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

- ***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.

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

In [1]:

```
import numpy as np
import pandas as pd

# get the features from the file features.txt
features = list()
with open('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 [2]:

```
import os
if not os.path.exists('train.csv'):

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

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

    y_train = pd.read_csv('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()
    train.to_csv('train.csv',index=False,header=True)
    print('File has created.')
else:
```

```
else:
    train = pd.read_csv('train.csv')
    print('Data is loaded.')
```

```
Data is loaded.
```

In [3]:

```
train.shape
```

Out[3]:

```
(7352, 564)
```

In [4]:

```
train.head()
```

Out[4]:

	tBodyAccmeanX	tBodyAccmeanY	tBodyAccmeanZ	tBodyAccstdX	tBodyAccstdY	tBodyAccstdZ	tBodyAccmadX	tBodyAccmadY	t
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	

5 rows × 564 columns

Obtain the test data

In [5]:

```
import os

if not os.path.exists('test.csv'):
    # get the data from txt files to pandas dataffame
    X_test = pd.read_csv('X_test.txt', delim_whitespace=True, header=None, names=features)

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

    # get y labels from the txt file
    y_test = pd.read_csv('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.to_csv('test.csv',index=False,header=True)
else:
    test = pd.read_csv('test.csv')
print('Done.')
```

```
Done.
```

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
```

In [ ]:

```

```

3. Check for data imbalance

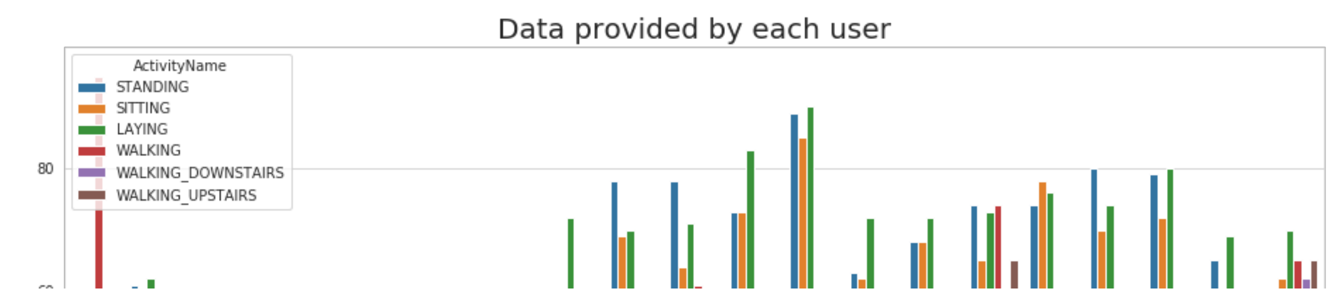
In [9]:

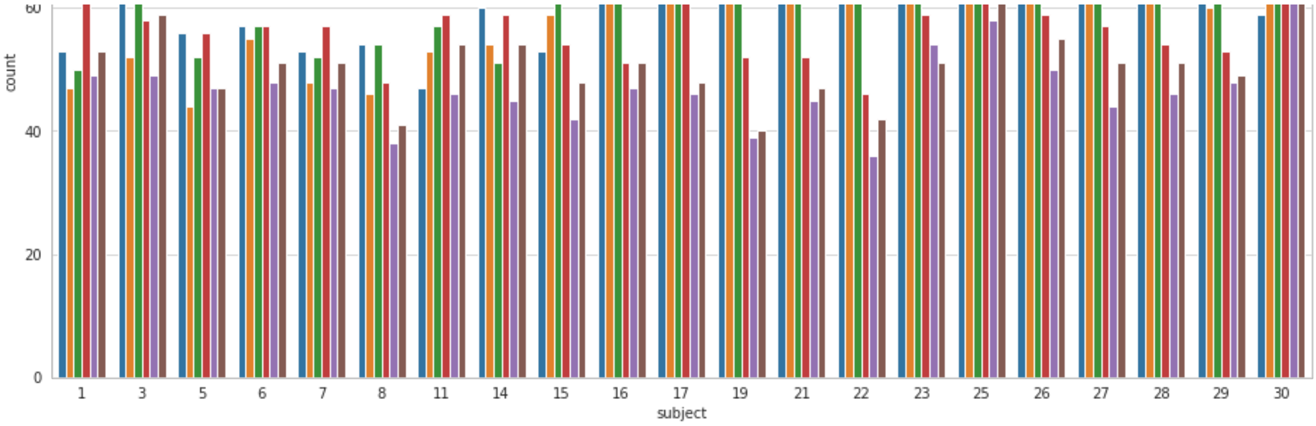
```
import matplotlib.pyplot as plt
import seaborn as sns

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

```
C:\Users\sanjeev\Anaconda3\lib\site-packages\statsmodels\tools\_testing.py:19: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [10]:

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

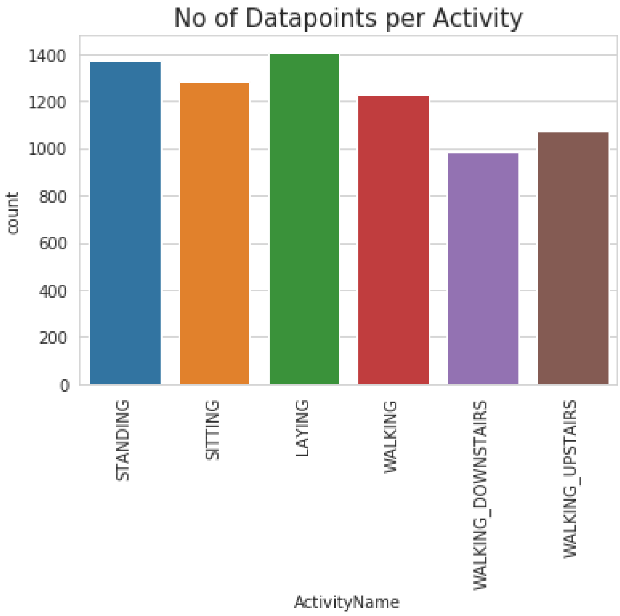




> We have got almost same number of reading from all the subjects

In [11]:

```
plt.title('No of Datapoints per Activity', fontsize=15)
sns.countplot(train.ActivityName)
plt.xticks(rotation=90)
plt.show()
```



Observation

> Our data is well balanced (almost)

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 = 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('train.csv', index=False)
test.to_csv('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

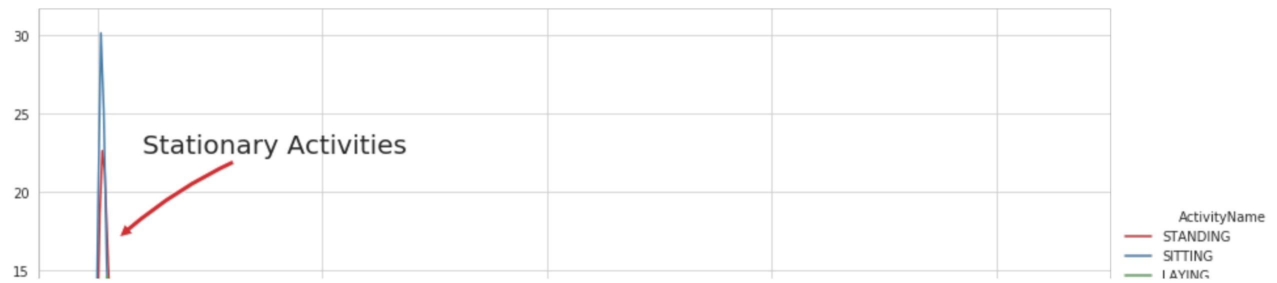
- **Static and Dynamic Activities**
  - In static activities (sit, stand, lie down) motion information will not be very useful.
  - In the dynamic activities (Walking, WalkingUpstairs,WalkingDownstairs) motion info will be significant.

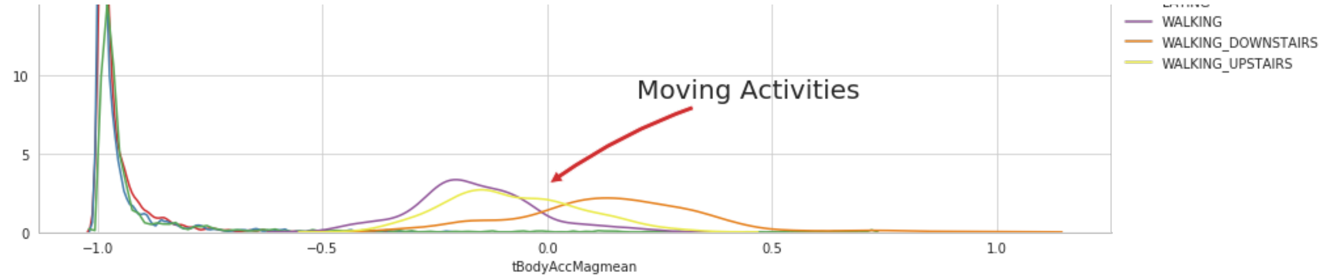
2. Stationary and Moving activities are completely different

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", xy=(-0.956,17), xytext=(-0.9, 23), size=20,\
            va='center', ha='left',\
            arrowprops=dict(arrowstyle="simple",connectionstyle="arc3,rad=0.1"))

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

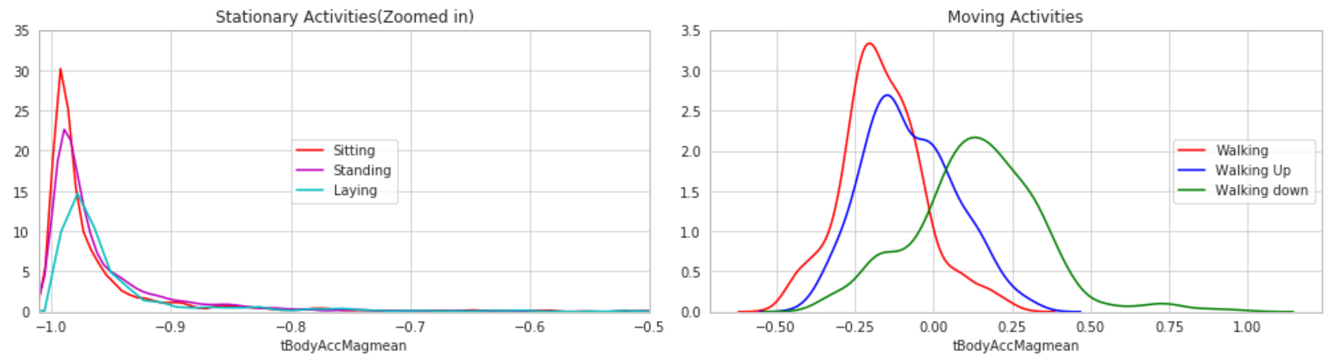
```
C:\Users\sanjeev\Anaconda3\lib\site-packages\seaborn\axisgrid.py:230: UserWarning: The `size` para
mter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```





In [15]:

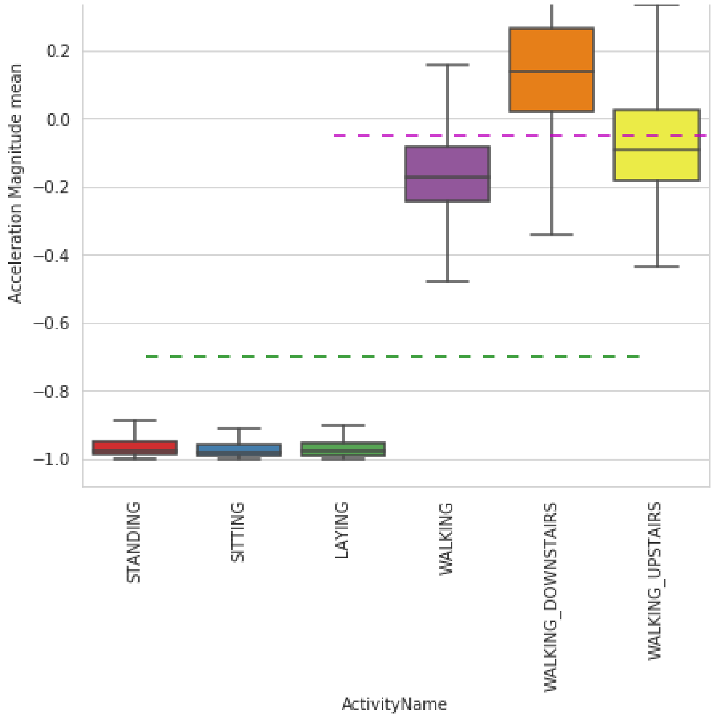
```
# for plotting purposes taking datapoints of each activity to a different dataframe
df1 = train[train['Activity']==1]
df2 = train[train['Activity']==2]
df3 = train[train['Activity']==3]
df4 = train[train['Activity']==4]
df5 = train[train['Activity']==5]
df6 = train[train['Activity']==6]

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()
```



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('Acceleration Magnitude mean')
plt.axhline(y=-0.7, xmin=0.1, xmax=0.9,dashes=(5,5), c='g')
plt.axhline(y=-0.05, xmin=0.4, dashes=(5,5), c='m')
plt.xticks(rotation=90)
plt.show()
```



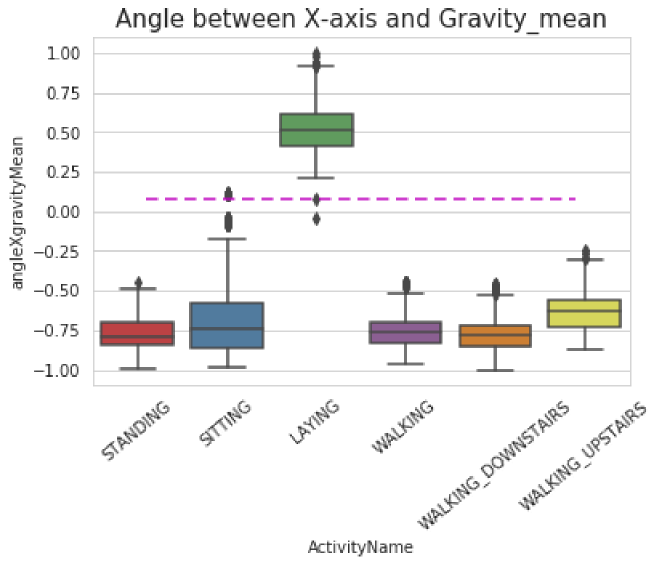
Observations:

- 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 also matters

In [17]:

```
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('Angle between X-axis and Gravity_mean', fontsize=15)
plt.xticks(rotation = 40)
plt.show()
```

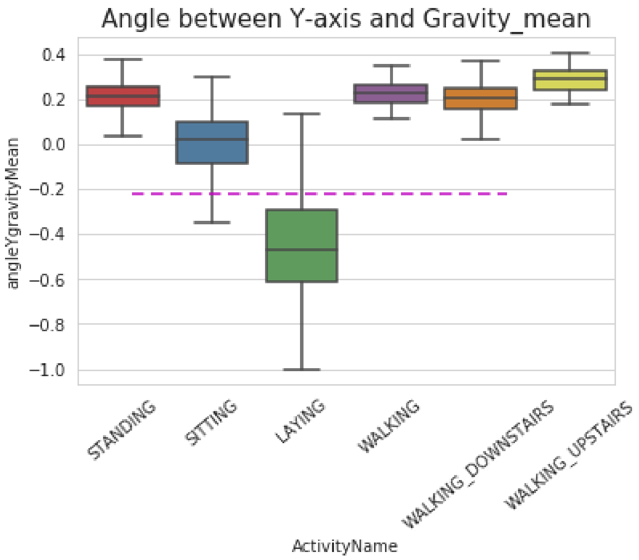


Observations:

- 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]:

```
sns.boxplot(x='ActivityName', y='angleYgravityMean', data = train, showfliers=False)
plt.title('Angle between Y-axis and Gravity_mean', fontsize=15)
plt.xticks(rotation = 40)
plt.axhline(y=-0.22, xmin=0.1, xmax=0.8, dashes=(5,3), c='m')
plt.show()
```



Apply t-sne on the data

In [19]:

```
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns
```

In [20]:

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

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

    for index,perplexity in enumerate(perplexities):
        # perform t-sne
        print('\nperforming tsne with perplexity {} and with {} iterations at max'.format(perplexit
y, n_iter))
        X_reduced = TSNE(verbose=2, perplexity=perplexity).fit_transform(X_data)
        print('Done..')

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

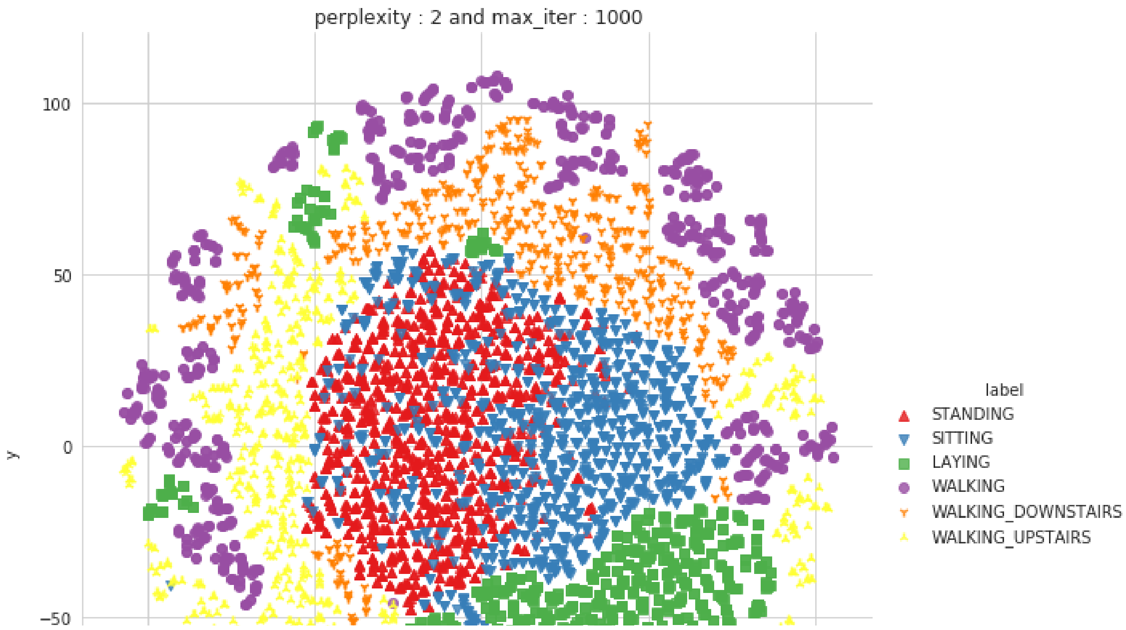
In [21]:

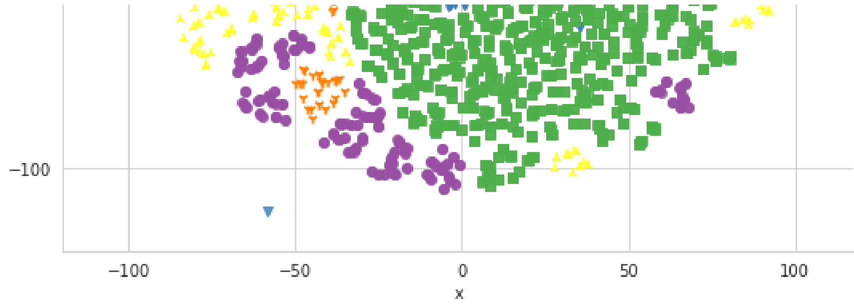
```
X_pre_tsne = train.drop(['subject', 'Activity','ActivityName'], axis=1)
y_pre_tsne = train['ActivityName']
perform_tsne(X_data = X_pre_tsne,y_data=y_pre_tsne, perplexities =[2,5,10,20,50])
```

```
performing tsne with perplexity 2 and with 1000 iterations at max
[t-SNE] Computing 7 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.486s...
[t-SNE] Computed neighbors for 7352 samples in 50.150s...
[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: 0.635855
[t-SNE] Computed conditional probabilities in 0.125s
[t-SNE] Iteration 50: error = 124.7682190, gradient norm = 0.0257704 (50 iterations in 6.676s)
[t-SNE] Iteration 100: error = 106.7810440, gradient norm = 0.0283834 (50 iterations in 4.605s)
[t-SNE] Iteration 150: error = 100.6281281, gradient norm = 0.0191540 (50 iterations in 3.739s)
[t-SNE] Iteration 200: error = 97.2982483, gradient norm = 0.0169978 (50 iterations in 3.673s)
[t-SNE] Iteration 250: error = 95.0379333, gradient norm = 0.0142829 (50 iterations in 3.780s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 95.037933
[t-SNE] Iteration 300: error = 4.1139207, gradient norm = 0.0015608 (50 iterations in 3.263s)
[t-SNE] Iteration 350: error = 3.2067728, gradient norm = 0.0010018 (50 iterations in 3.223s)
[t-SNE] Iteration 400: error = 2.7781920, gradient norm = 0.0007167 (50 iterations in 4.415s)
[t-SNE] Iteration 450: error = 2.5144770, gradient norm = 0.0005642 (50 iterations in 3.376s)
[t-SNE] Iteration 500: error = 2.3311925, gradient norm = 0.0004784 (50 iterations in 3.117s)
[t-SNE] Iteration 550: error = 2.1932833, gradient norm = 0.0004137 (50 iterations in 2.778s)
[t-SNE] Iteration 600: error = 2.0841417, gradient norm = 0.0003660 (50 iterations in 2.717s)
[t-SNE] Iteration 650: error = 1.9945242, gradient norm = 0.0003307 (50 iterations in 2.708s)
[t-SNE] Iteration 700: error = 1.9192704, gradient norm = 0.0003048 (50 iterations in 2.701s)
[t-SNE] Iteration 750: error = 1.8545545, gradient norm = 0.0002771 (50 iterations in 2.962s)
[t-SNE] Iteration 800: error = 1.7978274, gradient norm = 0.0002596 (50 iterations in 2.992s)
[t-SNE] Iteration 850: error = 1.7478549, gradient norm = 0.0002398 (50 iterations in 2.923s)
[t-SNE] Iteration 900: error = 1.7033331, gradient norm = 0.0002259 (50 iterations in 2.985s)
[t-SNE] Iteration 950: error = 1.6635573, gradient norm = 0.0002107 (50 iterations in 2.827s)
[t-SNE] Iteration 1000: error = 1.6273080, gradient norm = 0.0001986 (50 iterations in 2.984s)
[t-SNE] KL divergence after 1000 iterations: 1.627308
Done..
Creating plot for this t-sne visualization..
```

```
C:\Users\sanjeev\Anaconda3\lib\site-packages\seaborn\regression.py:546: UserWarning: The `size` pa
ramter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

```
saving this plot as image in present working directory...
```

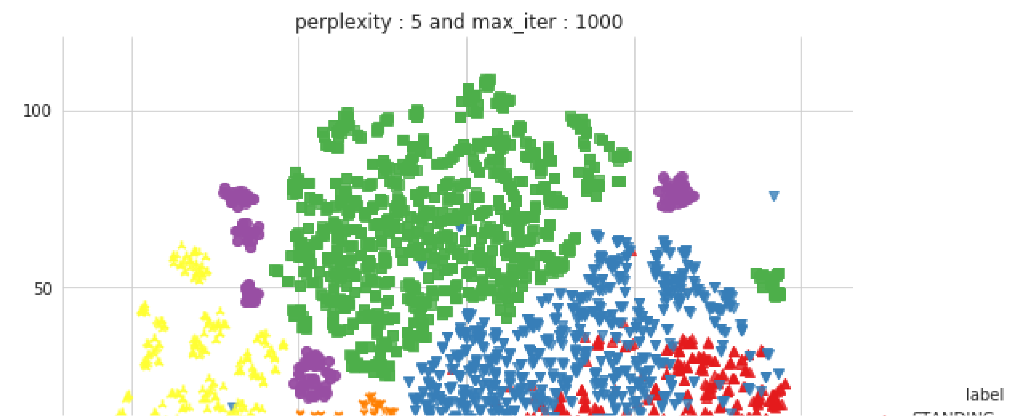


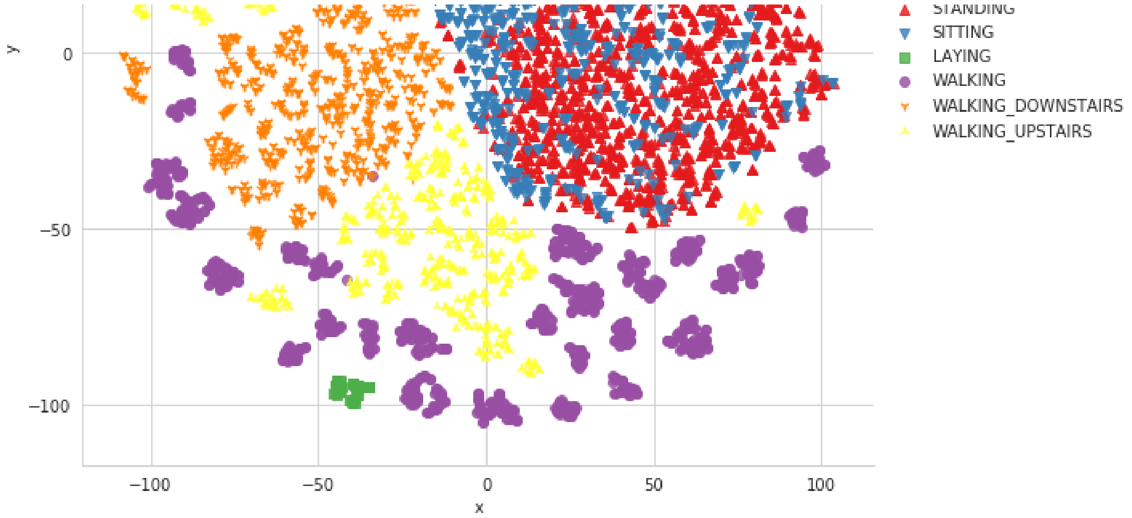



```
Done

performing tsne with perplexity 5 and with 1000 iterations at max
[t-SNE] Computing 16 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.312s...
[t-SNE] Computed neighbors for 7352 samples in 47.514s...
[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 0.187s
[t-SNE] Iteration 50: error = 114.0555573, gradient norm = 0.0234660 (50 iterations in 11.069s)
[t-SNE] Iteration 100: error = 97.8461609, gradient norm = 0.0152510 (50 iterations in 3.155s)
[t-SNE] Iteration 150: error = 93.1312790, gradient norm = 0.0089201 (50 iterations in 2.668s)
[t-SNE] Iteration 200: error = 91.1070709, gradient norm = 0.0067680 (50 iterations in 2.569s)
[t-SNE] Iteration 250: error = 89.9217148, gradient norm = 0.0053157 (50 iterations in 2.544s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 89.921715
[t-SNE] Iteration 300: error = 3.5675917, gradient norm = 0.0014618 (50 iterations in 2.594s)
[t-SNE] Iteration 350: error = 2.8087602, gradient norm = 0.0007490 (50 iterations in 2.582s)
[t-SNE] Iteration 400: error = 2.4282768, gradient norm = 0.0005215 (50 iterations in 2.866s)
[t-SNE] Iteration 450: error = 2.2109582, gradient norm = 0.0004047 (50 iterations in 2.757s)
[t-SNE] Iteration 500: error = 2.0662291, gradient norm = 0.0003309 (50 iterations in 2.678s)
[t-SNE] Iteration 550: error = 1.9611422, gradient norm = 0.0002805 (50 iterations in 2.672s)
[t-SNE] Iteration 600: error = 1.8802474, gradient norm = 0.0002452 (50 iterations in 2.647s)
[t-SNE] Iteration 650: error = 1.8152393, gradient norm = 0.0002195 (50 iterations in 2.686s)
[t-SNE] Iteration 700: error = 1.7617742, gradient norm = 0.0002005 (50 iterations in 2.685s)
[t-SNE] Iteration 750: error = 1.7167171, gradient norm = 0.0001777 (50 iterations in 2.681s)
[t-SNE] Iteration 800: error = 1.6778905, gradient norm = 0.0001652 (50 iterations in 2.740s)
[t-SNE] Iteration 850: error = 1.6442558, gradient norm = 0.0001543 (50 iterations in 2.688s)
[t-SNE] Iteration 900: error = 1.6144943, gradient norm = 0.0001440 (50 iterations in 2.708s)
[t-SNE] Iteration 950: error = 1.5878861, gradient norm = 0.0001344 (50 iterations in 2.689s)
[t-SNE] Iteration 1000: error = 1.5640718, gradient norm = 0.0001269 (50 iterations in 2.759s)
[t-SNE] KL divergence after 1000 iterations: 1.564072
Done..
Creating plot for this t-sne visualization..
```

```
C:\Users\sanjeev\Anaconda3\lib\site-packages\seaborn\regression.py:546: UserWarning: The `size` pa
ramter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

```
saving this plot as image in present working directory...
```

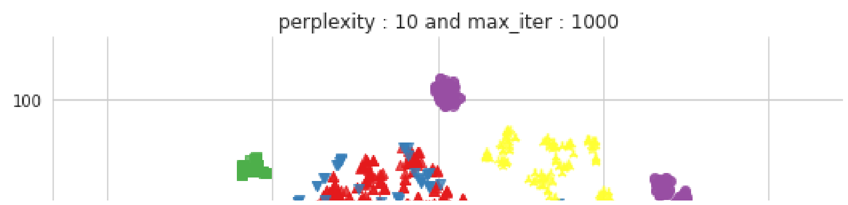


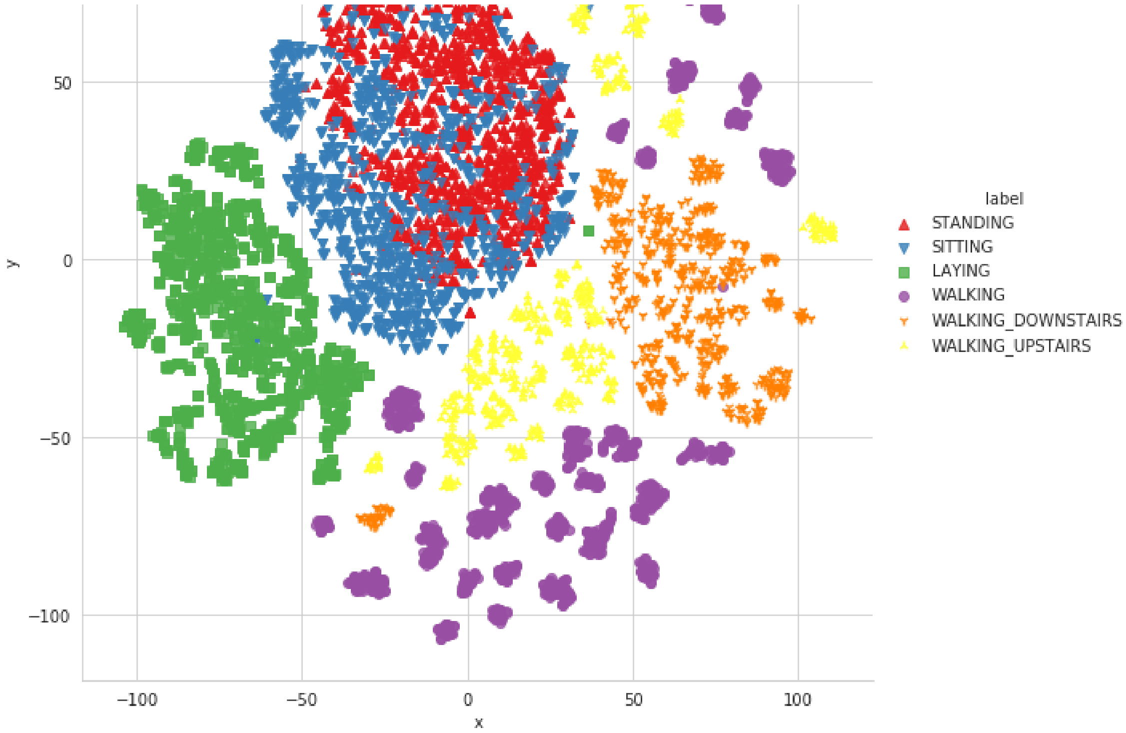



```
Done

performing tsne with perplexity 10 and with 1000 iterations at max
[t-SNE] Computing 31 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.293s...
[t-SNE] Computed neighbors for 7352 samples in 47.773s...
[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 0.125s
[t-SNE] Iteration 50: error = 105.8732071, gradient norm = 0.0164976 (50 iterations in 4.858s)
[t-SNE] Iteration 100: error = 90.5844574, gradient norm = 0.0112697 (50 iterations in 3.465s)
[t-SNE] Iteration 150: error = 87.5589371, gradient norm = 0.0062893 (50 iterations in 3.095s)
[t-SNE] Iteration 200: error = 86.2989120, gradient norm = 0.0065243 (50 iterations in 3.175s)
[t-SNE] Iteration 250: error = 85.5826340, gradient norm = 0.0029128 (50 iterations in 3.116s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.582634
[t-SNE] Iteration 300: error = 3.1351471, gradient norm = 0.0013860 (50 iterations in 2.987s)
[t-SNE] Iteration 350: error = 2.4937270, gradient norm = 0.0006509 (50 iterations in 2.771s)
[t-SNE] Iteration 400: error = 2.1737823, gradient norm = 0.0004212 (50 iterations in 2.774s)
[t-SNE] Iteration 450: error = 1.9893734, gradient norm = 0.0003163 (50 iterations in 2.786s)
[t-SNE] Iteration 500: error = 1.8710037, gradient norm = 0.0002546 (50 iterations in 2.808s)
[t-SNE] Iteration 550: error = 1.7874155, gradient norm = 0.0002133 (50 iterations in 2.807s)
[t-SNE] Iteration 600: error = 1.7246299, gradient norm = 0.0001839 (50 iterations in 2.817s)
[t-SNE] Iteration 650: error = 1.6753068, gradient norm = 0.0001621 (50 iterations in 2.833s)
[t-SNE] Iteration 700: error = 1.6357390, gradient norm = 0.0001437 (50 iterations in 2.832s)
[t-SNE] Iteration 750: error = 1.6034117, gradient norm = 0.0001309 (50 iterations in 2.871s)
[t-SNE] Iteration 800: error = 1.5759659, gradient norm = 0.0001212 (50 iterations in 2.855s)
[t-SNE] Iteration 850: error = 1.5530144, gradient norm = 0.0001120 (50 iterations in 2.849s)
[t-SNE] Iteration 900: error = 1.5330607, gradient norm = 0.0001051 (50 iterations in 2.846s)
[t-SNE] Iteration 950: error = 1.5159706, gradient norm = 0.0000987 (50 iterations in 2.836s)
[t-SNE] Iteration 1000: error = 1.5010815, gradient norm = 0.0000945 (50 iterations in 2.844s)
[t-SNE] KL divergence after 1000 iterations: 1.501081
Done..
Creating plot for this t-sne visualization..
```

```
C:\Users\sanjeev\Anaconda3\lib\site-packages\seaborn\regression.py:546: UserWarning: The `size` pa
ramter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

```
saving this plot as image in present working directory...
```

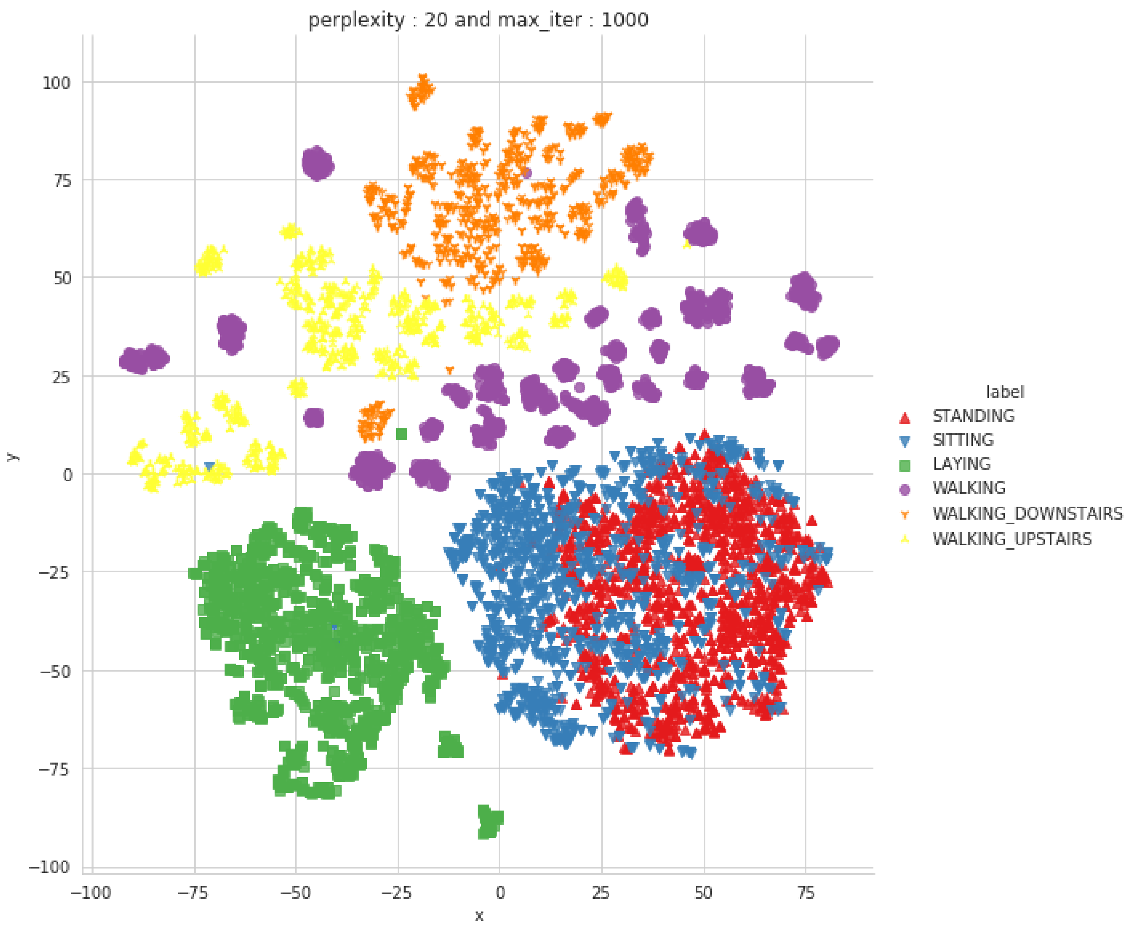





```
Done

performing tsne with perplexity 20 and with 1000 iterations at max
[t-SNE] Computing 61 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.291s...
[t-SNE] Computed neighbors for 7352 samples in 48.704s...
[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 0.232s
[t-SNE] Iteration 50: error = 98.0124283, gradient norm = 0.0108954 (50 iterations in 6.863s)
[t-SNE] Iteration 100: error = 84.0378571, gradient norm = 0.0076726 (50 iterations in 4.252s)
[t-SNE] Iteration 150: error = 81.9204407, gradient norm = 0.0040294 (50 iterations in 3.682s)
[t-SNE] Iteration 200: error = 81.1675034, gradient norm = 0.0026958 (50 iterations in 3.630s)
[t-SNE] Iteration 250: error = 80.7760849, gradient norm = 0.0018361 (50 iterations in 3.480s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 80.776085
[t-SNE] Iteration 300: error = 2.6949992, gradient norm = 0.0013013 (50 iterations in 3.298s)
[t-SNE] Iteration 350: error = 2.1627953, gradient norm = 0.0005764 (50 iterations in 3.126s)
[t-SNE] Iteration 400: error = 1.9131525, gradient norm = 0.0003471 (50 iterations in 3.110s)
[t-SNE] Iteration 450: error = 1.7670326, gradient norm = 0.0002475 (50 iterations in 3.125s)
[t-SNE] Iteration 500: error = 1.6731824, gradient norm = 0.0001917 (50 iterations in 3.142s)
[t-SNE] Iteration 550: error = 1.6089791, gradient norm = 0.0001578 (50 iterations in 3.138s)
[t-SNE] Iteration 600: error = 1.5624276, gradient norm = 0.0001320 (50 iterations in 3.131s)
[t-SNE] Iteration 650: error = 1.5270309, gradient norm = 0.0001165 (50 iterations in 3.165s)
[t-SNE] Iteration 700: error = 1.4993186, gradient norm = 0.0001042 (50 iterations in 3.198s)
[t-SNE] Iteration 750: error = 1.4771080, gradient norm = 0.0000953 (50 iterations in 3.230s)
[t-SNE] Iteration 800: error = 1.4592550, gradient norm = 0.0000859 (50 iterations in 3.201s)
[t-SNE] Iteration 850: error = 1.4442073, gradient norm = 0.0000859 (50 iterations in 3.218s)
[t-SNE] Iteration 900: error = 1.4327312, gradient norm = 0.0000769 (50 iterations in 3.213s)
[t-SNE] Iteration 950: error = 1.4231975, gradient norm = 0.0000739 (50 iterations in 3.210s)
[t-SNE] Iteration 1000: error = 1.4148101, gradient norm = 0.0000704 (50 iterations in 3.216s)
[t-SNE] KL divergence after 1000 iterations: 1.414810
Done..
Creating plot for this t-sne visualization..
```

```
C:\Users\sanjeev\Anaconda3\lib\site-packages\seaborn\regression.py:546: UserWarning: The `size` pa
ramter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

```
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 0.310s...
[t-SNE] Computed neighbors for 7352 samples in 50.319s...
[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 0.564s
[t-SNE] Iteration 50: error = 84.6528091, gradient norm = 0.0364481 (50 iterations in 5.905s)
[t-SNE] Iteration 100: error = 75.5724411, gradient norm = 0.0039807 (50 iterations in 4.684s)
[t-SNE] Iteration 150: error = 74.6383209, gradient norm = 0.0020514 (50 iterations in 4.149s)
[t-SNE] Iteration 200: error = 74.2526474, gradient norm = 0.0015941 (50 iterations in 4.158s)
[t-SNE] Iteration 250: error = 74.0653610, gradient norm = 0.0012683 (50 iterations in 4.334s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 74.065361
[t-SNE] Iteration 300: error = 2.1436260, gradient norm = 0.0011739 (50 iterations in 4.055s)
[t-SNE] Iteration 350: error = 1.7512568, gradient norm = 0.0004792 (50 iterations in 5.218s)
[t-SNE] Iteration 400: error = 1.5849240, gradient norm = 0.0002780 (50 iterations in 4.103s)
[t-SNE] Iteration 450: error = 1.4920912, gradient norm = 0.0001872 (50 iterations in 3.921s)
[t-SNE] Iteration 500: error = 1.4325999, gradient norm = 0.0001390 (50 iterations in 3.939s)
[t-SNE] Iteration 550: error = 1.3918090, gradient norm = 0.0001100 (50 iterations in 4.017s)
[t-SNE] Iteration 600: error = 1.3629713, gradient norm = 0.0000935 (50 iterations in 3.992s)
[t-SNE] Iteration 650: error = 1.3419070, gradient norm = 0.0000830 (50 iterations in 3.955s)
[t-SNE] Iteration 700: error = 1.3265507, gradient norm = 0.0000743 (50 iterations in 3.937s)
[t-SNE] Iteration 750: error = 1.3150469, gradient norm = 0.0000684 (50 iterations in 3.960s)
[t-SNE] Iteration 800: error = 1.3062477, gradient norm = 0.0000644 (50 iterations in 3.961s)
[t-SNE] Iteration 850: error = 1.2991018, gradient norm = 0.0000616 (50 iterations in 3.948s)
[t-SNE] Iteration 900: error = 1.2936642, gradient norm = 0.0000620 (50 iterations in 3.975s)
[t-SNE] Iteration 950: error = 1.2894293, gradient norm = 0.0000592 (50 iterations in 4.081s)
[t-SNE] Iteration 1000: error = 1.2861389, gradient norm = 0.0000545 (50 iterations in 3.997s)
[t-SNE] KL divergence after 1000 iterations: 1.286139
Done..
Creating plot for this t-sne visualization..
```

```
C:\Users\sanjeev\Anaconda3\lib\site-packages\seaborn\regression.py:546: UserWarning: The `size` pa
ramter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

```
saving this plot as image in present working directory...
```


```
Done
```

Obtain the train and test data

In [22]:

```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
print(train.shape, test.shape)
```

```
(7352, 564) (2947, 564)
```

In [23]:

```
train.head()
```

Out[23]:

	tBodyAccmeanX	tBodyAccmeanY	tBodyAccmeanZ	tBodyAccstdX	tBodyAccstdY	tBodyAccstdZ	tBodyAccmadX	tBodyAccmadY	t
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	

5 rows × 564 columns

In [24]:

```
# get X_train and y_train from csv files
X_train = train.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_train = train.ActivityName
```

In [25]:

```
# get X_test and y_test from test csv file
X_test = test.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_test = test.ActivityName
```

In [26]:

```
print('X_train and y_train : ({},{})'.format(X_train.shape, y_train.shape))
print('X_test  and y_test  : ({},{})'.format(X_test.shape, y_test.shape))
```

```
X_train and y_train : ((7352, 561),(7352,))
X_test  and y_test  : ((2947, 561),(2947,))
```

Let's model with our data

Labels that are useful in plotting confusion matrix

In [32]:

```
labels=['LAYING', 'SITTING','STANDING','WALKING','WALKING_DOWNSTAIRS','WALKING_UPSTAIRS']
```

Function to plot the confusion matrix

In [27]:

```
import itertools
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
plt.rcParams["font.family"] = 'DejaVu Sans'

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Generic function to run any model specified

In [28]:

```python
from datetime import datetime
def perform_model(model, X_train, y_train, X_test, y_test, class_labels, cm_normalize=True, \
                 print_cm=True, cm_cmap=plt.cm.Greens):


    # to store results at various phases
    results = dict()

    # time at which model starts training
    train_start_time = datetime.now()
    print('training the model..')
    model.fit(X_train, y_train)
    print('Done \n \n')
    train_end_time = datetime.now()
    results['training_time'] =  train_end_time - train_start_time
    print('training_time(HH:MM:SS.ms) - {}\n\n'.format(results['training_time']))


    # predict test data
    print('Predicting test data')
    test_start_time = datetime.now()
    y_pred = model.predict(X_test)
    test_end_time = datetime.now()
    print('Done \n \n')
    results['testing_time'] = test_end_time - test_start_time
    print('testing time(HH:MM:SS:ms) - {}\n\n'.format(results['testing_time']))
    results['predicted'] = y_pred


    # calculate overall accuracty of the model
    accuracy = metrics.accuracy_score(y_true=y_test, y_pred=y_pred)
    # store accuracy in results
    results['accuracy'] = accuracy
    print('---------------------')
    print('|      Accuracy      |')
    print('---------------------')
    print('\n    {}\n\n'.format(accuracy))


    # confusion matrix
    cm = metrics.confusion_matrix(y_test, y_pred)
    results['confusion_matrix'] = cm
    if print_cm:
        print('--------------------')
        print('| Confusion Matrix |')
        print('--------------------')
        print('\n {}'.format(cm))

    # plot confusin matrix
    plt.figure(figsize=(8,8))
    plt.grid(b=False)
    plot_confusion_matrix(cm, classes=class_labels, normalize=True, title='Normalized confusion
matrix', cmap = cm_cmap)
    plt.show()

    # get classification report
    print('-------------------------')
    print('| Classifiction Report |')
    print('-------------------------')
    classification_report = metrics.classification_report(y_test, y_pred)
    # store report in results
    results['classification_report'] = classification_report
    print(classification_report)

    # add the trained  model to the results
    results['model'] = model

    return results
```

Method to print the gridsearch Attributes

In [29]:

```
def print_grid_search_attributes(model):
    # Estimator that gave highest score among all the estimators formed in GridSearch
    print('--------------------------')
    print('|      Best Estimator     |')
    print('--------------------------')
    print('\n\t{}\n'.format(model.best_estimator_))


    # parameters that gave best results while performing grid search
    print('--------------------------')
    print('|     Best parameters     |')
    print('--------------------------')
    print('\tParameters of best estimator : \n\n\t{}\n'.format(model.best_params_))


    #  number of cross validation splits
    print('---------------------------------')
    print('|   No of CrossValidation sets   |')
    print('--------------------------------')
    print('\n\tTotal numbre of cross validation sets: {}\n'.format(model.n_splits_))


    # Average cross validated score of the best estimator, from the Grid Search
    print('--------------------------')
    print('|        Best Score       |')
    print('--------------------------')
    print('\n\tAverage Cross Validate scores of best estimator :
\n\n\t{}\n'.format(model.best_score_))
```

1. Logistic Regression with Grid Search

In [30]:

```
from sklearn import linear_model
from sklearn import metrics

from sklearn.model_selection import GridSearchCV
```

In [33]:

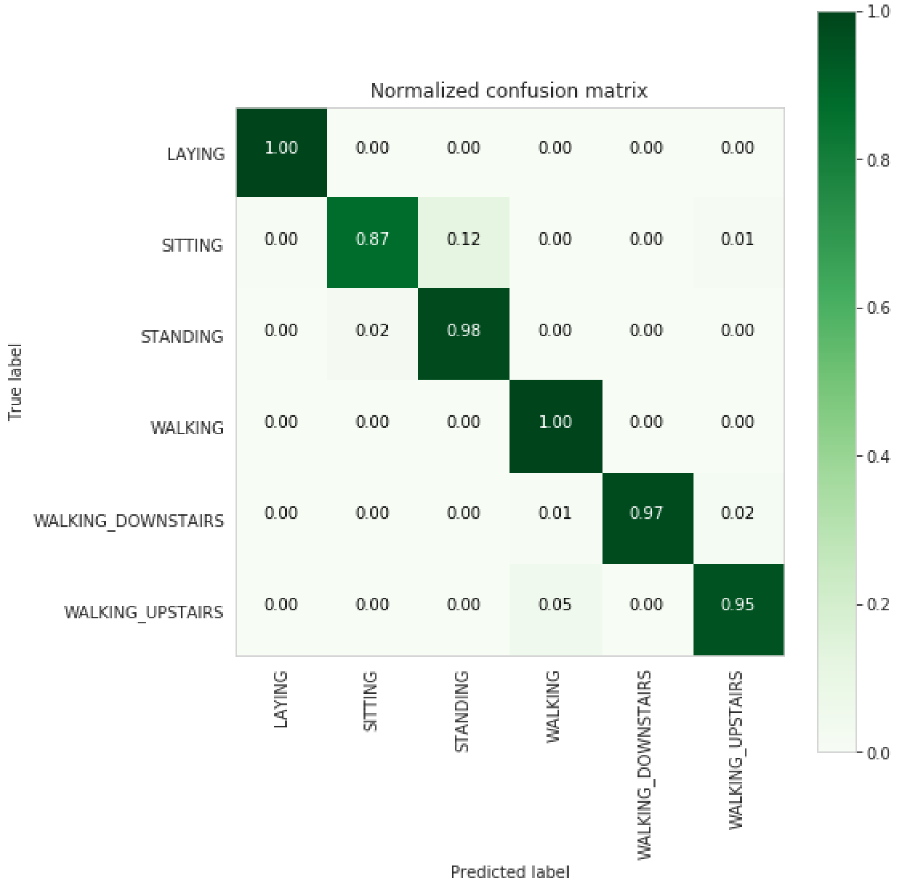
```
# start Grid search
parameters = {'C':[0.01, 0.1, 1, 10, 20, 30], 'penalty':['l2','l1']}
log_reg = linear_model.LogisticRegression()
log_reg_grid = GridSearchCV(log_reg, param_grid=parameters, cv=3, verbose=1, n_jobs=-1)
log_reg_grid_results =  perform_model(log_reg_grid, X_train, y_train, X_test,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

```
training the model..
Fitting 3 folds for each of 12 candidates, totalling 36 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  36 out of  36 | elapsed:  1.2min finished
C:\Users\sanjeev\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning:
Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\sanjeev\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:469: FutureWarning:
Default multi_class will be changed to 'auto' in 0.22. Specify the multi_class option to silence t
his warning.
  "this warning.", FutureWarning)
```

```
Done


training_time(HH:MM:SS.ms) - 0:01:22.522831


Predicting test data
Done
```

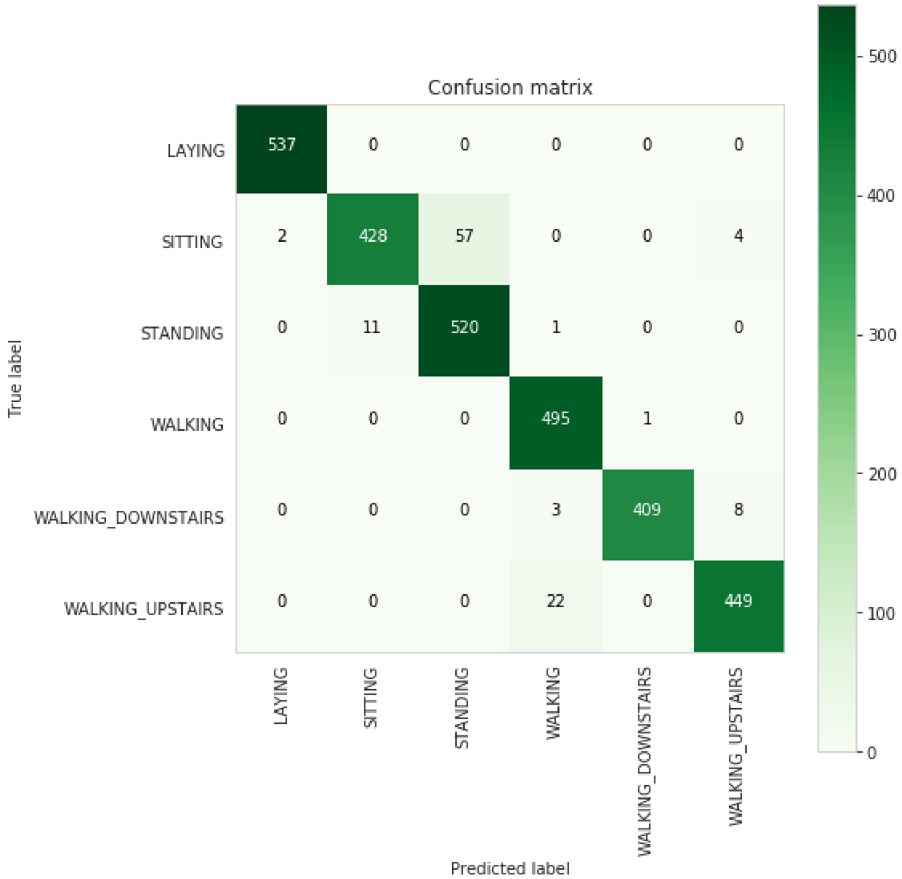
```
Done


testing time(HH:MM:SS:ms) - 0:00:00.477929


---------------------
|      Accuracy      |
---------------------

    0.9630132337970818


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  2 428  57   0   0   4]
 [  0  11 520   1   0   0]
 [  0   0   0 495   1   0]
 [  0   0   0   3 409   8]
 [  0   0   0  22   0 449]]
```




```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.97      0.87      0.92       491
          STANDING       0.90      0.98      0.94       532
           WALKING       0.95      1.00      0.97       496
WALKING_DOWNSTAIRS       1.00      0.97      0.99       420
  WALKING_UPSTAIRS       0.97      0.95      0.96       471

          accuracy                           0.96      2947
         macro avg       0.97      0.96      0.96      2947
      weighted avg       0.96      0.96      0.96      2947
```

In [34]:

```
plt.figure(figsize=(8,8))
plt.grid(b=False)
plot_confusion_matrix(log_reg_grid_results['confusion_matrix'], classes=labels, cmap=plt.cm.Greens
, )
plt.show()
```



In [35]:

```
# observe the attributes of the model
print_grid_search_attributes(log_reg_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
--------------------------

 LogisticRegression(C=30, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)

--------------------------
|     Best parameters     |
--------------------------
 Parameters of best estimator :

 {'C': 30, 'penalty': 'l2'}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

 Average Cross Validate scores of best estimator :
```

```
0.9458650707290533
```

2. Linear SVC with GridSearch

In [36]:

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

In [37]:

```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 = perform_model(lr_svc_grid, X_train, y_train, X_test, y_test, class_labels=lab
els)
```

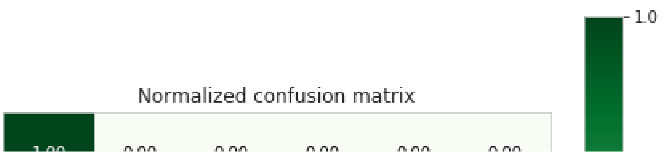
```
training the model..
```

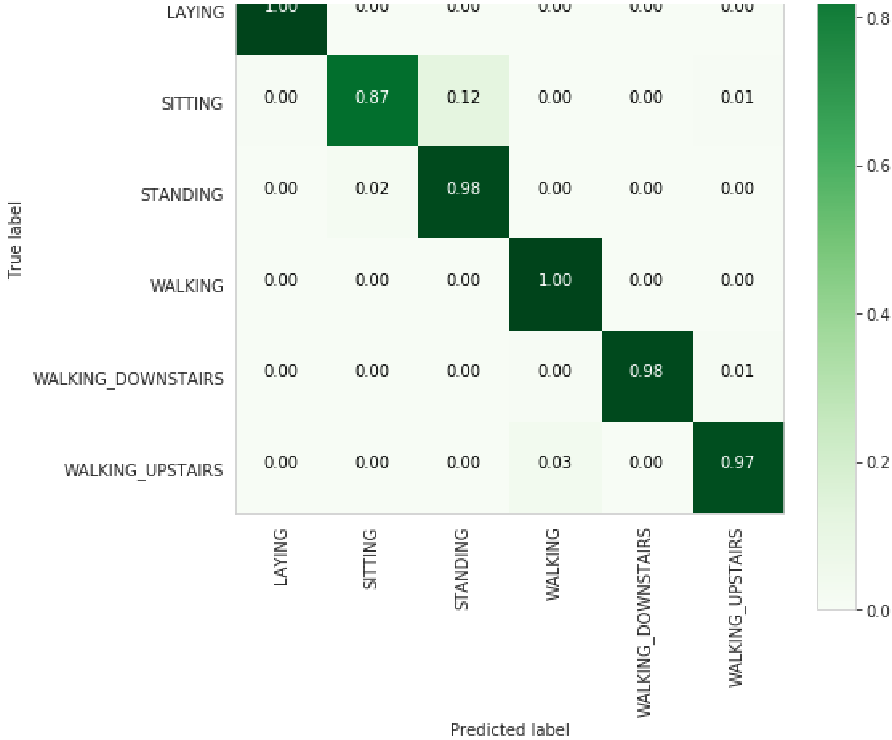
```
C:\Users\sanjeev\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:1978:
FutureWarning: The default value of cv will change from 3 to 5 in version 0.22. Specify it
explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
```

```
[Parallel(n_jobs=-1)]: Done  18 out of  18 | elapsed:   30.4s finished
C:\Users\sanjeev\Anaconda3\lib\site-packages\sklearn\svm\base.py:929: ConvergenceWarning:
Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
Done


training_time(HH:MM:SS.ms) - 0:00:37.065340


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.015620


---------------------
|      Accuracy      |
---------------------

    0.9657278588394977


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [   2 425  61   0   0   3]
 [   0  10 521   1   0   0]
 [   0   0   0 496   0   0]
 [   0   0   0   2 412   6]
 [   0   0   0  16   0 455]]
```






```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.98      0.87      0.92       491
          STANDING       0.90      0.98      0.94       532
           WALKING       0.96      1.00      0.98       496
WALKING_DOWNSTAIRS       1.00      0.98      0.99       420
  WALKING_UPSTAIRS       0.98      0.97      0.97       471

          accuracy                           0.97      2947
         macro avg       0.97      0.97      0.97      2947
      weighted avg       0.97      0.97      0.97      2947
```

In [38]:

```
print_grid_search_attributes(lr_svc_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
--------------------------

 LinearSVC(C=1, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=5e-05,
          verbose=0)

--------------------------
|     Best parameters     |
--------------------------
 Parameters of best estimator :

 {'C': 1}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------
```

```
Average Cross Validate scores of best estimator :

0.9464091403699674
```

3. Kernel SVM with GridSearch

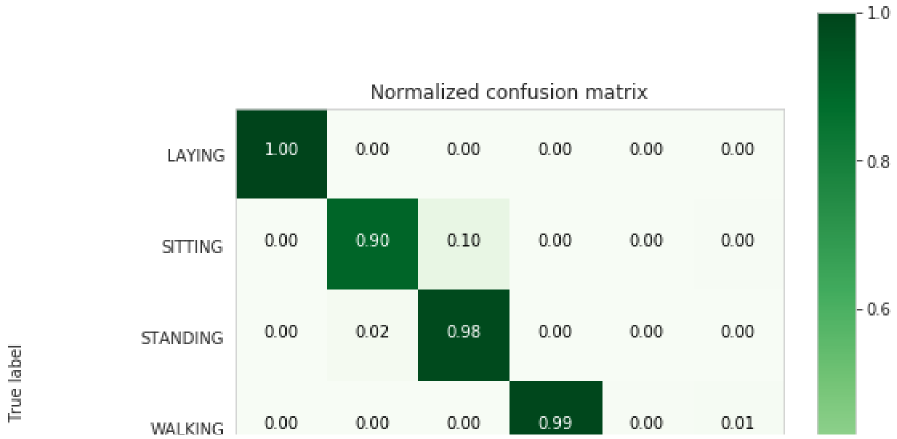
In [39]:

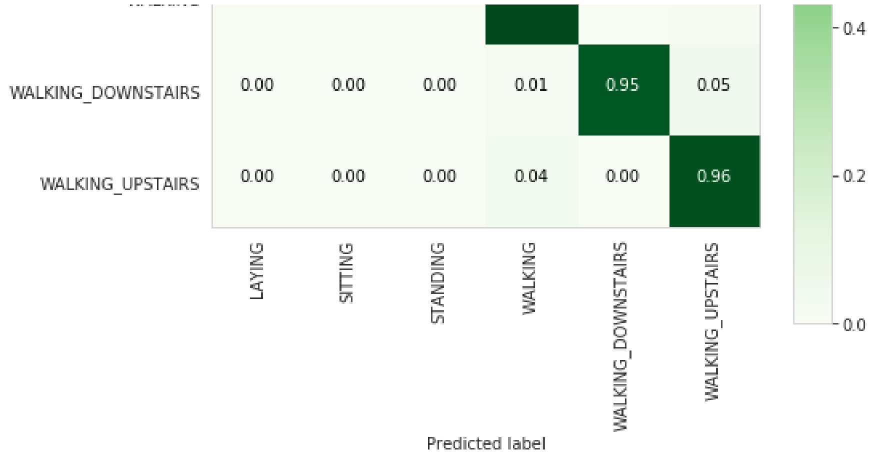
```python
from sklearn.svm import SVC
parameters = {'C':[2,8,16],\
              'gamma': [ 0.0078125, 0.125, 2]}
rbf_svm = SVC(kernel='rbf')
rbf_svm_grid = GridSearchCV(rbf_svm,param_grid=parameters, n_jobs=-1)
rbf_svm_grid_results = perform_model(rbf_svm_grid, X_train, y_train, X_test, y_test, class_labels=l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

```
training the model..
```

```
C:\Users\sanjeev\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:1978:
FutureWarning: The default value of cv will change from 3 to 5 in version 0.22. Specify it
explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

```
Done


training_time(HH:MM:SS.ms) - 0:03:52.367101


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:03.194097


---------------------
|      Accuracy      |
---------------------

    0.9626739056667798


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 441  48   0   0   2]
 [  0  12 520   0   0   0]
 [  0   0   0 489   2   5]
 [  0   0   0   4 397  19]
 [  0   0   0  17   1 453]]
```






```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.97      0.90      0.93       491
          STANDING       0.92      0.98      0.95       532
           WALKING       0.96      0.99      0.97       496
WALKING_DOWNSTAIRS       0.99      0.95      0.97       420
  WALKING_UPSTAIRS       0.95      0.96      0.95       471

          accuracy                           0.96      2947
         macro avg       0.96      0.96      0.96      2947
      weighted avg       0.96      0.96      0.96      2947
```

In [40]:

```
print_grid_search_attributes(rbf_svm_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
--------------------------

 SVC(C=16, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.0078125, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)

--------------------------
|     Best parameters     |
--------------------------
 Parameters of best estimator :

 {'C': 16, 'gamma': 0.0078125}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

 Average Cross Validate scores of best estimator :

 0.9440968443960827
```

4. Decision Trees with GridSearchCV

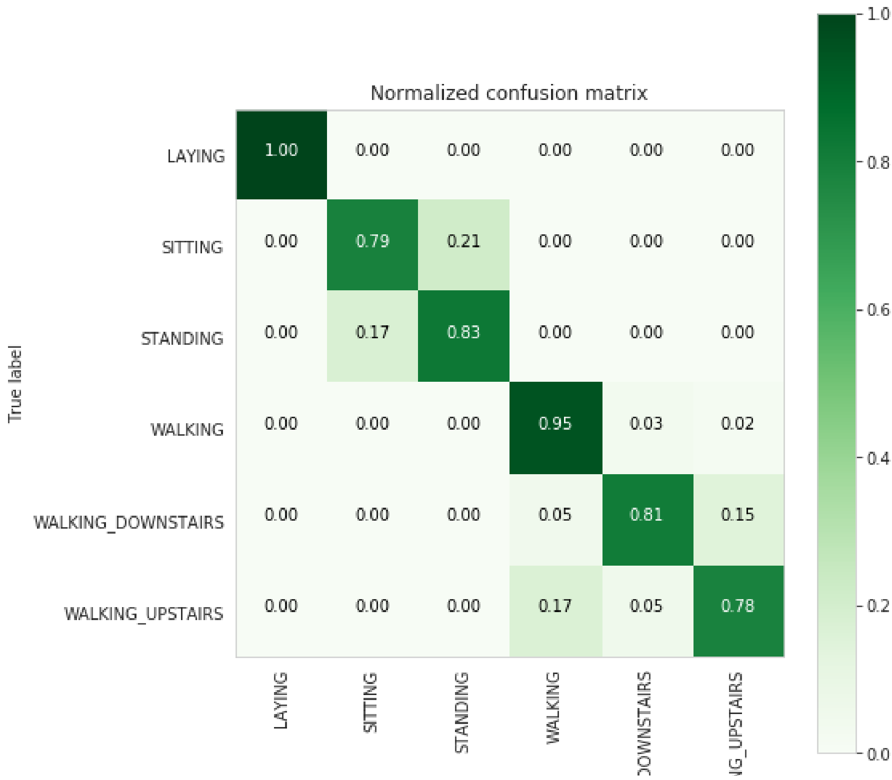
In [41]:

```
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 = perform_model(dt_grid, X_train, y_train, X_test, y_test, class_labels=labels)
```

```
training the model..
```

```
C:\Users\sanjeev\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:1978:
FutureWarning: The default value of cv will change from 3 to 5 in version 0.22. Specify it
explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

```
Done


training_time(HH:MM:SS.ms) - 0:00:11.826707


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00


---------------------
|      Accuracy      |
---------------------

    0.8625721072276892


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 386 105   0   0   0]
 [  0  93 439   0   0   0]
 [  0   0   0 471  17   8]
 [  0   0   0  19 340  61]
 [  0   0   0  78  24 369]]
```




```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.81      0.79      0.80       491
          STANDING       0.81      0.83      0.82       532
           WALKING       0.83      0.95      0.89       496
WALKING_DOWNSTAIRS       0.89      0.81      0.85       420
  WALKING_UPSTAIRS       0.84      0.78      0.81       471

          accuracy                           0.86      2947
         macro avg       0.86      0.86      0.86      2947
      weighted avg       0.86      0.86      0.86      2947
```

In [42]:

```
print_grid_search_attributes(dt_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
--------------------------

 DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=7,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')

--------------------------
|     Best parameters     |
--------------------------
 Parameters of best estimator :

 {'max_depth': 7}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

 Average Cross Validate scores of best estimator :

 0.8351468988030468
```

5. Random Forest Classifier with GridSearch

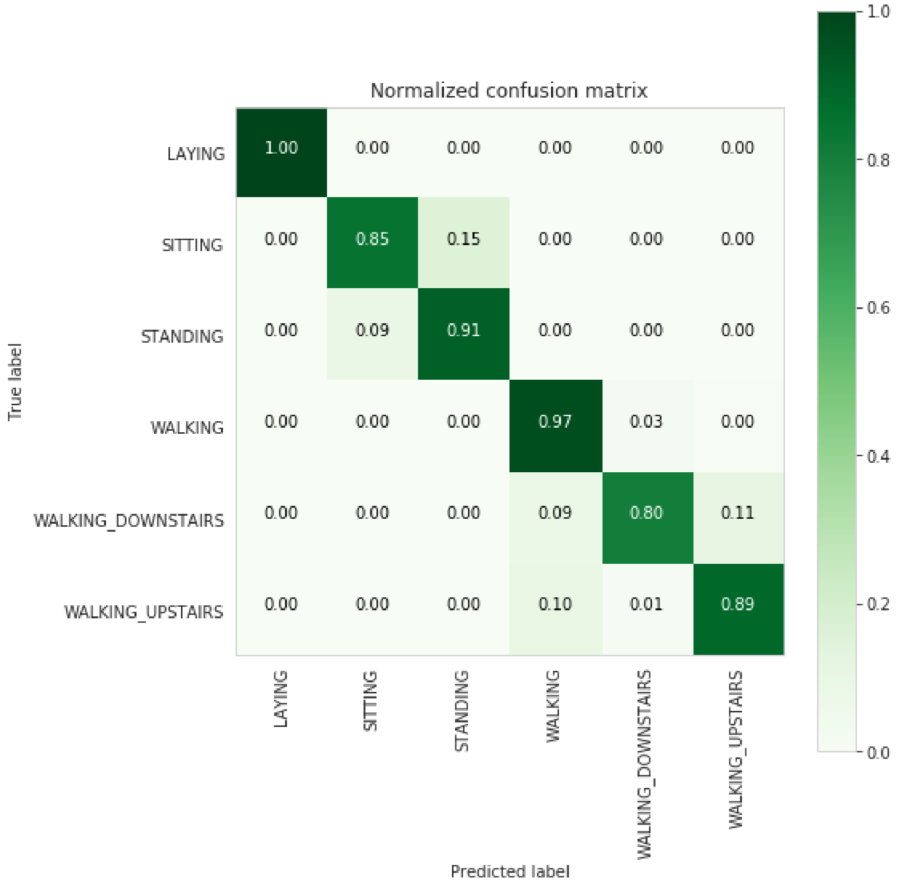
In [43]:

```
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 = perform_model(rfc_grid, X_train, y_train, X_test, y_test, class_labels=labels)
```

```
training the model..
```

```
C:\Users\sanjeev\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:1978:
FutureWarning: The default value of cv will change from 3 to 5 in version 0.22. Specify it
explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

```
Done


training_time(HH:MM:SS.ms) - 0:06:42.946082


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.075798


---------------------
|      Accuracy      |
---------------------

    0.9073634204275535


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 416  75   0   0   0]
 [  0  46 486   0   0   0]
 [  0   0   0 481  13   2]
 [  0   0   0  36 337  47]
 [  0   0   0  48   6 417]]
```




```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support
```

```
             LAYING       1.00      1.00      1.00       537
            SITTING       0.90      0.85      0.87       491
           STANDING       0.87      0.91      0.89       532
            WALKING       0.85      0.97      0.91       496
 WALKING_DOWNSTAIRS       0.95      0.80      0.87       420
   WALKING_UPSTAIRS       0.89      0.89      0.89       471

           accuracy                           0.91      2947
          macro avg       0.91      0.90      0.90      2947
       weighted avg       0.91      0.91      0.91      2947
```

In [44]:

```
print_grid_search_attributes(rfc_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
--------------------------

 RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=7, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=150,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)

--------------------------
|     Best parameters     |
--------------------------
 Parameters of best estimator : 

 {'max_depth': 7, 'n_estimators': 150}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

 Average Cross Validate scores of best estimator : 

 0.9156692056583242
```

6. Gradient Boosted Decision Trees With GridSearch

In [45]:

```
from sklearn.ensemble import GradientBoostingClassifier
param_grid = {'max_depth': np.arange(5,8,1), \
             'n_estimators':np.arange(130,170,10)}
gbdt = GradientBoostingClassifier()
gbdt_grid = GridSearchCV(gbdt, param_grid=param_grid, n_jobs=-1)
gbdt_grid_results = perform_model(gbdt_grid, X_train, y_train, X_test, y_test, class_labels=labels)
```

```
training the model..
```

```
C:\Users\sanjeev\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:1978:
FutureWarning: The default value of cv will change from 3 to 5 in version 0.22. Specify it
explicitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

```
Done
```

```
training_time(HH:MM:SS.ms) - 0:56:02.282017


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.109350


---------------------
|      Accuracy      |
---------------------

    0.9239904988123515


--------------------
| Confusion Matrix |
--------------------

 [[537   0   0   0   0   0]
 [  0 402  87   0   0   2]
 [  0  37 495   0   0   0]
 [  0   0   0 481   7   8]
 [  0   0   0   9 371  40]
 [  0   1   0  28   5 437]]
```


```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.91      0.82      0.86       491
          STANDING       0.85      0.93      0.89       532
           WALKING       0.93      0.97      0.95       496
WALKING_DOWNSTAIRS       0.97      0.88      0.92       420
  WALKING_UPSTAIRS       0.90      0.93      0.91       471

          accuracy                           0.92      2947
```

```
   macro avg       0.93      0.92      0.92      2947
weighted avg       0.93      0.92      0.92      2947
```

In [46]:

```
print_grid_search_attributes(gbdt_grid_results['model'])
```

```
--------------------------
|      Best Estimator     |
--------------------------

 GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=5,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=150,
                           n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)

--------------------------
|     Best parameters     |
--------------------------
 Parameters of best estimator :

 {'max_depth': 5, 'n_estimators': 150}

---------------------------------
|   No of CrossValidation sets   |
--------------------------------

 Total numbre of cross validation sets: 3

--------------------------
|        Best Score       |
--------------------------

 Average Cross Validate scores of best estimator :

 0.9045157780195865
```

LSTM

In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

```
# 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'])
```

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).to_numpy()
        )

    # 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).to_numpy()
```

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 [7]:

```
# Importing tensorflow
np.random.seed(42)
import tensorflow as tf
```

```
import tensorflow as tf
tf.set_random_seed(42)
```

In [8]:

```
# Configuring a session
session_conf = tf.ConfigProto(
    intra_op_parallelism_threads=1,
    inter_op_parallelism_threads=1
)
```

In [9]:

```
# Import Keras
from keras import backend as K
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```

```
Using TensorFlow backend.
```

In [10]:

```
# Importing libraries
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout
```

In [11]:

```
# Initializing parameters
epochs = 50
batch_size = 32
n_hidden = 32
```

In [12]:

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

In [13]:

```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

In [14]:

```
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
```

Model 1

LSTM - Dropout(0.5) - Dense

In [20]:

```python
# 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_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 32)                5376
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 198
=================================================================
Total params: 5,574
Trainable params: 5,574
Non-trainable params: 0
_________________________________________________________________
```

In [21]:

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

In [22]:

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

```
WARNING:tensorflow:From C:\Users\sanjeev\Anaconda3\lib\site-
packages\tensorflow\python\ops\math_grad.py:1250: add_dispatch_support.<locals>.wrapper (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From C:\Users\sanjeev\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables is deprecated. Plea
se use tf.compat.v1.global_variables instead.

Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 28s 4ms/step - loss: 1.3008 - accuracy: 0.4645 - val_
loss: 1.0891 - val_accuracy: 0.5565
Epoch 2/30
7352/7352 [==============================] - 26s 4ms/step - loss: 0.9056 - accuracy: 0.6171 - val_
loss: 0.8277 - val_accuracy: 0.5942
Epoch 3/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.7434 - accuracy: 0.6549 - val_
loss: 0.7569 - val_accuracy: 0.6230
Epoch 4/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.6725 - accuracy: 0.6800 - val_
loss: 0.6941 - val_accuracy: 0.6651
Epoch 5/30
7352/7352 [==============================] - 26s 4ms/step - loss: 0.6236 - accuracy: 0.7116 - val_
loss: 0.6568 - val_accuracy: 0.7326
Epoch 6/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.5866 - accuracy: 0.7333 - val_
loss: 0.7696 - val_accuracy: 0.6763
Epoch 7/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.5055 - accuracy: 0.7748 - val_
loss: 0.6162 - val_accuracy: 0.7272
Epoch 8/30
7352/7352 [==============================] - 26s 4ms/step - loss: 0.4769 - accuracy: 0.7791 - val_
```

```
loss: 0.5323 - val_accuracy: 0.7465
Epoch 9/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.4243 - accuracy: 0.7979 - val_
loss: 0.6893 - val_accuracy: 0.7167
Epoch 10/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.4033 - accuracy: 0.8096 - val_
loss: 0.5631 - val_accuracy: 0.7326
Epoch 11/30
7352/7352 [==============================] - 26s 4ms/step - loss: 0.3785 - accuracy: 0.8391 - val_
loss: 0.5226 - val_accuracy: 0.7937
Epoch 12/30
7352/7352 [==============================] - 26s 4ms/step - loss: 0.3327 - accuracy: 0.8791 - val_
loss: 0.5721 - val_accuracy: 0.8694
Epoch 13/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.2857 - accuracy: 0.9132 - val_
loss: 0.4536 - val_accuracy: 0.8812
Epoch 14/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.2537 - accuracy: 0.9207 - val_
loss: 0.5414 - val_accuracy: 0.8748
Epoch 15/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.2421 - accuracy: 0.9293 - val_
loss: 0.4205 - val_accuracy: 0.8968
Epoch 16/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.2077 - accuracy: 0.9331 - val_
loss: 0.4868 - val_accuracy: 0.8785
Epoch 17/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1950 - accuracy: 0.9384 - val_
loss: 0.5625 - val_accuracy: 0.8833
Epoch 18/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1844 - accuracy: 0.9419 - val_
loss: 0.6079 - val_accuracy: 0.8738
Epoch 19/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1846 - accuracy: 0.9414 - val_
loss: 0.4497 - val_accuracy: 0.8999
Epoch 20/30
7352/7352 [==============================] - 29s 4ms/step - loss: 0.1701 - accuracy: 0.9459 - val_
loss: 0.5215 - val_accuracy: 0.8795
Epoch 21/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1723 - accuracy: 0.9450 - val_
loss: 0.4698 - val_accuracy: 0.8887
Epoch 22/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1813 - accuracy: 0.9448 - val_
loss: 0.4783 - val_accuracy: 0.8795
Epoch 23/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1579 - accuracy: 0.9465 - val_
loss: 0.4126 - val_accuracy: 0.8968
Epoch 24/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1754 - accuracy: 0.9448 - val_
loss: 0.3679 - val_accuracy: 0.9111
Epoch 25/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1827 - accuracy: 0.9425 - val_
loss: 0.9810 - val_accuracy: 0.8426
Epoch 26/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1565 - accuracy: 0.9489 - val_
loss: 0.3639 - val_accuracy: 0.9043
Epoch 27/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1508 - accuracy: 0.9484 - val_
loss: 0.3858 - val_accuracy: 0.8996
Epoch 28/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1461 - accuracy: 0.9470 - val_
loss: 0.4374 - val_accuracy: 0.9013
Epoch 29/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1558 - accuracy: 0.9449 - val_
loss: 0.3596 - val_accuracy: 0.9043
Epoch 30/30
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1712 - accuracy: 0.9433 - val_
loss: 0.4166 - val_accuracy: 0.8955
```

Out[22]:

```
<keras.callbacks.callbacks.History at 0x2e063805ba8>
```

In [23]:

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

```
print(confusion_matrix(Y_test, model.predict(X_test)))
```

```
Pred                  LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                   510        0        27        0                   0
SITTING                    2      381       105        1                   1
STANDING                   0       86       446        0                   0
WALKING                    0        0         0      454                  15
WALKING_DOWNSTAIRS         0        0         0        0                 419
WALKING_UPSTAIRS           0        7         0        4                  31

Pred                  WALKING_UPSTAIRS
True
LAYING                               0
SITTING                              1
STANDING                             0
WALKING                             27
WALKING_DOWNSTAIRS                   1
WALKING_UPSTAIRS                   429
```

In [24]:

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

```
2947/2947 [==============================] - 1s 463us/step
```

In [25]:

```
score
```

Out[25]:

```
[0.41655886096877154, 0.8954869508743286]
```

Model 2

LSTM(32) - Dropout - LSTM(64) - Dropout - Dense

In [15]:

```
from keras.layers.normalization import BatchNormalization
```

In [37]:

```
# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(n_hidden,return_sequences=True,input_shape=(timesteps, input_dim)))
# Adding batch normalization layer
model.add(BatchNormalization())
# Adding a dropout layer
model.add(Dropout(0.5))
# Adding LSTM layer
model.add(LSTM(64))
# Adding batch normalization layer
model.add(BatchNormalization())
# 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_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_11 (LSTM)               (None, 128, 32)           5376
_________________________________________________________________
batch_normalization_7 (Batch (None, 128, 32)           128
```

```
batch_normalization_7 (Batch (None, 128, 32)          128
_________________________________________________________________
dropout_8 (Dropout)          (None, 128, 32)          0
_________________________________________________________________
lstm_12 (LSTM)               (None, 64)               24832
_________________________________________________________________
batch_normalization_8 (Batch (None, 64)               256
_________________________________________________________________
dropout_9 (Dropout)          (None, 64)               0
_________________________________________________________________
dense_4 (Dense)              (None, 6)                390
=================================================================
Total params: 30,982
Trainable params: 30,790
Non-trainable params: 192
_________________________________________________________________
```

In [38]:

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

In [39]:

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

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 67s 9ms/step - loss: 0.9396 - accuracy: 0.6396 - val_
loss: 0.9047 - val_accuracy: 0.6040
Epoch 2/30
7352/7352 [==============================] - 62s 8ms/step - loss: 0.6574 - accuracy: 0.7375 - val_
loss: 0.6003 - val_accuracy: 0.7791
Epoch 3/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.4637 - accuracy: 0.8509 - val_
loss: 0.4141 - val_accuracy: 0.8785
Epoch 4/30
7352/7352 [==============================] - 63s 9ms/step - loss: 0.2972 - accuracy: 0.9063 - val_
loss: 0.3885 - val_accuracy: 0.8996
Epoch 5/30
7352/7352 [==============================] - 63s 9ms/step - loss: 0.2370 - accuracy: 0.9261 - val_
loss: 0.4202 - val_accuracy: 0.8890
Epoch 6/30
7352/7352 [==============================] - 63s 9ms/step - loss: 0.2122 - accuracy: 0.9314 - val_
loss: 0.4281 - val_accuracy: 0.8992
Epoch 7/30
7352/7352 [==============================] - 62s 8ms/step - loss: 0.1949 - accuracy: 0.9348 - val_
loss: 0.4892 - val_accuracy: 0.8924
Epoch 8/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.1911 - accuracy: 0.9343 - val_
loss: 0.3084 - val_accuracy: 0.9175
Epoch 9/30
7352/7352 [==============================] - 67s 9ms/step - loss: 0.1771 - accuracy: 0.9327 - val_
loss: 0.4832 - val_accuracy: 0.9070
Epoch 10/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.1737 - accuracy: 0.9388 - val_
loss: 0.3536 - val_accuracy: 0.9162
Epoch 11/30
7352/7352 [==============================] - 67s 9ms/step - loss: 0.1637 - accuracy: 0.9373 - val_
loss: 0.3456 - val_accuracy: 0.9057
Epoch 12/30
7352/7352 [==============================] - 66s 9ms/step - loss: 0.1620 - accuracy: 0.9441 - val_
loss: 0.2723 - val_accuracy: 0.9233
Epoch 13/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.1568 - accuracy: 0.9403 - val_
loss: 0.4087 - val_accuracy: 0.9141
Epoch 14/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.1666 - accuracy: 0.9445 - val_
```

```
loss: 0.4055 - val_accuracy: 0.9158
Epoch 15/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.1613 - accuracy: 0.9429 - val_
loss: 0.3386 - val_accuracy: 0.9148
Epoch 16/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.1642 - accuracy: 0.9366 - val_
loss: 0.4195 - val_accuracy: 0.9206
Epoch 17/30
7352/7352 [==============================] - 64s 9ms/step - loss: 0.1619 - accuracy: 0.9438 - val_
loss: 0.4564 - val_accuracy: 0.9152
Epoch 18/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.1541 - accuracy: 0.9423 - val_
loss: 0.3679 - val_accuracy: 0.9237
Epoch 19/30
7352/7352 [==============================] - 63s 9ms/step - loss: 0.1552 - accuracy: 0.9425 - val_
loss: 0.4656 - val_accuracy: 0.9074
Epoch 20/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.1529 - accuracy: 0.9427 - val_
loss: 0.3913 - val_accuracy: 0.8334
Epoch 21/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.1661 - accuracy: 0.9382 - val_
loss: 0.5133 - val_accuracy: 0.9070
Epoch 22/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.1477 - accuracy: 0.9437 - val_
loss: 0.5740 - val_accuracy: 0.9097
Epoch 23/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1701 - accuracy: 0.9442 - val_
loss: 0.4876 - val_accuracy: 0.9135
Epoch 24/30
7352/7352 [==============================] - 63s 9ms/step - loss: 0.1564 - accuracy: 0.9475 - val_
loss: 0.3887 - val_accuracy: 0.9230
Epoch 25/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.1593 - accuracy: 0.9448 - val_
loss: 0.4500 - val_accuracy: 0.9108
Epoch 26/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1345 - accuracy: 0.9487 - val_
loss: 0.3703 - val_accuracy: 0.9158
Epoch 27/30
7352/7352 [==============================] - 67s 9ms/step - loss: 0.1405 - accuracy: 0.9494 - val_
loss: 0.4475 - val_accuracy: 0.9162
Epoch 28/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.1462 - accuracy: 0.9463 - val_
loss: 0.6044 - val_accuracy: 0.9030
Epoch 29/30
7352/7352 [==============================] - 65s 9ms/step - loss: 0.1440 - accuracy: 0.9455 - val_
loss: 0.5156 - val_accuracy: 0.9094
Epoch 30/30
7352/7352 [==============================] - 67s 9ms/step - loss: 0.1531 - accuracy: 0.9478 - val_
loss: 0.4488 - val_accuracy: 0.9101
```

Out[39]:

```
<keras.callbacks.callbacks.History at 0x2e06a8afc50>
```

In [40]:

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

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 537        0         0        0                   0
SITTING                  4      376       111        0                   0
STANDING                 0       89       443        0                   0
WALKING                  0        1         0      469                  10
WALKING_DOWNSTAIRS       0        0         0        4                 408
WALKING_UPSTAIRS         0        2         0        0                  20

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            0
STANDING                           0
WALKING                           16
WALKING_DOWNSTAIRS                 8
```

```
WALKING_UPSTAIRS                  449
```

In [41]:

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

```
2947/2947 [==============================] - 3s 1ms/step
```

In [42]:

```
score
```

Out[42]:

```
[0.44884221508369054, 0.9100780487060547]
```

Model 3

LSTM(32) - BatchNorm - Dropout(0.5) - LSTM(100) - BatchNorm - Dropout(0.5) - LSTM(200) - BatchNorm - Dropout(0.5) - LSTM(300) - BatchNorm - Dropout(0.5) - Dense

In [43]:

```
# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(n_hidden,return_sequences=True,input_shape=(timesteps, input_dim)))
# Adding batch normalization layer
model.add(BatchNormalization())
# Adding a dropout layer
model.add(Dropout(0.5))
# Adding LSTM layer
model.add(LSTM(100,return_sequences=True))
# Adding batch normalization layer
model.add(BatchNormalization())
# Adding a dropout layer
model.add(Dropout(0.5))
model.add(LSTM(200,return_sequences=True))
# Adding batch normalization layer
model.add(BatchNormalization())
# Adding a dropout layer
model.add(Dropout(0.5))
model.add(LSTM(300))
# Adding batch normalization layer
model.add(BatchNormalization())
# 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_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_13 (LSTM)               (None, 128, 32)           5376
_________________________________________________________________
batch_normalization_9 (Batch (None, 128, 32)           128
_________________________________________________________________
dropout_10 (Dropout)         (None, 128, 32)           0
_________________________________________________________________
lstm_14 (LSTM)               (None, 128, 100)          53200
_________________________________________________________________
batch_normalization_10 (Batc (None, 128, 100)          400
_________________________________________________________________
dropout_11 (Dropout)         (None, 128, 100)          0
_________________________________________________________________
lstm_15 (LSTM)               (None, 128, 200)          240800
_________________________________________________________________
```

```
batch_normalization_11 (Batc (None, 128, 200)          800
_________________________________________________________________
dropout_12 (Dropout)         (None, 128, 200)          0
_________________________________________________________________
lstm_16 (LSTM)               (None, 300)               601200
_________________________________________________________________
batch_normalization_12 (Batc (None, 300)               1200
_________________________________________________________________
dropout_13 (Dropout)         (None, 300)               0
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 1806
=================================================================
Total params: 904,910
Trainable params: 903,646
Non-trainable params: 1,264
_________________________________________________________________
```

In [44]:

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

In [45]:

```
history = model.fit(X_train,Y_train,batch_size=batch_size,validation_data=(X_test, Y_test),epochs=e
pochs)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 515s 70ms/step - loss: 0.8332 - accuracy: 0.6087 - va
l_loss: 0.7015 - val_accuracy: 0.6098
Epoch 2/30
7352/7352 [==============================] - 498s 68ms/step - loss: 0.7708 - accuracy: 0.6113 - va
l_loss: 0.7959 - val_accuracy: 0.6281
Epoch 3/30
7352/7352 [==============================] - 489s 67ms/step - loss: 0.7218 - accuracy: 0.6308 - va
l_loss: 0.7227 - val_accuracy: 0.6288
Epoch 4/30
7352/7352 [==============================] - 511s 70ms/step - loss: 0.7135 - accuracy: 0.6319 - va
l_loss: 0.7020 - val_accuracy: 0.6386
Epoch 5/30
7352/7352 [==============================] - 513s 70ms/step - loss: 0.7044 - accuracy: 0.6332 - va
l_loss: 0.7842 - val_accuracy: 0.6135
Epoch 6/30
7352/7352 [==============================] - 509s 69ms/step - loss: 0.7032 - accuracy: 0.6245 - va
l_loss: 0.9970 - val_accuracy: 0.5948
Epoch 7/30
7352/7352 [==============================] - 505s 69ms/step - loss: 0.7214 - accuracy: 0.6477 - va
l_loss: 0.7416 - val_accuracy: 0.6335
Epoch 8/30
7352/7352 [==============================] - 502s 68ms/step - loss: 0.6798 - accuracy: 0.6385 - va
l_loss: 0.8528 - val_accuracy: 0.6186
Epoch 9/30
7352/7352 [==============================] - 496s 68ms/step - loss: 0.6841 - accuracy: 0.6411 - va
l_loss: 0.7832 - val_accuracy: 0.6417
Epoch 10/30
7352/7352 [==============================] - 495s 67ms/step - loss: 0.6746 - accuracy: 0.6382 - va
l_loss: 0.7355 - val_accuracy: 0.5955
Epoch 11/30
7352/7352 [==============================] - 500s 68ms/step - loss: 0.7036 - accuracy: 0.6310 - va
l_loss: 0.6798 - val_accuracy: 0.6121
Epoch 12/30
7352/7352 [==============================] - 519s 71ms/step - loss: 0.6016 - accuracy: 0.6672 - va
l_loss: 0.7377 - val_accuracy: 0.5843
Epoch 13/30
7352/7352 [==============================] - 507s 69ms/step - loss: 0.4163 - accuracy: 0.8618 - va
l_loss: 0.4842 - val_accuracy: 0.8476
Epoch 14/30
7352/7352 [==============================] - 539s 73ms/step - loss: 0.2754 - accuracy: 0.9048 - va
l_loss: 0.4826 - val_accuracy: 0.8829
Epoch 15/30
7352/7352 [==============================] - 556s 76ms/step - loss: 0.2322 - accuracy: 0.9163 - va
l_loss: 0.3981 - val_accuracy: 0.8856
```

```
l_loss: 0.3981 - val_accuracy: 0.8856
Epoch 16/30
7352/7352 [==============================] - 6859s 933ms/step - loss: 0.2131 - accuracy: 0.9267 -
val_loss: 0.3468 - val_accuracy: 0.9067
Epoch 17/30
7352/7352 [==============================] - 520s 71ms/step - loss: 0.2049 - accuracy: 0.9252 - va
l_loss: 0.2879 - val_accuracy: 0.9203
Epoch 18/30
7352/7352 [==============================] - 503s 68ms/step - loss: 0.1901 - accuracy: 0.9279 - va
l_loss: 0.3502 - val_accuracy: 0.9175
Epoch 19/30
7352/7352 [==============================] - 498s 68ms/step - loss: 0.1929 - accuracy: 0.9323 - va
l_loss: 0.2692 - val_accuracy: 0.9186
Epoch 20/30
7352/7352 [==============================] - 494s 67ms/step - loss: 0.1921 - accuracy: 0.9301 - va
l_loss: 0.2513 - val_accuracy: 0.9182
Epoch 21/30
7352/7352 [==============================] - 496s 67ms/step - loss: 0.1805 - accuracy: 0.9340 - va
l_loss: 0.2996 - val_accuracy: 0.9206
Epoch 22/30
7352/7352 [==============================] - 495s 67ms/step - loss: 0.1884 - accuracy: 0.9385 - va
l_loss: 0.4036 - val_accuracy: 0.9169
Epoch 23/30
7352/7352 [==============================] - 497s 68ms/step - loss: 0.1819 - accuracy: 0.9312 - va
l_loss: 0.3592 - val_accuracy: 0.9074
Epoch 24/30
7352/7352 [==============================] - 494s 67ms/step - loss: 0.2005 - accuracy: 0.9285 - va
l_loss: 0.3478 - val_accuracy: 0.9063
Epoch 25/30
7352/7352 [==============================] - 494s 67ms/step - loss: 0.2068 - accuracy: 0.9310 - va
l_loss: 0.2414 - val_accuracy: 0.9162
Epoch 26/30
7352/7352 [==============================] - 494s 67ms/step - loss: 0.1905 - accuracy: 0.9308 - va
l_loss: 0.3426 - val_accuracy: 0.9091
Epoch 27/30
7352/7352 [==============================] - 494s 67ms/step - loss: 0.1835 - accuracy: 0.9374 - va
l_loss: 0.3108 - val_accuracy: 0.9145
Epoch 28/30
7352/7352 [==============================] - 495s 67ms/step - loss: 0.1781 - accuracy: 0.9366 - va
l_loss: 0.3854 - val_accuracy: 0.9121
Epoch 29/30
7352/7352 [==============================] - 510s 69ms/step - loss: 0.1730 - accuracy: 0.9388 - va
l_loss: 0.2765 - val_accuracy: 0.9250
Epoch 30/30
7352/7352 [==============================] - 497s 68ms/step - loss: 0.1574 - accuracy: 0.9354 - va
l_loss: 0.3217 - val_accuracy: 0.9172
```

In [46]:

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

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 537        0         0        0                   0
SITTING                  5      380       106        0                   0
STANDING                 0       93       439        0                   0
WALKING                  0        0         0      456                  23
WALKING_DOWNSTAIRS       0        0         0        0                 420
WALKING_UPSTAIRS         0        0         0        0                   0

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            0
STANDING                           0
WALKING                           17
WALKING_DOWNSTAIRS                 0
WALKING_UPSTAIRS                 471
```

In [47]:

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

```
2947/2947 [==============================] - 25s 9ms/step
```

In [48]:

```
score
```

Out[48]:

```
[0.32174529994839807, 0.917203962802887]
```

Model 4

LSTM

In [16]:

```
from keras.callbacks import EarlyStopping, ModelCheckpoint
```

In [17]:

```
# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(n_hidden,return_sequences=True,input_shape=(timesteps, input_dim)))
# Adding batch normalization layer
model.add(BatchNormalization())
# Adding a dropout layer
model.add(Dropout(0.5))
# Adding LSTM layer
model.add(LSTM(64,return_sequences=True))
# Adding batch normalization layer
model.add(BatchNormalization())
# Adding a dropout layer
model.add(Dropout(0.5))
# Adding LSTM layer
model.add(LSTM(128,return_sequences=True))
# Adding batch normalization layer
model.add(BatchNormalization())
# Adding a dropout layer
model.add(Dropout(0.5))
# Adding LSTM layer
model.add(LSTM(256,return_sequences=True))
# Adding batch normalization layer
model.add(BatchNormalization())
# Adding a dropout layer
model.add(Dropout(0.5))
# Adding LSTM layer
model.add(LSTM(512))
# Adding batch normalization layer
model.add(BatchNormalization())
# 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_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_1 (LSTM)                (None, 128, 32)           5376      
_________________________________________________________________
batch_normalization_1 (Batch (None, 128, 32)           128       
_________________________________________________________________
dropout_1 (Dropout)          (None, 128, 32)           0         
_________________________________________________________________
lstm_2 (LSTM)                (None, 128, 64)           24832     
_________________________________________________________________
batch_normalization_2 (Batch (None, 128, 64)           256       
_________________________________________________________________
```

```
dropout_2 (Dropout)          (None, 128, 64)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 128, 128)          98816
_________________________________________________________________
batch_normalization_3 (Batch (None, 128, 128)          512
_________________________________________________________________
dropout_3 (Dropout)          (None, 128, 128)          0
_________________________________________________________________
lstm_4 (LSTM)                (None, 128, 256)          394240
_________________________________________________________________
batch_normalization_4 (Batch (None, 128, 256)          1024
_________________________________________________________________
dropout_4 (Dropout)          (None, 128, 256)          0
_________________________________________________________________
lstm_5 (LSTM)                (None, 512)               1574912
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 3078
=================================================================
Total params: 2,105,222
Trainable params: 2,103,238
Non-trainable params: 1,984
_________________________________________________________________
```

In [18]:

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

In [19]:

```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10, min_delta=0.0001,restore_best_weights=True)
mc = ModelCheckpoint('best_model.hdf5', monitor='val_acc', verbose=1, save_best_only=True, mode='max')
```

In [20]:

```
history = model.fit(X_train,Y_train,batch_size=batch_size,validation_data=(X_test, Y_test),epochs=epochs,callbacks=[es,mc])
```

```
WARNING:tensorflow:From C:\Users\sanjeev\Anaconda3\lib\site-packages\tensorflow\python\ops\math_grad.py:1250: add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From C:\Users\sanjeev\Anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

Train on 7352 samples, validate on 2947 samples
Epoch 1/50
7352/7352 [==============================] - 902s 123ms/step - loss: 0.8267 - accuracy: 0.6431 - val_loss: 1.7162 - val_accuracy: 0.4744
Epoch 2/50
```

```
C:\Users\sanjeev\Anaconda3\lib\site-packages\keras\callbacks\callbacks.py:707: RuntimeWarning: Can save best model only with val_acc available, skipping.
  'skipping.' % (self.monitor), RuntimeWarning)
```

```
7352/7352 [==============================] - 887s 121ms/step - loss: 0.7458 - accuracy: 0.6745 - val_loss: 1.1356 - val_accuracy: 0.5823
Epoch 3/50
7352/7352 [==============================] - 1005s 137ms/step - loss: 0.7271 - accuracy: 0.7232 - val_loss: 0.7087 - val_accuracy: 0.7163
Epoch 4/50
7352/7352 [==============================] - 12351s 2s/step - loss: 0.5814 - accuracy: 0.7556 - val_loss: 0.8868 - val_accuracy: 0.6824
Epoch 5/50
```

```
7352/7352 [==============================] - 1056s 144ms/step - loss: 0.4730 - accuracy: 0.7780 -
val_loss: 0.5698 - val_accuracy: 0.7248
Epoch 6/50
7352/7352 [==============================] - 1036s 141ms/step - loss: 0.4586 - accuracy: 0.7606 -
val_loss: 0.9700 - val_accuracy: 0.6576
Epoch 7/50
7352/7352 [==============================] - 1029s 140ms/step - loss: 0.4553 - accuracy: 0.7722 -
val_loss: 0.4239 - val_accuracy: 0.7560
Epoch 8/50
7352/7352 [==============================] - 1043s 142ms/step - loss: 0.4074 - accuracy: 0.7763 -
val_loss: 0.4408 - val_accuracy: 0.7523
Epoch 9/50
7352/7352 [==============================] - 1046s 142ms/step - loss: 0.4023 - accuracy: 0.7756 -
val_loss: 0.6162 - val_accuracy: 0.6932
Epoch 10/50
7352/7352 [==============================] - 1040s 141ms/step - loss: 0.4209 - accuracy: 0.7666 -
val_loss: 0.5086 - val_accuracy: 0.7706
Epoch 11/50
7352/7352 [==============================] - 1058s 144ms/step - loss: 0.4112 - accuracy: 0.7786 -
val_loss: 0.4923 - val_accuracy: 0.7621
Epoch 12/50
7352/7352 [==============================] - 1020s 139ms/step - loss: 0.3832 - accuracy: 0.7867 -
val_loss: 0.5143 - val_accuracy: 0.7452
Epoch 13/50
7352/7352 [==============================] - 1010s 137ms/step - loss: 0.3892 - accuracy: 0.7760 -
val_loss: 0.5402 - val_accuracy: 0.7727
Epoch 14/50
7352/7352 [==============================] - 1007s 137ms/step - loss: 0.3729 - accuracy: 0.7931 -
val_loss: 0.5458 - val_accuracy: 0.7421
Epoch 15/50
7352/7352 [==============================] - 1007s 137ms/step - loss: 0.3883 - accuracy: 0.7905 -
val_loss: 0.4959 - val_accuracy: 0.7615
Epoch 16/50
7352/7352 [==============================] - 1012s 138ms/step - loss: 0.3866 - accuracy: 0.7899 -
val_loss: 0.5622 - val_accuracy: 0.6892
Epoch 17/50
7352/7352 [==============================] - 1007s 137ms/step - loss: 0.3369 - accuracy: 0.8694 -
val_loss: 0.4182 - val_accuracy: 0.8697
Epoch 18/50
7352/7352 [==============================] - 1011s 138ms/step - loss: 0.2468 - accuracy: 0.9161 -
val_loss: 0.2906 - val_accuracy: 0.9128
Epoch 19/50
7352/7352 [==============================] - 1020s 139ms/step - loss: 0.1990 - accuracy: 0.9234 -
val_loss: 0.2919 - val_accuracy: 0.9162
Epoch 20/50
7352/7352 [==============================] - 1029s 140ms/step - loss: 0.1864 - accuracy: 0.9278 -
val_loss: 0.3058 - val_accuracy: 0.9152
Epoch 21/50
7352/7352 [==============================] - 1038s 141ms/step - loss: 0.1770 - accuracy: 0.9339 -
val_loss: 0.2002 - val_accuracy: 0.9403
Epoch 22/50
7352/7352 [==============================] - 1017s 138ms/step - loss: 0.1723 - accuracy: 0.9362 -
val_loss: 0.2493 - val_accuracy: 0.9348
Epoch 23/50
7352/7352 [==============================] - 1015s 138ms/step - loss: 0.1633 - accuracy: 0.9285 -
val_loss: 0.4483 - val_accuracy: 0.8890
Epoch 24/50
7352/7352 [==============================] - 1055s 144ms/step - loss: 0.1517 - accuracy: 0.9397 -
val_loss: 0.4098 - val_accuracy: 0.9036
Epoch 25/50
7352/7352 [==============================] - 1224s 166ms/step - loss: 0.1499 - accuracy: 0.9381 -
val_loss: 0.2804 - val_accuracy: 0.9104
Epoch 26/50
7352/7352 [==============================] - 1052s 143ms/step - loss: 0.1937 - accuracy: 0.9348 -
val_loss: 0.2311 - val_accuracy: 0.9237
Epoch 27/50
7352/7352 [==============================] - 1043s 142ms/step - loss: 0.1779 - accuracy: 0.9313 -
val_loss: 0.3720 - val_accuracy: 0.8690
Epoch 28/50
7352/7352 [==============================] - 1033s 140ms/step - loss: 0.1602 - accuracy: 0.9372 -
val_loss: 0.3132 - val_accuracy: 0.9121
Epoch 29/50
7352/7352 [==============================] - 1033s 141ms/step - loss: 0.1646 - accuracy: 0.9388 -
val_loss: 0.2218 - val_accuracy: 0.9203
Epoch 30/50
7352/7352 [==============================] - 1023s 139ms/step - loss: 0.1479 - accuracy: 0.9402 -
val_loss: 0.2908 - val_accuracy: 0.8989
```

```
Epoch 31/50
7352/7352 [==============================] - 25171s 3s/step - loss: 0.1524 - accuracy: 0.9389 - va
l_loss: 0.3713 - val_accuracy: 0.9026
Restoring model weights from the end of the best epoch
Epoch 00031: early stopping
```

In [21]:

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

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 537        0         0        0                   0
SITTING                 19      412        59        0                   1
STANDING                 0       72       460        0                   0
WALKING                  1        0         0      492                   0
WALKING_DOWNSTAIRS       0        0         0        0                 413
WALKING_UPSTAIRS         0        0         0        4                  10

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            0
STANDING                           0
WALKING                            3
WALKING_DOWNSTAIRS                 7
WALKING_UPSTAIRS                 457
```

In [22]:

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

```
2947/2947 [==============================] - 56s 19ms/step
```

In [23]:

```
score
```

Out[23]:

```
[0.20017623385080346, 0.9402782320976257]
```

In [ ]:

```

```

Results

In [66]:

```
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ["Model","Hyperparameter Tuing","Accuracy"]
table.add_row(["Logistic Regression","GridSearchCV",94.5])
table.add_row(["Linear SVC","GridSearchCV",94.6])
table.add_row(["Kernal SVM","GridSearchCV",94.4])
table.add_row(["Decision Tree","GridSearchCV",83.5])
table.add_row(["Random Forest Classifier","GridSearchCV",91.5])
table.add_row(["GBDT","GridSearchCV",90.4])

print(table.get_string(title="ML Model Results"))
```

```
+-------------------------------------------------------------+
|                       ML Model Results                      |
+--------------------------+----------------------+----------+
|          Model           | Hyperparameter Tuing | Accuracy |
```

```
+--------------------------+----------------------+----------+
|   Logistic Regression    |     GridSearchCV     |   94.5   |
|        Linear SVC        |     GridSearchCV     |   94.6   |
|        Kernal SVM        |     GridSearchCV     |   94.4   |
|      Decision Tree       |     GridSearchCV     |   83.5   |
| Random Forest Classifier |     GridSearchCV     |   91.5   |
|           GBDT           |     GridSearchCV     |   90.4   |
+--------------------------+----------------------+----------+
```

In [24]:

```python
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ["Model","LSTM Layers","Optimizer","Accuracy","Loss"]
table.add_row([1,1,"rmsprop",89.54,0.41])
table.add_row([2,2,"rmsprop",91.01,0.44])
table.add_row([3,4,"rmsprop",91.72,0.32])
table.add_row([4,4,"adam",94.02,0.20])

print(table.get_string(title="LSTM Model Results"))
```

```
+----------------------------------------------------+
|                 LSTM Model Results                 |
+-------+-------------+-----------+----------+------+
| Model | LSTM Layers | Optimizer | Accuracy | Loss |
+-------+-------------+-----------+----------+------+
|   1   |      1      |  rmsprop  |  89.54   | 0.41 |
|   2   |      2      |  rmsprop  |  91.01   | 0.44 |
|   3   |      4      |  rmsprop  |  91.72   | 0.32 |
|   4   |      4      |    adam   |  94.02   | 0.2  |
+-------+-------------+-----------+----------+------+
```

Conclusion

LinearSVC has given best accuracy 94.6 in ML models. LSTM with 4 layers has given best accuracy 94.02 and loss 0.2