

Out[23]: [0.4121060073375702, 0.8944689631462097]

- With a simple 2 layer architecture we got 90.09% accuracy and a loss of 0.30
- We can further imporve the performace with Hyperparameter tuning

Assigment :

- try out with different lstm units
- try out with different drop out units
- add additional lstm layer and try increasing drop out units

Walkthrough :

- Experiment 1 :We will hyperparameter tune for multiple unit counts for lstm layer 1 .
- Experiment 2 :Choose the best parameters for layer1 lstm and layer 1 dropout,by keeping this fixed tune for layer2 lstm and and layer2 dropout

- Experiment 3 : Train with more number of epochs for the best selected architecture from exp2
- Experiment 4 : Train with more number of epochs for the best selected architecture from exp1

In [18]:

```
def experiment1_architecture(layer1_units,dropout):
    # Initiliazing the sequential model
    model = Sequential()
    # Configuring the parameters
    model.add(LSTM(layer1_units, input_shape=(timesteps, input_dim)))
    # Adding a dropout layer
    model.add(Dropout(dropout))
    # Adding a dense output layer with sigmoid activation
    model.add(Dense(n_classes, activation='sigmoid'))
    model.summary()
    return model

def experiment2_architecture(layer1_units,dropout1,layer2_units,dropout2):
    # Initiliazing the sequential model
    model = Sequential()
    # Configuring the parameters
    model.add(LSTM(layer1_units, return_sequences = True,input_shape=(timesteps, input_dim)))
    # Adding a dropout layer
    model.add(Dropout(dropout1))
    model.add(LSTM(layer2_units))
    model.add(Dropout(dropout2))
    # Adding a dense output layer with sigmoid activation
    model.add(Dense(n_classes, activation='sigmoid'))
    model.summary()
    return model
```

In [19]:

```
def EXP1(layer1_units,dropouts,return_y=False):
    exp1_scores = []
    for u1 in layer1_units:
        for dropout in dropouts:
            print('-'*100)
            print('Training Model with layer1 units:',u1,' Dropout:',dropout)
            print('-'*100)
            print('\n')
            model = experiment1_architecture(u1,dropout)
            ### compiling the model
            model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
            # Training the model
            history = model.fit(X_train,
                      Y_train,
                      batch_size=batch_size,
                      validation_data=(X_test, Y_test),
                      epochs=epochs)
            val_accuracy = np.amax(history.history['val_accuracy'])
            accuracy = np.amax(history.history['accuracy'])
            print('-'*100)
            print('Val Accuracy:',val_accuracy,' Train Accuracy:',accuracy)
            print('-'*100)
            print('\n')
            print('-'*100)
#             print('Confusion Matrix')
#             print('-'*100)
#             print('\n')
            # Confusion Matrix
            print(confusion_matrix_(Y_test, model.predict(X_test)))
            testscore = model.evaluate(X_test, Y_test)

            exp1_scores.append((u1,dropout,accuracy,val_accuracy,testscore))
    if return_y:
        return exp1_scores,Y_test,model.predict(X_test)
    else:
        return exp1_scores

epochs=8
exp1_scores = EXP1([32,64],[0.25,0.5,0.75])
```

In [31]:

```
exp1_df = pd.DataFrame(columns = ['Layer1Units','Dropout','TrainAccuracy','ValAccuracy','Score'] ,
                       data = exp1_scores)
```

In [32]:
```
exp1_df
```

Out[32]:

	Layer1Units	Dropout	TrainAccuracy	ValAccuracy	Score
0	32	0.25	0.937568	0.900577	[0.3132242262363434, 0.9005768299102783]
1	32	0.50	0.923830	0.883950	[0.46504589915275574, 0.8724126219749451]
2	32	0.75	0.844124	0.834069	[0.6033707857131958, 0.8340685367584229]
3	64	0.25	0.946953	0.901934	[0.3227238655090332, 0.8998982310295105]
4	64	0.50	0.940288	0.894808	[0.40707921981811523, 0.894808292388916]
5	64	0.75	0.923422	0.896166	[0.3765288293361664, 0.8961656093597412]

In [36]:
```
print('As we can see the best hyper parameter from the 6 combinations is :\n ',exp1_df.sort_values(by='Score').iloc[0])
print('\n')
print('score')
```

```
As we can see the best hyper parameter from the 6 combinations is :
  Layer1Units                                          32
Dropout                                            0.25
TrainAccuracy                                  0.937568
ValAccuracy                                    0.900577
Score            [0.3132242262363434, 0.9005768299102783]
Name: 0, dtype: object
```

Experiment 2 : LSTM layer2

- lets take our best layer1unit and best drop out rate from exp1

In [43]:

```python
def EXP2(layer2_units,dropouts):
    exp2_scores = []
    for u1 in layer2_units:
        for dropout in dropouts:
            print('-'*100)
            print('Training Model with layer1 units:',u1,' Dropout:',dropout)
            print('-'*100)
            print('\n')
            model = experiment2_architecture(32,0.25,u1,dropout)
            ### compiling the model
            model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
            # Training the model
            history = model.fit(X_train,
                      Y_train,
                      batch_size=batch_size,
                      validation_data=(X_test, Y_test),
                      epochs=epochs)
            val_accuracy = np.amax(history.history['val_accuracy'])
            accuracy = np.amax(history.history['accuracy'])
            print('-'*100)
            print('Val Accuracy:',val_accuracy,' Train Accuracy:',accuracy)
            print('-'*100)
            print('\n')
            print('-'*100)
#             print('Confusion Matrix')
#             print('-'*100)
#             print('\n')
            # Confusion Matrix
            print(confusion_matrix_(Y_test, model.predict(X_test)))
            testscore = model.evaluate(X_test, Y_test)

            exp2_scores.append((32,0.25,u1,dropout,accuracy,val_accuracy,testscore))
    return exp2_scores

epochs=8
exp2_scores = EXP2([16,32,64],[0.25,0.35,0.5])
```

```
----------------------------------------------------------------------------------------------------
Training Model with layer1 units: 16  Dropout: 0.25
----------------------------------------------------------------------------------------------------


Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_16 (LSTM)               (None, 128, 32)           5376
_________________________________________________________________
dropout_15 (Dropout)         (None, 128, 32)           0
_________________________________________________________________
lstm_17 (LSTM)               (None, 16)                3136
_________________________________________________________________
dropout_16 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_12 (Dense)             (None, 6)                 102
```

```
=================================================================
Total params: 8,614
Trainable params: 8,614
Non-trainable params: 0
_________________________________________________________________
Epoch 1/8
460/460 [==============================] - 36s 72ms/step - loss: 1.2269 - accuracy: 0.5000 - val_loss: 0.7957 - val_accuracy: 0.6831
Epoch 2/8
460/460 [==============================] - 40s 86ms/step - loss: 0.7050 - accuracy: 0.7060 - val_loss: 0.6706 - val_accuracy: 0.7652
Epoch 3/8
460/460 [==============================] - 34s 73ms/step - loss: 0.5209 - accuracy: 0.8263 - val_loss: 0.7121 - val_accuracy: 0.7516
Epoch 4/8
460/460 [==============================] - 33s 72ms/step - loss: 0.3533 - accuracy: 0.8886 - val_loss: 0.4184 - val_accuracy: 0.8829
Epoch 5/8
460/460 [==============================] - 33s 73ms/step - loss: 0.2625 - accuracy: 0.9232 - val_loss: 0.5136 - val_accuracy: 0.8537
Epoch 6/8
460/460 [==============================] - 33s 72ms/step - loss: 0.2238 - accuracy: 0.9358 - val_loss: 0.4782 - val_accuracy: 0.8772
Epoch 7/8
460/460 [==============================] - 34s 73ms/step - loss: 0.2096 - accuracy: 0.9333 - val_loss: 0.5080 - val_accuracy: 0.8778
Epoch 8/8
460/460 [==============================] - 34s 74ms/step - loss: 0.1670 - accuracy: 0.9460 - val_loss: 1.0715 - val_accuracy: 0.8185
----------------------------------------------------------------------------------------------------
Val Accuracy: 0.8829317688941956  Train Accuracy: 0.9409684538841248
----------------------------------------------------------------------------------------------------


----------------------------------------------------------------------------------------------------
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0         0        0                   0
SITTING                  0      394        67        3                   0
STANDING                 0       85       329        8                   0
WALKING                  0        0         0      349                  32
WALKING_DOWNSTAIRS       0        0         0        0                 374
WALKING_UPSTAIRS         1        0         0        2                  12

Pred                WALKING_UPSTAIRS
True
LAYING                            27
SITTING                           27
STANDING                         110
WALKING                          115
WALKING_DOWNSTAIRS                46
WALKING_UPSTAIRS                 456
93/93 [==============================] - 2s 17ms/step - loss: 1.0715 - accuracy: 0.8185
----------------------------------------------------------------------------------------------------
Training Model with layer1 units: 16  Dropout: 0.35
----------------------------------------------------------------------------------------------------


Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_18 (LSTM)               (None, 128, 32)           5376
_________________________________________________________________
dropout_17 (Dropout)         (None, 128, 32)           0
_________________________________________________________________
lstm_19 (LSTM)               (None, 16)                3136
```

```
_________________________________________________________________
dropout_18 (Dropout)         (None, 16)                0         
_________________________________________________________________
dense_13 (Dense)             (None, 6)                 102       
=================================================================
Total params: 8,614
Trainable params: 8,614
Non-trainable params: 0
_________________________________________________________________
Epoch 1/8
460/460 [==============================] - 38s 75ms/step - loss: 1.3062 - accuracy: 0.4623 - val_loss: 0.8896 - val_accuracy: 0.6617
Epoch 2/8
460/460 [==============================] - 34s 73ms/step - loss: 0.6701 - accuracy: 0.7393 - val_loss: 0.5426 - val_accuracy: 0.8178
Epoch 3/8
460/460 [==============================] - 34s 73ms/step - loss: 0.4446 - accuracy: 0.8506 - val_loss: 0.4390 - val_accuracy: 0.8673
Epoch 4/8
460/460 [==============================] - 34s 73ms/step - loss: 0.3293 - accuracy: 0.8856 - val_loss: 0.3721 - val_accuracy: 0.8758
Epoch 5/8
460/460 [==============================] - 33s 72ms/step - loss: 0.2443 - accuracy: 0.9210 - val_loss: 0.4277 - val_accuracy: 0.8812
Epoch 6/8
460/460 [==============================] - 33s 72ms/step - loss: 0.2025 - accuracy: 0.9309 - val_loss: 0.3580 - val_accuracy: 0.8958
Epoch 7/8
460/460 [==============================] - 35s 77ms/step - loss: 0.1752 - accuracy: 0.9354 - val_loss: 0.4520 - val_accuracy: 0.8823
Epoch 8/8
460/460 [==============================] - 36s 79ms/step - loss: 0.1619 - accuracy: 0.9476 - val_loss: 0.3582 - val_accuracy: 0.8968
----------------------------------------------------------------------------------------------------
Val Accuracy: 0.8968442678451538  Train Accuracy: 0.9413765072822571
----------------------------------------------------------------------------------------------------



----------------------------------------------------------------------------------------------------
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True                                                                         
LAYING                 510        0         0        0                   0   
SITTING                  0      376       106        0                   0   
STANDING                 0       88       444        0                   0   
WALKING                  0        0         2      473                   0   
WALKING_DOWNSTAIRS       0        0         0        6                 406   
WALKING_UPSTAIRS         0        1         0       21                  15   

Pred                WALKING_UPSTAIRS  
True                                  
LAYING                            27  
SITTING                            9  
STANDING                           0  
WALKING                           21  
WALKING_DOWNSTAIRS                 8  
WALKING_UPSTAIRS                 434  
93/93 [==============================] - 2s 17ms/step - loss: 0.3582 - accuracy: 0.8968
----------------------------------------------------------------------------------------------------
Training Model with layer1 units: 16  Dropout: 0.5
----------------------------------------------------------------------------------------------------


Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_20 (LSTM)               (None, 128, 32)           5376      
```

```
_________________________________________________________________
dropout_19 (Dropout)         (None, 128, 32)           0         
_________________________________________________________________
lstm_21 (LSTM)               (None, 16)                3136      
_________________________________________________________________
dropout_20 (Dropout)         (None, 16)                0         
_________________________________________________________________
dense_14 (Dense)             (None, 6)                 102       
=================================================================
Total params: 8,614
Trainable params: 8,614
Non-trainable params: 0
_________________________________________________________________
Epoch 1/8

460/460 [==============================] - 38s 76ms/step - loss: 1.3923 - accuracy: 0.4201 - val_loss: 0.8757 - val_accuracy: 0.6447
Epoch 2/8
460/460 [==============================] - 38s 83ms/step - loss: 0.8346 - accuracy: 0.6605 - val_loss: 0.7184 - val_accuracy: 0.7055
Epoch 3/8
460/460 [==============================] - 34s 74ms/step - loss: 0.6716 - accuracy: 0.7234 - val_loss: 0.5599 - val_accuracy: 0.7896
Epoch 4/8
460/460 [==============================] - 37s 80ms/step - loss: 0.4771 - accuracy: 0.8314 - val_loss: 0.4962 - val_accuracy: 0.8541
Epoch 5/8
460/460 [==============================] - 37s 80ms/step - loss: 0.3613 - accuracy: 0.8934 - val_loss: 0.4473 - val_accuracy: 0.8554
Epoch 6/8
460/460 [==============================] - 34s 74ms/step - loss: 0.2986 - accuracy: 0.9134 - val_loss: 0.3756 - val_accuracy: 0.8873
Epoch 7/8
460/460 [==============================] - 34s 74ms/step - loss: 0.2681 - accuracy: 0.9180 - val_loss: 0.3890 - val_accuracy: 0.8941
Epoch 8/8
460/460 [==============================] - 34s 74ms/step - loss: 0.2328 - accuracy: 0.9290 - val_loss: 0.4543 - val_accuracy: 0.8772
----------------------------------------------------------------------------------------------------
Val Accuracy: 0.8941296339035034  Train Accuracy: 0.9270946383476257
----------------------------------------------------------------------------------------------------



----------------------------------------------------------------------------------------------------
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True                                                                         
LAYING                 510        0         0        0                   0   
SITTING                  2      378        97        3                   2   
STANDING                 0       92       431        7                   1   
WALKING                  0        0         0      443                  53   
WALKING_DOWNSTAIRS       0        0         0        1                 417   
WALKING_UPSTAIRS         0        0         0       33                  32   

Pred                WALKING_UPSTAIRS  
True                                  
LAYING                            27  
SITTING                            9  
STANDING                           1  
WALKING                            0  
WALKING_DOWNSTAIRS                 2  
WALKING_UPSTAIRS                 406  
93/93 [==============================] - 2s 20ms/step - loss: 0.4543 - accuracy: 0.8772
----------------------------------------------------------------------------------------------------
Training Model with layer1 units: 32  Dropout: 0.25
----------------------------------------------------------------------------------------------------
```

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_22 (LSTM)               (None, 128, 32)           5376      
_________________________________________________________________
dropout_21 (Dropout)         (None, 128, 32)           0         
_________________________________________________________________
lstm_23 (LSTM)               (None, 32)                8320      
_________________________________________________________________
dropout_22 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_15 (Dense)             (None, 6)                 198       
=================================================================
Total params: 13,894
Trainable params: 13,894
Non-trainable params: 0
_________________________________________________________________
Epoch 1/8
460/460 [==============================] - 40s 79ms/step - loss: 1.1920 - accuracy: 0.4767 - val_loss: 0.7582 - val_accuracy: 0.6827
Epoch 2/8
460/460 [==============================] - 36s 78ms/step - loss: 0.5782 - accuracy: 0.7684 - val_loss: 0.5281 - val_accuracy: 0.8171
Epoch 3/8
460/460 [==============================] - 35s 77ms/step - loss: 0.3523 - accuracy: 0.8905 - val_loss: 0.4265 - val_accuracy: 0.8711
Epoch 4/8
460/460 [==============================] - 35s 76ms/step - loss: 0.2617 - accuracy: 0.9108 - val_loss: 0.3510 - val_accuracy: 0.8887
Epoch 5/8
460/460 [==============================] - 38s 82ms/step - loss: 0.2014 - accuracy: 0.9348 - val_loss: 0.4273 - val_accuracy: 0.8636
Epoch 6/8
460/460 [==============================] - 35s 76ms/step - loss: 0.1715 - accuracy: 0.9413 - val_loss: 0.3672 - val_accuracy: 0.8955
Epoch 7/8
460/460 [==============================] - 37s 80ms/step - loss: 0.1596 - accuracy: 0.9364 - val_loss: 0.3290 - val_accuracy: 0.9023
Epoch 8/8
460/460 [==============================] - 37s 81ms/step - loss: 0.1400 - accuracy: 0.9420 - val_loss: 0.5866 - val_accuracy: 0.8490
----------------------------------------------------------------------------------------------------
Val Accuracy: 0.9022734761238098  Train Accuracy: 0.9408324360847473
----------------------------------------------------------------------------------------------------


----------------------------------------------------------------------------------------------------
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True                                                                         
LAYING                 509        1         0        0                   0   
SITTING                  2      410        76        0                   1   
STANDING                 0      102       429        1                   0   
WALKING                  0        8        46      358                   5   
WALKING_DOWNSTAIRS       0        0         0        1                 335   
WALKING_UPSTAIRS         0        0         1        1                   8   

Pred                WALKING_UPSTAIRS  
True                                  
LAYING                            27  
SITTING                            2  
STANDING                           0  
WALKING                           79  
WALKING_DOWNSTAIRS                84  
WALKING_UPSTAIRS                 461  
93/93 [==============================] - 2s 22ms/step - loss: 0.5866 - accuracy: 0.8490
----------------------------------------------------------------------------------------------------
```

```
Training Model with layer1 units: 32  Dropout: 0.35
----------------------------------------------------------------------------------------------------


Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_24 (LSTM)               (None, 128, 32)           5376      
_________________________________________________________________
dropout_23 (Dropout)         (None, 128, 32)           0         
_________________________________________________________________
lstm_25 (LSTM)               (None, 32)                8320      
_________________________________________________________________
dropout_24 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_16 (Dense)             (None, 6)                 198       
=================================================================
Total params: 13,894
Trainable params: 13,894
Non-trainable params: 0
_________________________________________________________________
Epoch 1/8
460/460 [==============================] - 41s 81ms/step - loss: 1.2596 - accuracy: 0.4573 - val_loss: 0.7549 - val_accuracy: 0.6973
Epoch 2/8
460/460 [==============================] - 39s 84ms/step - loss: 0.6380 - accuracy: 0.7351 - val_loss: 0.6138 - val_accuracy: 0.7648
Epoch 3/8
460/460 [==============================] - 37s 80ms/step - loss: 0.4644 - accuracy: 0.8131 - val_loss: 0.5244 - val_accuracy: 0.8388
Epoch 4/8
460/460 [==============================] - 35s 76ms/step - loss: 0.3084 - accuracy: 0.8932 - val_loss: 0.3522 - val_accuracy: 0.8911

Epoch 5/8
460/460 [==============================] - 35s 76ms/step - loss: 0.1942 - accuracy: 0.9402 - val_loss: 0.3910 - val_accuracy: 0.8904
Epoch 6/8
460/460 [==============================] - 36s 79ms/step - loss: 0.1690 - accuracy: 0.9415 - val_loss: 0.7080 - val_accuracy: 0.8429
Epoch 7/8
460/460 [==============================] - 36s 79ms/step - loss: 0.1604 - accuracy: 0.9421 - val_loss: 0.4216 - val_accuracy: 0.9036
Epoch 8/8
460/460 [==============================] - 35s 75ms/step - loss: 0.1439 - accuracy: 0.9472 - val_loss: 0.5197 - val_accuracy: 0.8935
----------------------------------------------------------------------------------------------------
Val Accuracy: 0.903630793094635  Train Accuracy: 0.9451850056648254
----------------------------------------------------------------------------------------------------



----------------------------------------------------------------------------------------------------
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True                                                                         
LAYING                 510        0         0        0                   0   
SITTING                  0      376        89       15                   3   
STANDING                 0       84       447        1                   0   
WALKING                  0        0         1      455                  25   
WALKING_DOWNSTAIRS       0        0         0        0                 417   
WALKING_UPSTAIRS         0        0         0       20                  23   

Pred                WALKING_UPSTAIRS  
True                                  
LAYING                            27  
SITTING                            8  
```

```
STANDING                           0
WALKING                           15
WALKING_DOWNSTAIRS                 3
WALKING_UPSTAIRS                 428
93/93 [==============================] - 2s 21ms/step - loss: 0.5197 - accuracy: 0.8935
----------------------------------------------------------------------------------------------------
Training Model with layer1 units: 32  Dropout: 0.5
----------------------------------------------------------------------------------------------------


Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_26 (LSTM)               (None, 128, 32)           5376      
_________________________________________________________________
dropout_25 (Dropout)         (None, 128, 32)           0         
_________________________________________________________________
lstm_27 (LSTM)               (None, 32)                8320      
_________________________________________________________________
dropout_26 (Dropout)         (None, 32)                0         
_________________________________________________________________
dense_17 (Dense)             (None, 6)                 198       
=================================================================
Total params: 13,894
Trainable params: 13,894
Non-trainable params: 0
_________________________________________________________________
Epoch 1/8
460/460 [==============================] - 39s 77ms/step - loss: 1.2270 - accuracy: 0.4910 - val_loss: 0.7902 - val_accuracy: 0.6776
Epoch 2/8
460/460 [==============================] - 34s 75ms/step - loss: 0.6480 - accuracy: 0.7598 - val_loss: 0.5002 - val_accuracy: 0.8263
Epoch 3/8
460/460 [==============================] - 35s 76ms/step - loss: 0.4044 - accuracy: 0.8826 - val_loss: 0.4649 - val_accuracy: 0.8510
Epoch 4/8
460/460 [==============================] - 36s 78ms/step - loss: 0.3020 - accuracy: 0.9029 - val_loss: 0.4583 - val_accuracy: 0.8687
Epoch 5/8
460/460 [==============================] - 36s 77ms/step - loss: 0.2087 - accuracy: 0.9332 - val_loss: 0.4209 - val_accuracy: 0.8775
Epoch 6/8
460/460 [==============================] - 34s 73ms/step - loss: 0.1850 - accuracy: 0.9405 - val_loss: 0.5935 - val_accuracy: 0.8541
Epoch 7/8
460/460 [==============================] - 36s 79ms/step - loss: 0.1880 - accuracy: 0.9342 - val_loss: 0.2414 - val_accuracy: 0.9145
Epoch 8/8
460/460 [==============================] - 35s 76ms/step - loss: 0.1725 - accuracy: 0.9403 - val_loss: 0.4030 - val_accuracy: 0.8860
----------------------------------------------------------------------------------------------------
Val Accuracy: 0.9144893288612366  Train Accuracy: 0.9394722580909729
----------------------------------------------------------------------------------------------------


----------------------------------------------------------------------------------------------------
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True                                                                         
LAYING                 505        4         0        0                   0   
SITTING                  1      391        87        2                   0   
STANDING                 0       92       432        5                   0   
WALKING                  0        0         0      486                   9   
WALKING_DOWNSTAIRS       0        0         0       27                 355   
WALKING_UPSTAIRS         0        0         0       27                   2   
```

```
Pred                WALKING_UPSTAIRS
True
LAYING                            28
SITTING                           10
STANDING                           3
WALKING                            1
WALKING_DOWNSTAIRS                38
WALKING_UPSTAIRS                 442
93/93 [==============================] - 2s 21ms/step - loss: 0.4030 - accuracy: 0.8860
----------------------------------------------------------------------------------------------------
Training Model with layer1 units: 64  Dropout: 0.25
----------------------------------------------------------------------------------------------------


Model: "sequential_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_28 (LSTM)               (None, 128, 32)           5376
_________________________________________________________________
dropout_27 (Dropout)         (None, 128, 32)           0
_________________________________________________________________
lstm_29 (LSTM)               (None, 64)                24832
_________________________________________________________________
dropout_28 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_18 (Dense)             (None, 6)                 390
=================================================================
Total params: 30,598
Trainable params: 30,598
Non-trainable params: 0
_________________________________________________________________
Epoch 1/8
460/460 [==============================] - 44s 89ms/step - loss: 1.0947 - accuracy: 0.5241 - val_loss: 0.6810 - val_accuracy: 0.7374
Epoch 2/8
460/460 [==============================] - 41s 89ms/step - loss: 0.4993 - accuracy: 0.8086 - val_loss: 0.6253 - val_accuracy: 0.8303
Epoch 3/8
460/460 [==============================] - 41s 89ms/step - loss: 0.3046 - accuracy: 0.8970 - val_loss: 0.5253 - val_accuracy: 0.8208
Epoch 4/8
460/460 [==============================] - 41s 89ms/step - loss: 0.2198 - accuracy: 0.9144 - val_loss: 0.4560 - val_accuracy: 0.8785
Epoch 5/8
460/460 [==============================] - 41s 89ms/step - loss: 0.1724 - accuracy: 0.9396 - val_loss: 0.5283 - val_accuracy: 0.8605
Epoch 6/8
460/460 [==============================] - 41s 90ms/step - loss: 0.1522 - accuracy: 0.9419 - val_loss: 0.5795 - val_accuracy: 0.8588
Epoch 7/8
460/460 [==============================] - 41s 90ms/step - loss: 0.1473 - accuracy: 0.9439 - val_loss: 0.4878 - val_accuracy: 0.8870
Epoch 8/8
460/460 [==============================] - 45s 98ms/step - loss: 0.1439 - accuracy: 0.9456 - val_loss: 0.4472 - val_accuracy: 0.8799

----------------------------------------------------------------------------------------------------
Val Accuracy: 0.8870037198066711  Train Accuracy: 0.9466812014579773
----------------------------------------------------------------------------------------------------


----------------------------------------------------------------------------------------------------
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0         0        0                   0
SITTING                  4      378       104        0                   1
```

```
STANDING                 0       85       444       0                   0
WALKING                  0        0        36     384                  15
WALKING_DOWNSTAIRS       0        0         0       6                 413
WALKING_UPSTAIRS         0        1         0       1                   5

Pred                WALKING_UPSTAIRS
True
LAYING                            27
SITTING                            4
STANDING                           3
WALKING                           61
WALKING_DOWNSTAIRS                 1
WALKING_UPSTAIRS                 464
93/93 [==============================] - 3s 28ms/step - loss: 0.4472 - accuracy: 0.8799
----------------------------------------------------------------------------------------------------
Training Model with layer1 units: 64  Dropout: 0.35
----------------------------------------------------------------------------------------------------


Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_30 (LSTM)               (None, 128, 32)           5376
_________________________________________________________________
dropout_29 (Dropout)         (None, 128, 32)           0
_________________________________________________________________
lstm_31 (LSTM)               (None, 64)                24832
_________________________________________________________________
dropout_30 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_19 (Dense)             (None, 6)                 390
=================================================================
Total params: 30,598
Trainable params: 30,598
Non-trainable params: 0
_________________________________________________________________
Epoch 1/8
460/460 [==============================] - 47s 95ms/step - loss: 1.1676 - accuracy: 0.5027 - val_loss: 0.7541 - val_accuracy: 0.7394
Epoch 2/8
460/460 [==============================] - 43s 93ms/step - loss: 0.5183 - accuracy: 0.8259 - val_loss: 0.5972 - val_accuracy: 0.8351
Epoch 3/8
460/460 [==============================] - 43s 93ms/step - loss: 0.3188 - accuracy: 0.8957 - val_loss: 0.4360 - val_accuracy: 0.8755
Epoch 4/8
460/460 [==============================] - 43s 93ms/step - loss: 0.2325 - accuracy: 0.9193 - val_loss: 0.8257 - val_accuracy: 0.8059
Epoch 5/8
460/460 [==============================] - 44s 95ms/step - loss: 0.1895 - accuracy: 0.9356 - val_loss: 0.4456 - val_accuracy: 0.8829
Epoch 6/8
460/460 [==============================] - 43s 94ms/step - loss: 0.1594 - accuracy: 0.9409 - val_loss: 0.4093 - val_accuracy: 0.8941
Epoch 7/8
460/460 [==============================] - 46s 99ms/step - loss: 0.1483 - accuracy: 0.9434 - val_loss: 0.3693 - val_accuracy: 0.8935
Epoch 8/8
460/460 [==============================] - 44s 95ms/step - loss: 0.1560 - accuracy: 0.9453 - val_loss: 0.3513 - val_accuracy: 0.8975
----------------------------------------------------------------------------------------------------
Val Accuracy: 0.8975229263305664  Train Accuracy: 0.9435527920722961
----------------------------------------------------------------------------------------------------


----------------------------------------------------------------------------------------------------
```

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0        22        0                   0
SITTING                  0      375       112        1                   0
STANDING                 0       95       434        3                   0
WALKING                  0        1         2      481                  11
WALKING_DOWNSTAIRS       0        0         0       19                 397
WALKING_UPSTAIRS         0        0         0        6                  17

Pred                WALKING_UPSTAIRS
True
LAYING                             5
SITTING                            3
STANDING                           0
WALKING                            1
WALKING_DOWNSTAIRS                 4
WALKING_UPSTAIRS                 448
93/93 [==============================] - 3s 31ms/step - loss: 0.3513 - accuracy: 0.8975
----------------------------------------------------------------------------------------------------
Training Model with layer1 units: 64  Dropout: 0.5
----------------------------------------------------------------------------------------------------


Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_32 (LSTM)               (None, 128, 32)           5376
_________________________________________________________________
dropout_31 (Dropout)         (None, 128, 32)           0
_________________________________________________________________
lstm_33 (LSTM)               (None, 64)                24832
_________________________________________________________________
dropout_32 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_20 (Dense)             (None, 6)                 390
=================================================================
Total params: 30,598
Trainable params: 30,598
Non-trainable params: 0
_________________________________________________________________
Epoch 1/8
460/460 [==============================] - 47s 96ms/step - loss: 1.1609 - accuracy: 0.4995 - val_loss: 0.8925 - val_accuracy: 0.6569
Epoch 2/8
460/460 [==============================] - 42s 92ms/step - loss: 0.5980 - accuracy: 0.7491 - val_loss: 0.6153 - val_accuracy: 0.8015
Epoch 3/8
460/460 [==============================] - 42s 92ms/step - loss: 0.3837 - accuracy: 0.8801 - val_loss: 0.5265 - val_accuracy: 0.8463
Epoch 4/8
460/460 [==============================] - 43s 93ms/step - loss: 0.2522 - accuracy: 0.9098 - val_loss: 2.0341 - val_accuracy: 0.6502
Epoch 5/8
460/460 [==============================] - 43s 93ms/step - loss: 0.2282 - accuracy: 0.9302 - val_loss: 0.3757 - val_accuracy: 0.8711
Epoch 6/8
460/460 [==============================] - 43s 93ms/step - loss: 0.1730 - accuracy: 0.9384 - val_loss: 0.3433 - val_accuracy: 0.8870
Epoch 7/8
460/460 [==============================] - 43s 93ms/step - loss: 0.1623 - accuracy: 0.9370 - val_loss: 0.3714 - val_accuracy: 0.8880
Epoch 8/8
460/460 [==============================] - 43s 93ms/step - loss: 0.1455 - accuracy: 0.9451 - val_loss: 0.4741 - val_accuracy: 0.8887
----------------------------------------------------------------------------------------------------
Val Accuracy: 0.8887003660202026  Train Accuracy: 0.9420565962791443
```

```
-----------------------------------------------------------------------------------------------


-----------------------------------------------------------------------------------------------
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 512        0         0        0                   0
SITTING                  5      371       111        0                   0
STANDING                 0       68       463        1                   0
WALKING                  0        0        30      426                  15
WALKING_DOWNSTAIRS       0        0         0        2                 408
WALKING_UPSTAIRS         0        0         2       10                  20

Pred                WALKING_UPSTAIRS
True
LAYING                            25
SITTING                            4
STANDING                           0
WALKING                           25
WALKING_DOWNSTAIRS                10
WALKING_UPSTAIRS                 439

93/93 [==============================] - 3s 27ms/step - loss: 0.4741 - accuracy: 0.8887
```

In [44]:

```
exp2_df = pd.DataFrame(columns = ['Layer1Units','Dropout1','Layer2Units','Dropout2','TrainAccuracy','ValAccuracy','Score'] ,
                       data = exp2_scores)
```

In [47]:

```
exp2_df.sort_values(by='Score')
```

Out[47]:

	Layer1Units	Dropout1	Layer2Units	Dropout2	TrainAccuracy	ValAccuracy	Score
7	32	0.25	64	0.35	0.943553	0.897523	[0.3512811064720154, 0.8975229263305664]
1	32	0.25	16	0.35	0.941377	0.896844	[0.35821279883384705, 0.8968442678451538]
5	32	0.25	32	0.50	0.939472	0.914489	[0.4029683470726013, 0.8859857320785522]
6	32	0.25	64	0.25	0.946681	0.887004	[0.4472033381462097, 0.8798778653144836]
2	32	0.25	16	0.50	0.927095	0.894130	[0.4542565941810608, 0.8771632313728333]
8	32	0.25	64	0.50	0.942057	0.888700	[0.474104106426239, 0.8887003660202026]
4	32	0.25	32	0.35	0.945185	0.903631	[0.5197166204452515, 0.8934509754180908]
3	32	0.25	32	0.25	0.940832	0.902273	[0.5865686535835266, 0.8489989638328552]
0	32	0.25	16	0.25	0.940968	0.882932	[1.0714815855026245, 0.8184594511985779]

In [48]:

```
print('We can see that best model with one layer of LSTM gives 90% accuracy and with two layers it gives 89.75% of accuracy')
print()
```

```
We can see that best model with one layer of LSTM gives 90% accuracy and with two layers it gives 89.75% of accuracy
```

Experiment 3: train the best found model for more epochs with 2 lstm layers

In [51]:

```
epochs=30
exp3_scores = EXP2([64],[0.35])
```

```
----------------------------------------------------------------------------------------------------
Training Model with layer1 units: 64  Dropout: 0.35
----------------------------------------------------------------------------------------------------


Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_36 (LSTM)               (None, 128, 32)           5376      
_________________________________________________________________
dropout_35 (Dropout)         (None, 128, 32)           0         
_________________________________________________________________
lstm_37 (LSTM)               (None, 64)                24832     
_________________________________________________________________
dropout_36 (Dropout)         (None, 64)                0         
_________________________________________________________________
dense_22 (Dense)             (None, 6)                 390       
=================================================================
Total params: 30,598
Trainable params: 30,598
Non-trainable params: 0
_________________________________________________________________
Epoch 1/30
460/460 [==============================] - 38s 77ms/step - loss: 1.1211 - accuracy: 0.5152 - val_loss: 0.6898 - val_accuracy: 0.7302
Epoch 2/30
460/460 [==============================] - 40s 86ms/step - loss: 0.5727 - accuracy: 0.7649 - val_loss: 0.5581 - val_accuracy: 0.8032
Epoch 3/30
460/460 [==============================] - 39s 85ms/step - loss: 0.3229 - accuracy: 0.8968 - val_loss: 0.4362 - val_accuracy: 0.8599
Epoch 4/30
460/460 [==============================] - 38s 83ms/step - loss: 0.2232 - accuracy: 0.9203 - val_loss: 0.9883 - val_accuracy: 0.7699
Epoch 5/30
460/460 [==============================] - 41s 89ms/step - loss: 0.1892 - accuracy: 0.9381 - val_loss: 0.4958 - val_accuracy: 0.8744
Epoch 6/30
460/460 [==============================] - 42s 91ms/step - loss: 0.1554 - accuracy: 0.9424 - val_loss: 0.4864 - val_accuracy: 0.8599
Epoch 7/30
460/460 [==============================] - 39s 85ms/step - loss: 0.1597 - accuracy: 0.9387 - val_loss: 0.4525 - val_accuracy: 0.8670
Epoch 8/30
460/460 [==============================] - 39s 85ms/step - loss: 0.1425 - accuracy: 0.9482 - val_loss: 0.4307 - val_accuracy: 0.8965
Epoch 9/30
460/460 [==============================] - 42s 92ms/step - loss: 0.1304 - accuracy: 0.9465 - val_loss: 0.5749 - val_accuracy: 0.8890
Epoch 10/30
460/460 [==============================] - 38s 84ms/step - loss: 0.1420 - accuracy: 0.9500 - val_loss: 0.4607 - val_accuracy: 0.9019
Epoch 11/30
460/460 [==============================] - 38s 82ms/step - loss: 0.1324 - accuracy: 0.9458 - val_loss: 0.4597 - val_accuracy: 0.9030
Epoch 12/30
460/460 [==============================] - 40s 87ms/step - loss: 0.1199 - accuracy: 0.9537 - val_loss: 0.4394 - val_accuracy: 0.9121
Epoch 13/30
460/460 [==============================] - 47s 102ms/step - loss: 0.1175 - accuracy: 0.9568 - val_loss: 0.5623 - val_accuracy: 0.8948
Epoch 14/30
460/460 [==============================] - 44s 95ms/step - loss: 0.1227 - accuracy: 0.9545 - val_loss: 0.6960 - val_accuracy: 0.8979
Epoch 15/30
460/460 [==============================] - 45s 98ms/step - loss: 0.1186 - accuracy: 0.9531 - val_loss: 0.7664 - val_accuracy: 0.8714
Epoch 16/30
460/460 [==============================] - 45s 98ms/step - loss: 0.1154 - accuracy: 0.9504 - val_loss: 0.5787 - val_accuracy: 0.9074
```

```
Epoch 17/30
460/460 [==============================] - 43s 93ms/step - loss: 0.1455 - accuracy: 0.9449 - val_loss: 0.7724 - val_accuracy: 0.8789
Epoch 18/30
460/460 [==============================] - 44s 96ms/step - loss: 0.1435 - accuracy: 0.9518 - val_loss: 0.6881 - val_accuracy: 0.8951
Epoch 19/30
460/460 [==============================] - 40s 86ms/step - loss: 0.1211 - accuracy: 0.9530 - val_loss: 0.6853 - val_accuracy: 0.8948
Epoch 20/30
460/460 [==============================] - 40s 86ms/step - loss: 0.1370 - accuracy: 0.9481 - val_loss: 0.7157 - val_accuracy: 0.8789
Epoch 21/30
460/460 [==============================] - 39s 86ms/step - loss: 0.1221 - accuracy: 0.9508 - val_loss: 0.6466 - val_accuracy: 0.9091
Epoch 22/30
460/460 [==============================] - 40s 86ms/step - loss: 0.1162 - accuracy: 0.9517 - val_loss: 0.7480 - val_accuracy: 0.8999
Epoch 23/30
460/460 [==============================] - 40s 86ms/step - loss: 0.1474 - accuracy: 0.9486 - val_loss: 0.6676 - val_accuracy: 0.9077
Epoch 24/30
460/460 [==============================] - 47s 103ms/step - loss: 0.1136 - accuracy: 0.9536 - val_loss: 0.6813 - val_accuracy: 0.9033
Epoch 25/30
460/460 [==============================] - 42s 91ms/step - loss: 0.0976 - accuracy: 0.9586 - val_loss: 0.6944 - val_accuracy: 0.8965
Epoch 26/30
460/460 [==============================] - 40s 86ms/step - loss: 0.1073 - accuracy: 0.9565 - val_loss: 0.6699 - val_accuracy: 0.8951
Epoch 27/30
460/460 [==============================] - 41s 89ms/step - loss: 0.1198 - accuracy: 0.9500 - val_loss: 0.7216 - val_accuracy: 0.8945
Epoch 28/30
460/460 [==============================] - 40s 87ms/step - loss: 0.1101 - accuracy: 0.9577 - val_loss: 0.7816 - val_accuracy: 0.8887
Epoch 29/30
460/460 [==============================] - 43s 93ms/step - loss: 0.1005 - accuracy: 0.9551 - val_loss: 0.7573 - val_accuracy: 0.8962
Epoch 30/30
460/460 [==============================] - 46s 99ms/step - loss: 0.0918 - accuracy: 0.9612 - val_loss: 0.7453 - val_accuracy: 0.8843
----------------------------------------------------------------------------------------------------
Val Accuracy: 0.9121140241622925  Train Accuracy: 0.9574265480041504
----------------------------------------------------------------------------------------------------



----------------------------------------------------------------------------------------------------
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0        27        0                   0
SITTING                  0      422        65        1                   0
STANDING                 0      140       391        1                   0
WALKING                  0        0         0      437                  36
WALKING_DOWNSTAIRS       0        0         0        0                 420
WALKING_UPSTAIRS         0        0         9       15                  21

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            3
STANDING                           0
WALKING                           23
WALKING_DOWNSTAIRS                 0
WALKING_UPSTAIRS                 426
93/93 [==============================] - 3s 30ms/step - loss: 0.7453 - accuracy: 0.8843
```

In [53]:

```
exp3_df = pd.DataFrame(columns = ['Layer1Units','Dropout1','Layer2Units','Dropout2','TrainAccuracy','ValAccuracy','Score'] ,
                       data = exp3_scores)
```

In [54]:

```
exp3_df
```

Out[54]:

	Layer1Units	Dropout1	Layer2Units	Dropout2	TrainAccuracy	ValAccuracy	Score
0	32	0.25	64	0.35	0.957427	0.912114	[0.7452968955039978, 0.8842890858650208]

Experiment 4: train the best found model for more epochs with 2 lstm layers

In [20]:
```
epochs=50
exp4_scores,ytest,ypred = EXP1([32],[0.25],True)
```

```
--------------------------------------------------------------------------------------------------------
Training Model with layer1 units: 32  Dropout: 0.25
--------------------------------------------------------------------------------------------------------


Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm (LSTM)                  (None, 32)                5376      
_________________________________________________________________
dropout (Dropout)            (None, 32)                0         
_________________________________________________________________
dense (Dense)                (None, 6)                 198       
=================================================================
Total params: 5,574
Trainable params: 5,574
Non-trainable params: 0
_________________________________________________________________
Epoch 1/50
460/460 [==============================] - 31s 34ms/step - loss: 1.3076 - accuracy: 0.4562 - val_loss: 0.8487 - val_accuracy: 0.6552
Epoch 2/50
460/460 [==============================] - 17s 37ms/step - loss: 0.7356 - accuracy: 0.6754 - val_loss: 0.7122 - val_accuracy: 0.7268
Epoch 3/50
460/460 [==============================] - 17s 37ms/step - loss: 0.5915 - accuracy: 0.7603 - val_loss: 0.5989 - val_accuracy: 0.7889
Epoch 4/50
460/460 [==============================] - 17s 38ms/step - loss: 0.4342 - accuracy: 0.8469 - val_loss: 0.5092 - val_accuracy: 0.8222
Epoch 5/50
460/460 [==============================] - 17s 37ms/step - loss: 0.2714 - accuracy: 0.9111 - val_loss: 0.3608 - val_accuracy: 0.8911
Epoch 6/50
460/460 [==============================] - 16s 35ms/step - loss: 0.2221 - accuracy: 0.9302 - val_loss: 0.4483 - val_accuracy: 0.8653
Epoch 7/50
460/460 [==============================] - 16s 35ms/step - loss: 0.1973 - accuracy: 0.9301 - val_loss: 0.3573 - val_accuracy: 0.8836
Epoch 8/50
460/460 [==============================] - 19s 41ms/step - loss: 0.1756 - accuracy: 0.9354 - val_loss: 0.3745 - val_accuracy: 0.8982
Epoch 9/50
460/460 [==============================] - 17s 38ms/step - loss: 0.1780 - accuracy: 0.9393 - val_loss: 0.3604 - val_accuracy: 0.8877
Epoch 10/50
460/460 [==============================] - 16s 35ms/step - loss: 0.1609 - accuracy: 0.9394 - val_loss: 0.3688 - val_accuracy: 0.8856
Epoch 11/50
460/460 [==============================] - 16s 35ms/step - loss: 0.1647 - accuracy: 0.9373 - val_loss: 0.3396 - val_accuracy: 0.8975
Epoch 12/50
460/460 [==============================] - 16s 35ms/step - loss: 0.1430 - accuracy: 0.9470 - val_loss: 0.4112 - val_accuracy: 0.8955
Epoch 13/50
460/460 [==============================] - 16s 34ms/step - loss: 0.1356 - accuracy: 0.9524 - val_loss: 0.5074 - val_accuracy: 0.8853
Epoch 14/50
460/460 [==============================] - 16s 34ms/step - loss: 0.1386 - accuracy: 0.9477 - val_loss: 0.3709 - val_accuracy: 0.8951
Epoch 15/50
460/460 [==============================] - 18s 38ms/step - loss: 0.1335 - accuracy: 0.9466 - val_loss: 0.4379 - val_accuracy: 0.9013
Epoch 16/50
460/460 [==============================] - 17s 38ms/step - loss: 0.1333 - accuracy: 0.9490 - val_loss: 0.3550 - val_accuracy: 0.9203
Epoch 17/50
460/460 [==============================] - 16s 34ms/step - loss: 0.1398 - accuracy: 0.9462 - val_loss: 0.3907 - val_accuracy: 0.9006
Epoch 18/50
460/460 [==============================] - 16s 36ms/step - loss: 0.1248 - accuracy: 0.9539 - val_loss: 0.3656 - val_accuracy: 0.9216
```

```
Epoch 19/50
460/460 [==============================] - 16s 35ms/step - loss: 0.1186 - accuracy: 0.9540 - val_loss: 0.5029 - val_accuracy: 0.8880
Epoch 20/50
460/460 [==============================] - 17s 38ms/step - loss: 0.1309 - accuracy: 0.9483 - val_loss: 0.4282 - val_accuracy: 0.8982
Epoch 21/50
460/460 [==============================] - 18s 38ms/step - loss: 0.1326 - accuracy: 0.9496 - val_loss: 0.4332 - val_accuracy: 0.9080
Epoch 22/50
460/460 [==============================] - 17s 37ms/step - loss: 0.1296 - accuracy: 0.9500 - val_loss: 0.6998 - val_accuracy: 0.8951
Epoch 23/50
460/460 [==============================] - 17s 36ms/step - loss: 0.1427 - accuracy: 0.9535 - val_loss: 0.5418 - val_accuracy: 0.8931
Epoch 24/50
460/460 [==============================] - 16s 34ms/step - loss: 0.1219 - accuracy: 0.9555 - val_loss: 0.6438 - val_accuracy: 0.8785
Epoch 25/50
460/460 [==============================] - 16s 35ms/step - loss: 0.1166 - accuracy: 0.9558 - val_loss: 0.5444 - val_accuracy: 0.8945
Epoch 26/50
460/460 [==============================] - 18s 40ms/step - loss: 0.1116 - accuracy: 0.9540 - val_loss: 0.5413 - val_accuracy: 0.9125
Epoch 27/50
460/460 [==============================] - 17s 36ms/step - loss: 0.1252 - accuracy: 0.9517 - val_loss: 0.5265 - val_accuracy: 0.9074
Epoch 28/50
460/460 [==============================] - 17s 36ms/step - loss: 0.1196 - accuracy: 0.9520 - val_loss: 0.5264 - val_accuracy: 0.9060
Epoch 29/50
460/460 [==============================] - 17s 37ms/step - loss: 0.1220 - accuracy: 0.9546 - val_loss: 0.6210 - val_accuracy: 0.8809
Epoch 30/50
460/460 [==============================] - 16s 36ms/step - loss: 0.1324 - accuracy: 0.9513 - val_loss: 0.5287 - val_accuracy: 0.8904
Epoch 31/50
460/460 [==============================] - 16s 35ms/step - loss: 0.1246 - accuracy: 0.9513 - val_loss: 0.5904 - val_accuracy: 0.9057
Epoch 32/50
460/460 [==============================] - 16s 36ms/step - loss: 0.1245 - accuracy: 0.9525 - val_loss: 0.5367 - val_accuracy: 0.8948
Epoch 33/50
460/460 [==============================] - 16s 35ms/step - loss: 0.1221 - accuracy: 0.9492 - val_loss: 0.5060 - val_accuracy: 0.8955
Epoch 34/50
460/460 [==============================] - 17s 36ms/step - loss: 0.1214 - accuracy: 0.9541 - val_loss: 0.5426 - val_accuracy: 0.9016
Epoch 35/50
460/460 [==============================] - 16s 35ms/step - loss: 0.1198 - accuracy: 0.9563 - val_loss: 0.5815 - val_accuracy: 0.8985
Epoch 36/50
460/460 [==============================] - 16s 36ms/step - loss: 0.1167 - accuracy: 0.9537 - val_loss: 0.5748 - val_accuracy: 0.8948
Epoch 37/50
460/460 [==============================] - 16s 35ms/step - loss: 0.1188 - accuracy: 0.9547 - val_loss: 0.5325 - val_accuracy: 0.9121
Epoch 38/50
460/460 [==============================] - 16s 35ms/step - loss: 0.1236 - accuracy: 0.9559 - val_loss: 0.4596 - val_accuracy: 0.9111
Epoch 39/50
460/460 [==============================] - 16s 35ms/step - loss: 0.1273 - accuracy: 0.9498 - val_loss: 0.5595 - val_accuracy: 0.9030
Epoch 40/50
460/460 [==============================] - 17s 36ms/step - loss: 0.1115 - accuracy: 0.9574 - val_loss: 0.5877 - val_accuracy: 0.9094
Epoch 41/50
460/460 [==============================] - 16s 35ms/step - loss: 0.1102 - accuracy: 0.9574 - val_loss: 0.5567 - val_accuracy: 0.9050
Epoch 42/50
460/460 [==============================] - 17s 36ms/step - loss: 0.1117 - accuracy: 0.9569 - val_loss: 0.4832 - val_accuracy: 0.9084
Epoch 43/50
460/460 [==============================] - 18s 40ms/step - loss: 0.1552 - accuracy: 0.9407 - val_loss: 0.5866 - val_accuracy: 0.8948
Epoch 44/50
460/460 [==============================] - 18s 39ms/step - loss: 0.1192 - accuracy: 0.9512 - val_loss: 0.5146 - val_accuracy: 0.9067
Epoch 45/50
460/460 [==============================] - 18s 40ms/step - loss: 0.1188 - accuracy: 0.9554 - val_loss: 0.4962 - val_accuracy: 0.9125
Epoch 46/50
460/460 [==============================] - 19s 41ms/step - loss: 0.1200 - accuracy: 0.9569 - val_loss: 0.4663 - val_accuracy: 0.9009
Epoch 47/50
460/460 [==============================] - 18s 39ms/step - loss: 0.1034 - accuracy: 0.9617 - val_loss: 0.4789 - val_accuracy: 0.8996
Epoch 48/50
```

```
460/460 [==============================] - 18s 40ms/step - loss: 0.1144 - accuracy: 0.9523 - val_loss: 0.4899 - val_accuracy: 0.9053
Epoch 49/50
460/460 [==============================] - 18s 39ms/step - loss: 0.1015 - accuracy: 0.9601 - val_loss: 0.4873 - val_accuracy: 0.8877
Epoch 50/50

460/460 [==============================] - 18s 40ms/step - loss: 0.1216 - accuracy: 0.9520 - val_loss: 0.5330 - val_accuracy: 0.8992
----------------------------------------------------------------------------------------------------
Val Accuracy: 0.9216151833534241  Train Accuracy: 0.957290530204773
----------------------------------------------------------------------------------------------------



----------------------------------------------------------------------------------------------------
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0         5        0                   0
SITTING                  0      369       121        0                   0
STANDING                 0       72       460        0                   0
WALKING                  0        0         6      466                   4
WALKING_DOWNSTAIRS       0        0         0        4                 410
WALKING_UPSTAIRS         0        1         3       13                  19

Pred                WALKING_UPSTAIRS
True
LAYING                            22
SITTING                            1
STANDING                           0
WALKING                           20
WALKING_DOWNSTAIRS                 6
WALKING_UPSTAIRS                 435
93/93 [==============================] - 1s 11ms/step - loss: 0.5330 - accuracy: 0.8992
```

In [21]:

```
exp4_df = pd.DataFrame(columns = ['Layer1Units','Dropout1','TrainAccuracy','ValAccuracy','Score'] ,
                       data = exp4_scores)
```

In [22]:

```
exp4_df
```

Out[22]:

	Layer1Units	Dropout1	TrainAccuracy	ValAccuracy	Score
0	32	0.25	0.957291	0.921615	[0.5329602956771851, 0.8992195725440979]

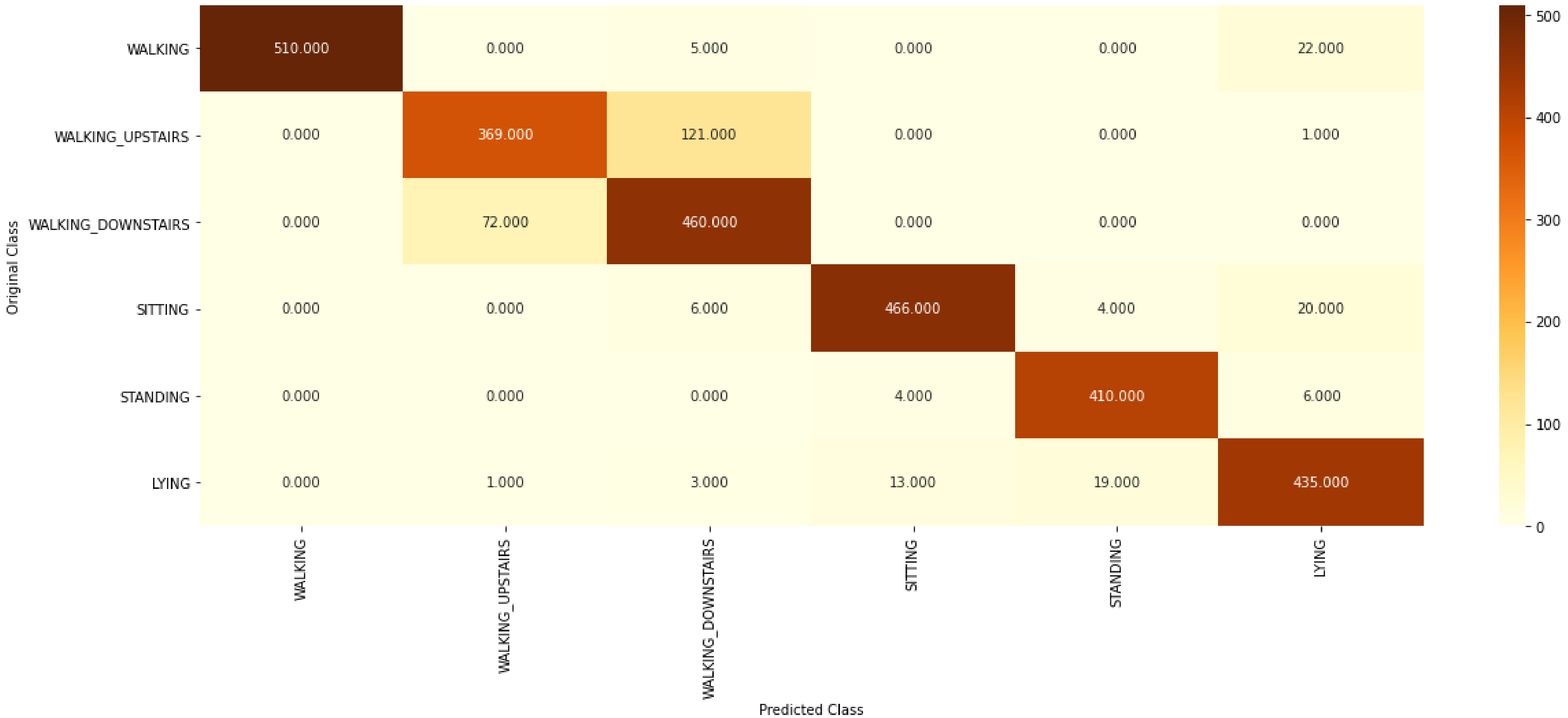
Observations :

1. Conducted Hyperparameter tuning in 4 parts.
2. found that the DL architecture with one layer of lstm (32units) and dropout 0.25 gives the highest accracy.
3. increased epochs for the best architecture, still maximum accuracy attained was 90%.

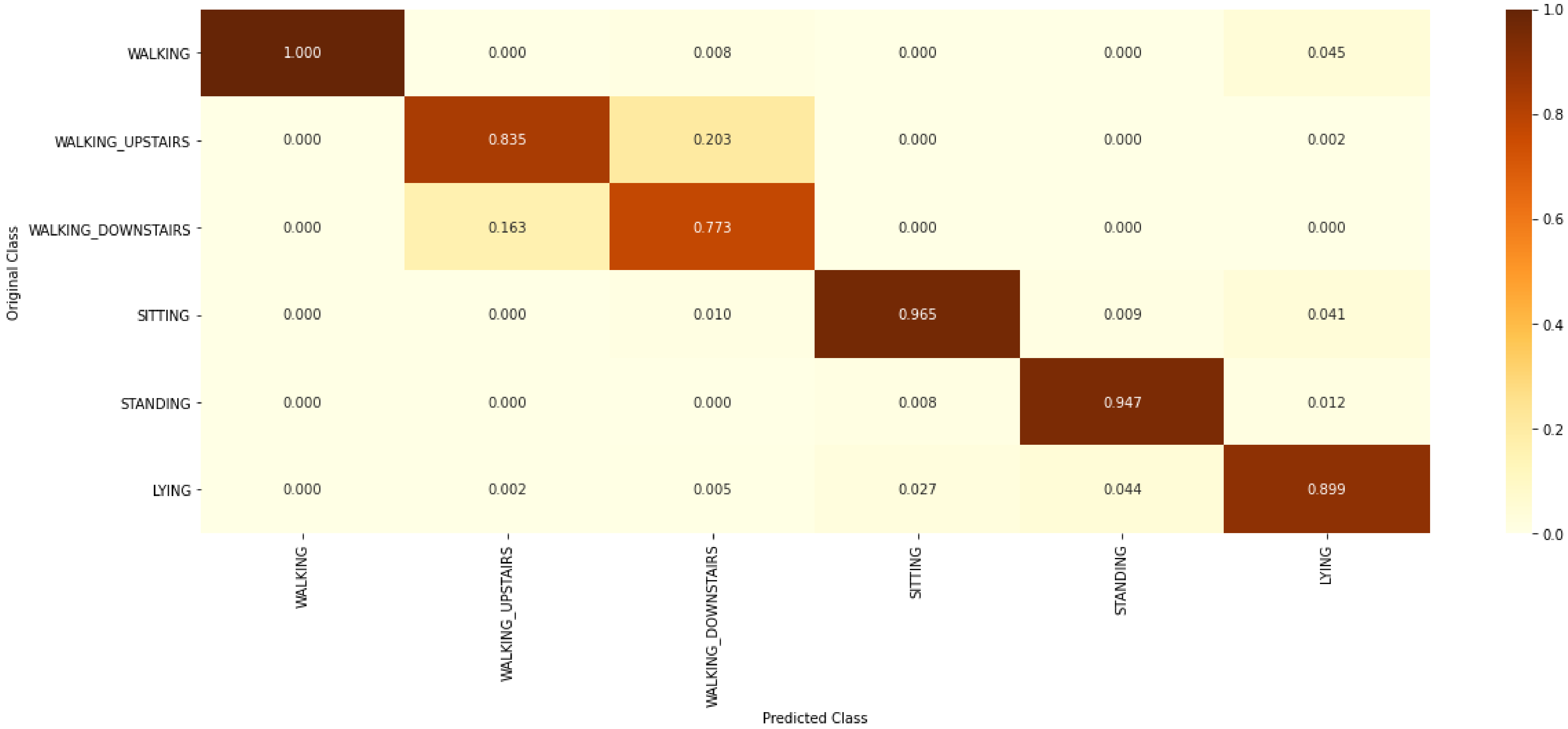
In [28]:

```
1 plot_confusion_matrix(ytest,ypred)
```

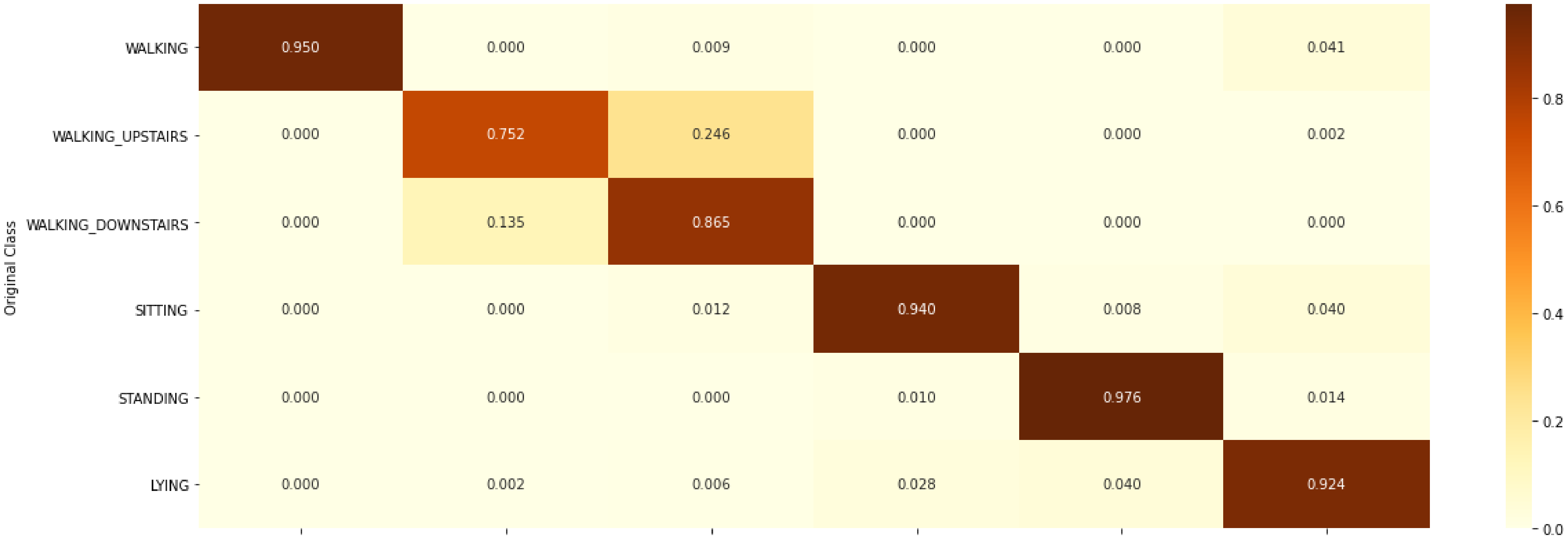
```
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



Model built without expert crafted fetures performs fairly well with 90% accuracy with layer 1 . Model able to provide good precisiona nd recall for standing,sitting,lying,walking.