HUMAN ACTIVITY RECOGNITION

In [1]:
```python
# Human Activity Image
from IPython.display import Image
Image(filename='Human Activities.png',width=900)
```

Out[1]:

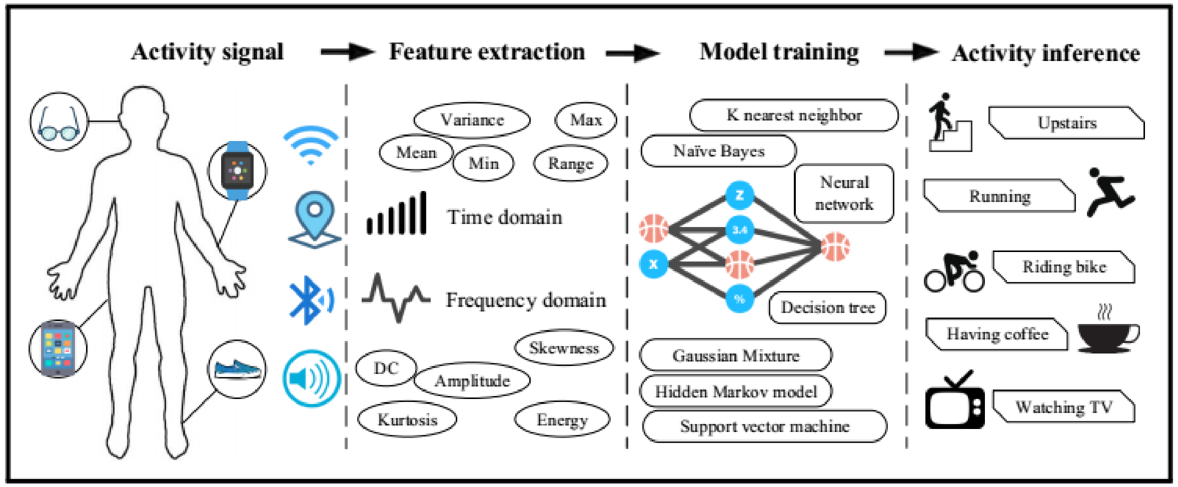


Table of Contents

1.Introduction

- This dataset is taken from UCI's machine learning repository.
- We are provided with sensory data from smartphone, now given this data our task is to predict the activity of a person.
- This is a multi-class classification problem. We have 6 activities as our class labels, they are:
- **Walking, Sitting, Standing, Laying down, Walking upstairs and walking downstairs**

2.Data Description

- Now what are smartphone sensors?
- A smartphone sensor is a type of sensing device installed in a phone to gather data for various user purpose often in conjuction with a mobile app.
- Few example of smartphone sensors are:
- Accelerometer,gyroscope,A proximity sensor, Finger print sensor and so on.
- In this current dataset we only use accelrometer and gyroscope.

What is accelerometer?

- Accelerometer detects acceleration,vibration and tilt to determine movement and exact orientation along 3 axes. Apps use this smartphone sensor to determine whether the phone is in portrait mode or landscape mode. It can also tell you whether the phone screen is facing up or down.

What is a gryroscope?

- Gyroscope provides orientation details and direction like up/down, left/right but with greater precision like how much the device is tilted. Gyroscope can measure rotation too. So it can tell you how much a smartphone is rotated and in which direction. Popular apps like Pokemon Go,Google Sky Map uses gyroscope like sensors to determine the direction towards which our phone is tilted.

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 are: 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

- We can estimate some set of variables from the above signals. ie., We will estimate the following properties on each and every signal that we recorded so far. ***mean()***: Mean value ***std()***: Standard deviation ***mad()***: Median absolute deviation ***max()***: Largest value in array ***min()***: Smallest value in array ***sma()***: Signal magnitude area ***energy()***: Energy measure. Sum of the squares divided by the number of values. ***iqr()***: Interquartile range ***entropy()***: Signal entropy ***arCoeff()***: Autorregresion coefficients with Burg order equal to 4 ***correlation()***: correlation coefficient between two signals ***maxInds()***: index of the frequency component with largest magnitude ***meanFreq()***: Weighted average of the frequency components to obtain a mean frequency ***skewness()***: skewness of the frequency domain signal ***kurtosis()***: kurtosis of the frequency domain signal ***bandsEnergy()***: Energy of a frequency interval within the 64 bins of the FFT of each window. ***angle()***: Angle between to vectors.
- 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 seperated

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

Performance Metric(s)

- Accuracy
- Confusion Matrix

Objective

We will have a quick overview of the data

- Accelerometer and Gyroscope readings are taken from 30 volunteers(referred as subjects) while performing the following 6 Activities. **Walking** , **Walkingupstairs** , **Walkingdownstairs** , **Standing** , **Sitting** , **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.

OBJECTIVE

- Given a new datapoint we have to predict the Activity

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.

Reading data for analysis

In [2]:
```
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
```

In [3]:
```
! wget --header="Host: doc-0c-9o-docs.googleusercontent.com" --header=
"User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.3
6 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36" --header="Acc
ept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,i
mage/apng,*/*;q=0.8" --header="Accept-Language: en-US,en;q=0.9" --heade
r="Referer: https://drive.google.com/drive/my-drive" --header="Cookie:
 AUTH_lj40l4f1btnitp0jsbip6m5ulebavm34_nonce=chm27b290vbfm; _ga=GA1.2.9
09748213.1552671681; _gid=GA1.2.2089903835.1554080079" --header="Connec
tion: keep-alive" "https://doc-0c-9o-docs.googleusercontent.com/docs/se
curesc/a6ek8quepju8gd1inh0e2jccjqr0c54d/no8oeul1fk3albih08ncnnfb8qbv4h6
u/1554098400000/13629942648867610103/13629942648867610103/1QB_6WhDqeF-z
w3ycZo2mpSm1FY2v7c7F?e=download&nonce=chm27b290vbfm&user=13629942648867
610103&hash=lsl8egat3jmttcnapoe4eiu2j2fkv1c6" -O "X_train.txt" -c
```

```
--2019-04-01 10:02:32--  https://doc-0c-9o-docs.googleusercontent.com/d
ocs/securesc/a6ek8quepju8gd1inh0e2jccjqr0c54d/no8oeul1fk3albih08ncnnfb8
qbv4h6u/1554098400000/13629942648867610103/13629942648867610103/1QB_6Wh
DqeF-zw3ycZo2mpSm1FY2v7c7F?e=download&nonce=chm27b290vbfm&user=13629942
648867610103&hash=lsl8egat3jmttcnapoe4eiu2j2fkv1c6
Resolving doc-0c-9o-docs.googleusercontent.com (doc-0c-9o-docs.googleus
ercontent.com)... 74.125.20.132, 2607:f8b0:400e:c07::84
Connecting to doc-0c-9o-docs.googleusercontent.com (doc-0c-9o-docs.goog
leusercontent.com)|74.125.20.132|:443... connected.
HTTP request sent, awaiting response... 416 Requested range not satisfi
```

```
able

    The file is already fully retrieved; nothing to do.
```

Obtain the train data

In [4]:
```
# Obtain the data
import warnings
warnings.simplefilter("ignore")
# 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, sque
eze=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:'WAL
KING_DOWNSTAIRS',\
                       4:'SITTING', 5:'STANDING',6:'LAYING'})

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

Out[4]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X
4562	0.283514	-0.0061	-0.099433	-0.99345	-0.935157	-0.964718	-0.993687

1 rows × 564 columns

In [5]:
```
# Dimension of the dataset
train.shape
```

Out[5]:
```
(7352, 564)
```

In [6]:
```
! wget --header="Host: doc-08-9o-docs.googleusercontent.com" --header=
"User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.3
6 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36" --header="Acc
ept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,i
mage/apng,*/*;q=0.8" --header="Accept-Language: en-US,en;q=0.9" --heade
r="Referer: https://drive.google.com/drive/my-drive" --header="Cookie:
 AUTH_lj40l4f1btnitp0jsbip6m5ulebavm34=13629942648867610103|15540984000
00|r6bfg3me1ub65inmkg3ud3148jdlrp1q; _ga=GA1.2.909748213.1552671681; _g
id=GA1.2.2089903835.1554080079" --header="Connection: keep-alive" "http
s://doc-08-9o-docs.googleusercontent.com/docs/securesc/a6ek8quepju8gd1i
nh0e2jccjqr0c54d/h550071kl0e6sqj9hp4md11en8svdutp/1554098400000/1362994
2648867610103/13629942648867610103/13uTsYokGlKMCssf7SIrm92Zt-32ID2w4?e=
download" -O "X_test.txt" -c
```

```
--2019-04-01 10:02:54--  https://doc-08-9o-docs.googleusercontent.com/d
ocs/securesc/a6ek8quepju8gd1inh0e2jccjqr0c54d/h550071kl0e6sqj9hp4md11en
8svdutp/1554098400000/13629942648867610103/13629942648867610103/13uTsYo
kGlKMCssf7SIrm92Zt-32ID2w4?e=download
Resolving doc-08-9o-docs.googleusercontent.com (doc-08-9o-docs.googleus
ercontent.com)... 74.125.20.132, 2607:f8b0:400e:c07::84
Connecting to doc-08-9o-docs.googleusercontent.com (doc-08-9o-docs.goog
leusercontent.com)|74.125.20.132|:443... connected.
HTTP request sent, awaiting response... 416 Requested range not satisfi
able

    The file is already fully retrieved; nothing to do.
```

Obtain the test dataset

In [7]:
```
import warnings
warnings.simplefilter("ignore")
```

```
# 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, squeez
e=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:'WALKI
NG_DOWNSTAIRS',\
                       4:'SITTING', 5:'STANDING',6:'LAYING'})


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

Out[7]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	t
1420	0.208316	-0.003139	-0.114968	-0.968483	-0.932202	-0.933711	-0.973101	-

1 rows × 564 columns

In [7]:
```
test.shape
```

Out[7]:
```
(2947, 564)
```

4. Exploratory Data Analysis - Data cleaning steps

We will perform basic data cleaning steps like checking out for duplicates,finding missing values, checking for data imbalance

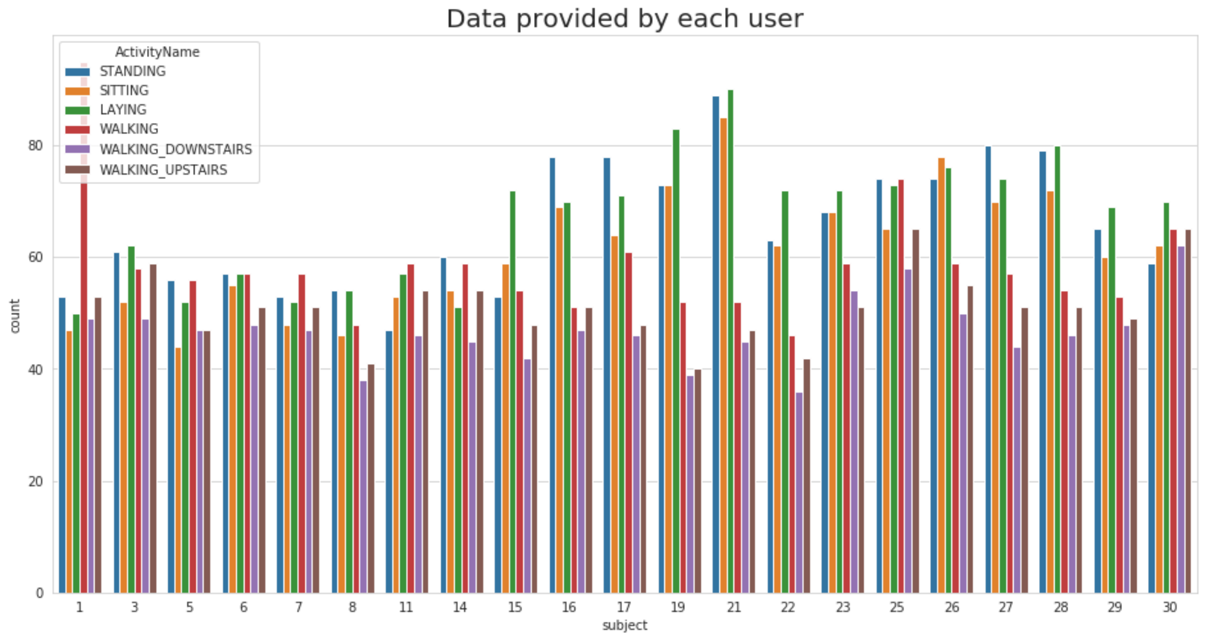
In [8]:
```
# Check for duplicates
print('No of duplicates in train: {}'.format(sum(train.duplicated())))
print('No of duplicates in test : {}'.format(sum(test.duplicated())))
```

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

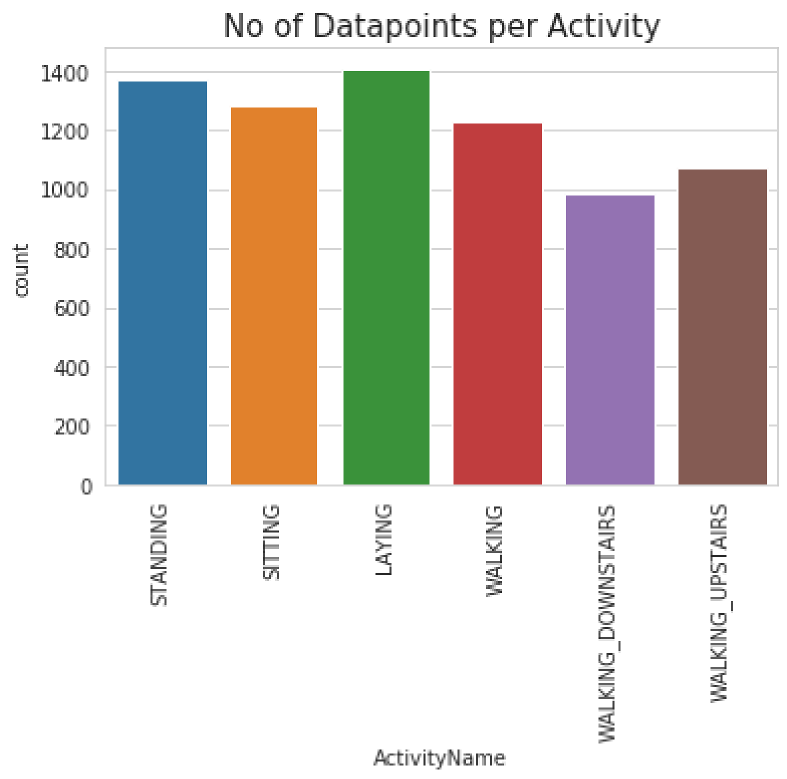
In [9]:
```
# Checking for NaN/values
print('We have {} NaN/Null values in train'.format(train.isnull().value
s.sum()))
print('We have {} NaN/Null values in test'.format(test.isnull().values.
sum()))
```

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

In [10]:
```
# Checking for data imbalance
% matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

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

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



In [11]:
```
# Bar plot
plt.title('No of Datapoints per Activity', fontsize=15)
sns.countplot(train.ActivityName)
plt.xticks(rotation=90)
plt.show()

# From the plot we can say that our data well balanced. There is no dom
inance of one class over the others.
```



In [12]:
```
# Replacing feature names - We will remove hypen,punctuation mark, brac
kets from our feature names.
columns = train.columns

columns = columns.str.replace('[()]','')
columns = columns.str.replace('[-]', '')
columns = columns.str.replace('[,]','')

train.columns = columns
test.columns = columns

test.columns
```

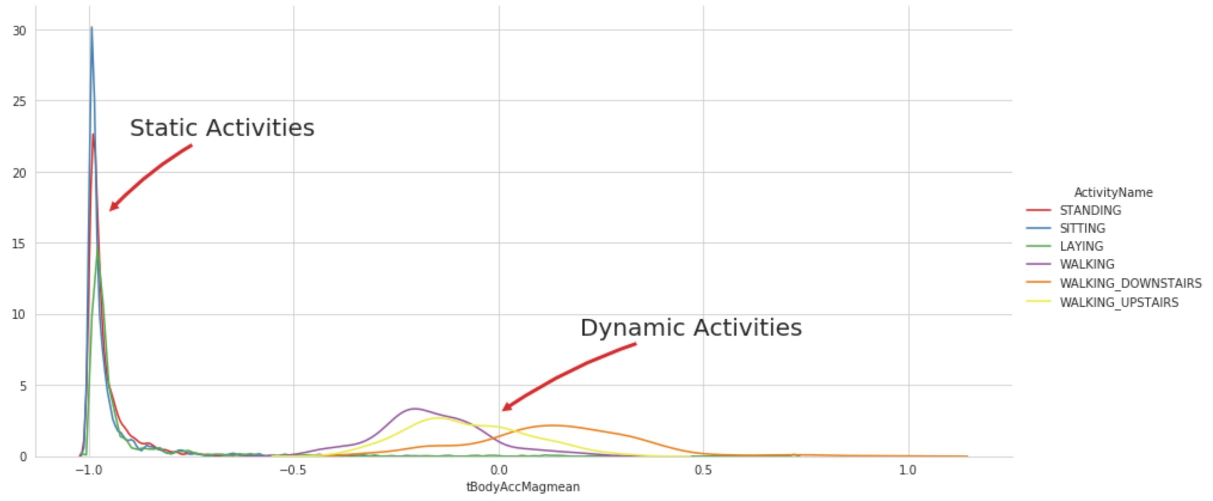
Out[12]:
```
Index(['tBodyAccmeanX', 'tBodyAccmeanY', 'tBodyAccmeanZ', 'tBodyAccstd
X',
       'tBodyAccstdY', 'tBodyAccstdZ', 'tBodyAccmadX', 'tBodyAccmadY',
       'tBodyAccmadZ', 'tBodyAccmaxX',
       ...
       'angletBodyAccMeangravity', 'angletBodyAccJerkMeangravityMean',
       'angletBodyGyroMeangravityMean',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

In [13]:
```
# Saving the file in csv format.
train.to_csv('train.csv', index=False)
test.to_csv('test.csv', index=False)
```

5. Plotting few features to understand the data better

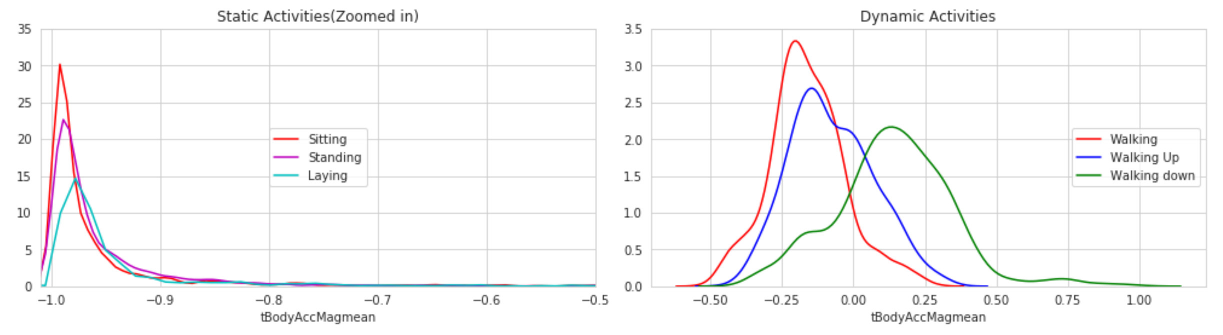
- If we observe our class labels correctly, the activities like sitting,standing or laying down can be termed as **Static activities**
- Also the other 3 activities walking,walking upstairs,wlking downstairs can be termed as **Dynamic activities**

In [14]:
```
# When we plot the two we can get more infomation
import warnings
warnings.filterwarnings("ignore")
% matplotlib inline

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("Static Activities", xy=(-0.956,17), xytext=(-0.9, 23), si
ze=20,\
            va='center', ha='left',\
            arrowprops=dict(arrowstyle="simple",connectionstyle="arc3,r
ad=0.1"))

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



In [15]:
```
# For plotting purposes taking datapoints of each activity to a differe
nt 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('Static Activities(Zoomed in)')
sns.distplot(df4['tBodyAccMagmean'],color = 'r',hist = False, label =
'Sitting')
sns.distplot(df5['tBodyAccMagmean'],color = 'm',hist = False,label = 'S
tanding')
sns.distplot(df6['tBodyAccMagmean'],color = 'c',hist = False, label =
'Laying')
```

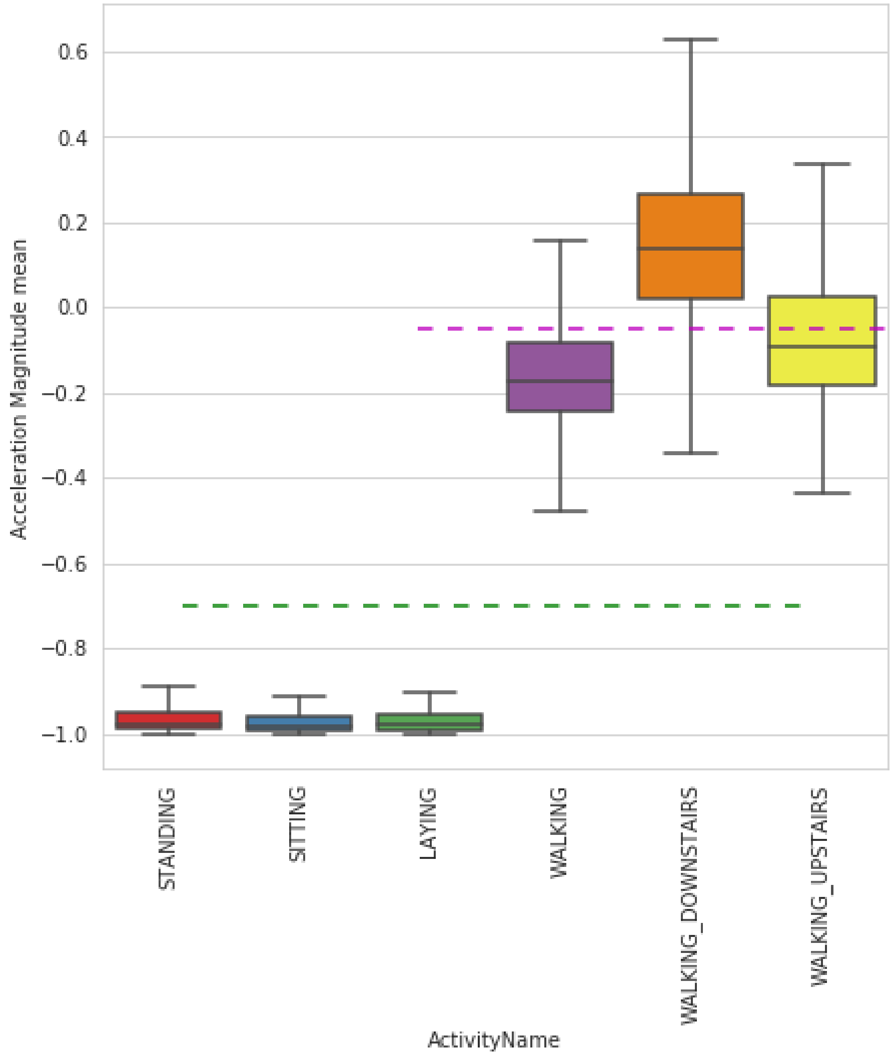
```
plt.axis([-1.01, -0.5, 0, 35])
plt.legend(loc='center')

plt.subplot(2,2,2)
plt.title('Dynamic 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()
```



In [16]:

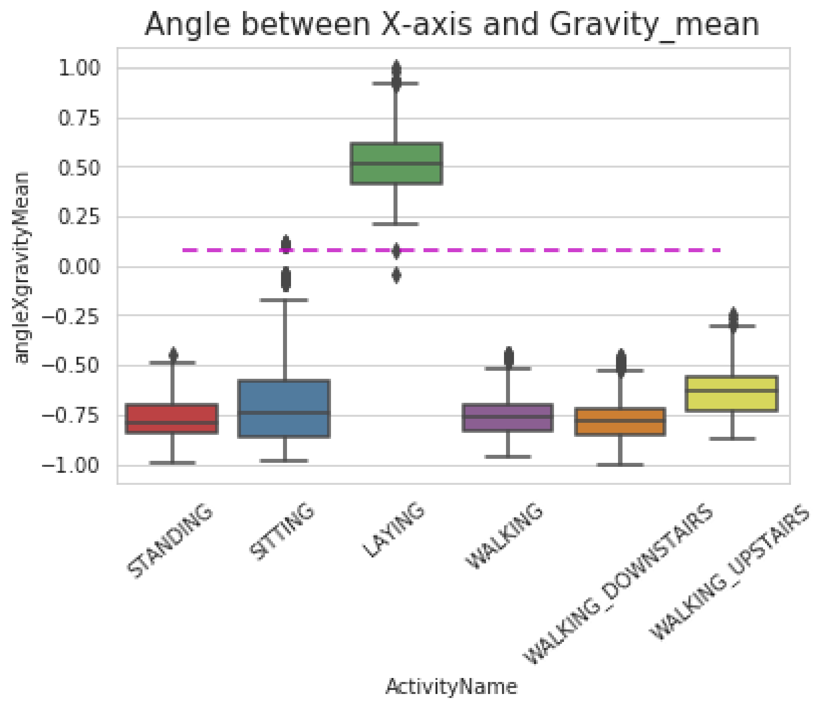
```
# Magnitude of an acceleration feature separates the classes very well

plt.figure(figsize=(7,7))
sns.boxplot(x='ActivityName', y='tBodyAccMagmean',data=train, showflier
s=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()

# We have set a threshold value to this plot, If the activity is below
```

```
 that threshold value we consider them as static activity
# or it is considered as dynamic activity. For example: If tAccMean is
 < -0.8 then the Activities are either Standing or
# Sitting or Laying. If tAccMean is > -0.6 then the Activities are eith
er Walking or WalkingDownstairs or WalkingUpstairs.
```

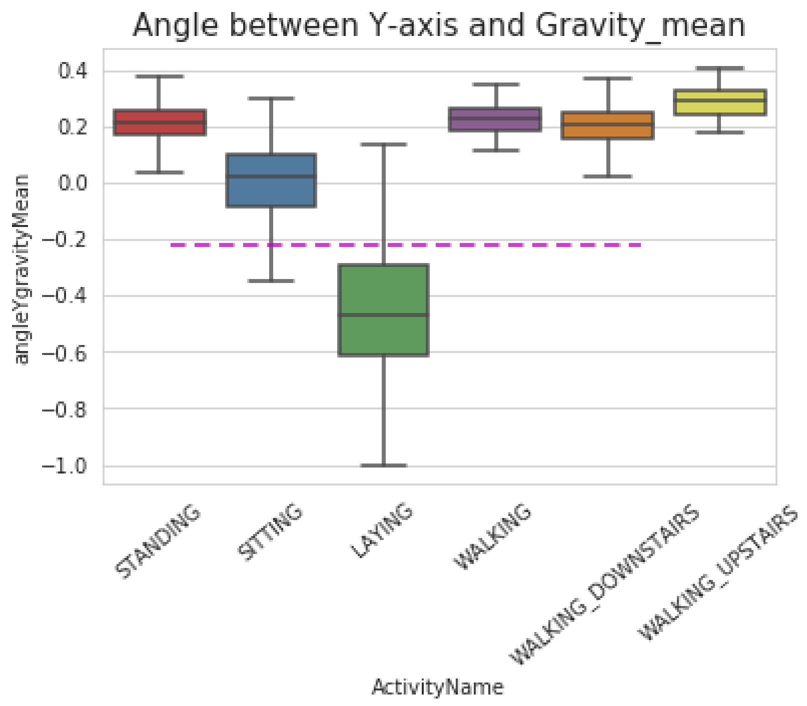


In [17]:
```
# Position of GravityAccelerationComponants also matters
```

```
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()

# 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]:
```
# Plotting angleYgravityMean
sns.boxplot(x='ActivityName', y='angleYgravityMean', data = train, show
fliers=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()

# Again here with a simple if-else statement like - If angleY,gravityMe
an < -0.2 then Activity is Laying. we can classify the
# label laying down.
```



In [19]:
```
#### Applying t-sne on the data

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

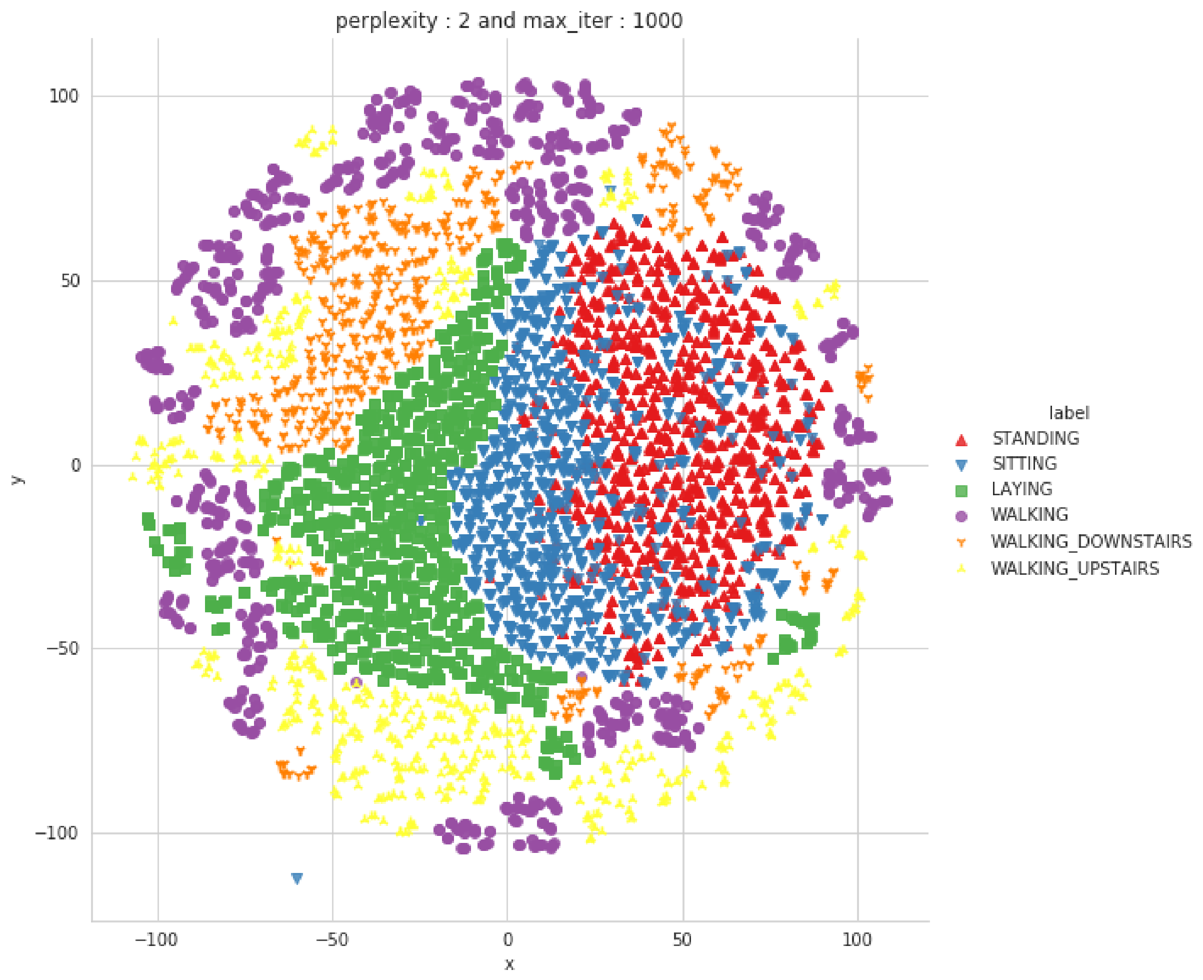
In [20]:
```
# Here we map 561 dimension data into a 2 dimension data.

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

    for index,perplexity in enumerate(perplexities):
        # perform t-sne
        print('\nperforming tsne with perplexity {} and with {} iterati
ons at max'.format(perplexity, n_iter))
        X_reduced = TSNE(verbose=2, perplexity=perplexity).fit_transfor
m(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] ,'lab
el':y_data})

        # draw the plot in appropriate place in the grid
        sns.lmplot(data=df, x='x', y='y', hue='label', fit_reg=False, s
ize=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(perp
lexity, n_iter)
        print('saving this plot as image in present working director
y...')
        plt.savefig(img_name)
        plt.show()
        print('Done')
```

In [21]:
```
# t-sne with perplexity of 2
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])
```

```
performing tsne with perplexity 2 and with 1000 iterations at max
[t-SNE] Computing 7 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.262s...
[t-SNE] Computed neighbors for 7352 samples in 46.838s...
[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 0.043s
[t-SNE] Iteration 50: error = 124.7493362, gradient norm = 0.0265852 (5
0 iterations in 5.146s)
[t-SNE] Iteration 100: error = 107.6834412, gradient norm = 0.0282927
(50 iterations in 3.533s)
[t-SNE] Iteration 150: error = 101.2511292, gradient norm = 0.0206050
(50 iterations in 2.736s)
[t-SNE] Iteration 200: error = 97.7431793, gradient norm = 0.0165400 (5
0 iterations in 2.654s)
[t-SNE] Iteration 250: error = 95.3990936, gradient norm = 0.0134456 (5
0 iterations in 2.623s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 95.
399094
[t-SNE] Iteration 300: error = 4.1235919, gradient norm = 0.0015656 (50
iterations in 2.340s)
[t-SNE] Iteration 350: error = 3.2130418, gradient norm = 0.0009935 (50
iterations in 2.187s)
[t-SNE] Iteration 400: error = 2.7830787, gradient norm = 0.0007150 (50
iterations in 2.239s)
[t-SNE] Iteration 450: error = 2.5190017, gradient norm = 0.0005655 (50
iterations in 2.241s)
[t-SNE] Iteration 500: error = 2.3353491, gradient norm = 0.0004784 (50
iterations in 2.244s)
[t-SNE] Iteration 550: error = 2.1973324, gradient norm = 0.0004136 (50
iterations in 2.269s)
[t-SNE] Iteration 600: error = 2.0879865, gradient norm = 0.0003706 (50
iterations in 2.333s)
[t-SNE] Iteration 650: error = 1.9980880, gradient norm = 0.0003323 (50
iterations in 2.314s)
[t-SNE] Iteration 700: error = 1.9224949, gradient norm = 0.0002997 (50
iterations in 2.305s)
```

```
[t-SNE] Iteration 750: error = 1.8573335, gradient norm = 0.0002772 (50
iterations in 2.272s)
[t-SNE] Iteration 800: error = 1.8007392, gradient norm = 0.0002566 (50
iterations in 2.267s)
[t-SNE] Iteration 850: error = 1.7509712, gradient norm = 0.0002387 (50
iterations in 2.329s)
[t-SNE] Iteration 900: error = 1.7062844, gradient norm = 0.0002255 (50
iterations in 2.346s)
[t-SNE] Iteration 950: error = 1.6661959, gradient norm = 0.0002110 (50
iterations in 2.293s)
[t-SNE] Iteration 1000: error = 1.6301001, gradient norm = 0.0001975 (5
0 iterations in 2.301s)
[t-SNE] KL divergence after 1000 iterations: 1.630100
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```



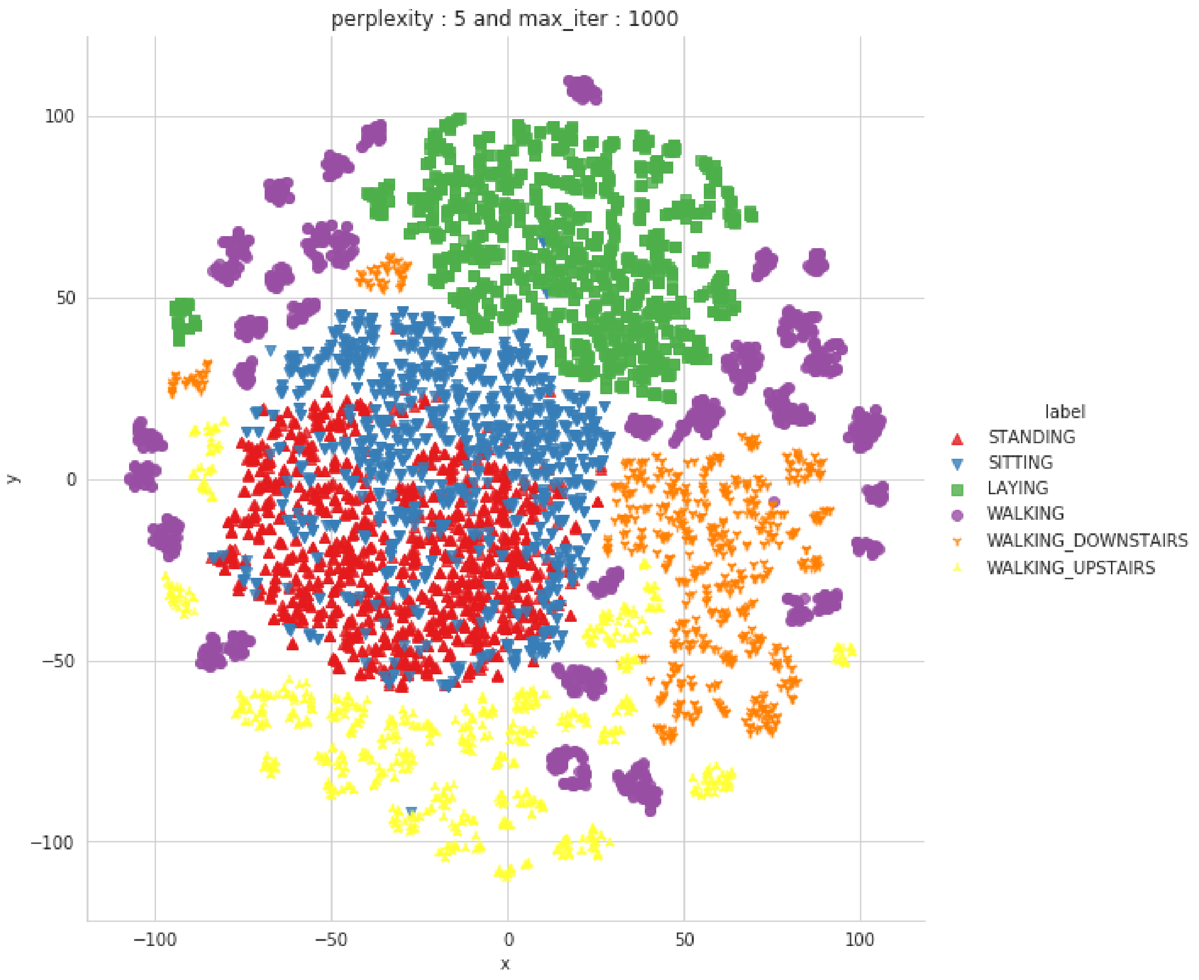
```
Done
```

```
In [22]: # t-sne with perplexity of 5
         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 =[5])
```

```
performing tsne with perplexity 5 and with 1000 iterations at max
[t-SNE] Computing 16 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.271s...
```

```
[t-SNE] Computed neighbors for 7352 samples in 46.392s...
[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 0.072s
[t-SNE] Iteration 50: error = 113.9519196, gradient norm = 0.0211731 (5
0 iterations in 10.965s)
[t-SNE] Iteration 100: error = 97.6018295, gradient norm = 0.0146886 (5
0 iterations in 3.011s)
[t-SNE] Iteration 150: error = 93.1757889, gradient norm = 0.0098560 (5
0 iterations in 2.405s)
[t-SNE] Iteration 200: error = 91.1766052, gradient norm = 0.0061800 (5
0 iterations in 2.224s)
[t-SNE] Iteration 250: error = 89.9785919, gradient norm = 0.0092150 (5
0 iterations in 2.141s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 89.
978592
[t-SNE] Iteration 300: error = 3.5679553, gradient norm = 0.0014538 (50
iterations in 2.143s)
[t-SNE] Iteration 350: error = 2.8121483, gradient norm = 0.0007427 (50
iterations in 2.145s)
[t-SNE] Iteration 400: error = 2.4332764, gradient norm = 0.0005228 (50
iterations in 2.191s)
[t-SNE] Iteration 450: error = 2.2165484, gradient norm = 0.0004082 (50
iterations in 2.205s)
[t-SNE] Iteration 500: error = 2.0719385, gradient norm = 0.0003326 (50
iterations in 2.220s)
[t-SNE] Iteration 550: error = 1.9669892, gradient norm = 0.0002814 (50
iterations in 2.212s)
[t-SNE] Iteration 600: error = 1.8860511, gradient norm = 0.0002479 (50
iterations in 2.243s)
[t-SNE] Iteration 650: error = 1.8208932, gradient norm = 0.0002187 (50
iterations in 2.225s)
```

```
[t-SNE] Iteration 700: error = 1.7671012, gradient norm = 0.0001969 (50
iterations in 2.251s)
[t-SNE] Iteration 750: error = 1.7219945, gradient norm = 0.0001785 (50
iterations in 2.230s)
[t-SNE] Iteration 800: error = 1.6830827, gradient norm = 0.0001647 (50
iterations in 2.262s)
[t-SNE] Iteration 850: error = 1.6491663, gradient norm = 0.0001533 (50
iterations in 2.251s)
[t-SNE] Iteration 900: error = 1.6193430, gradient norm = 0.0001428 (50
iterations in 2.252s)
[t-SNE] Iteration 950: error = 1.5926923, gradient norm = 0.0001342 (50
iterations in 2.249s)
[t-SNE] Iteration 1000: error = 1.5689209, gradient norm = 0.0001266 (5
0 iterations in 2.241s)
[t-SNE] KL divergence after 1000 iterations: 1.568921
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```



```
Done
```

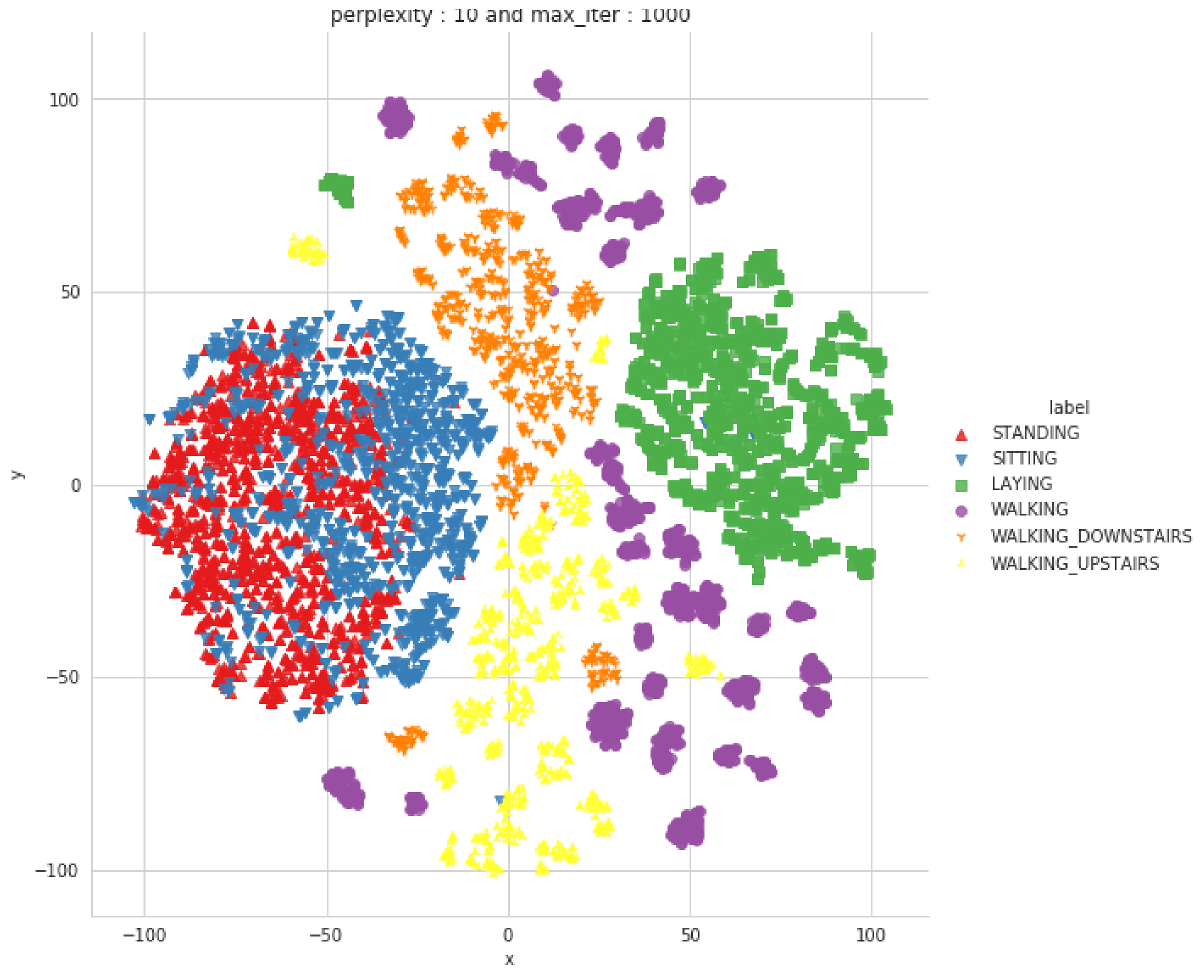
In [23]:

```
# t-sne with perlexity of 10
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 =[10])
```

```
performing tsne with perplexity 10 and with 1000 iterations at max
[t-SNE] Computing 31 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.263s...
```

```
[t-SNE] Computed neighbors for 7352 samples in 47.217s...
[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 0.135s
[t-SNE] Iteration 50: error = 105.7044601, gradient norm = 0.0212599 (5
0 iterations in 3.778s)
[t-SNE] Iteration 100: error = 90.6509781, gradient norm = 0.0113368 (5
0 iterations in 2.657s)
[t-SNE] Iteration 150: error = 87.8414764, gradient norm = 0.0098973 (5
0 iterations in 2.297s)
[t-SNE] Iteration 200: error = 86.5277481, gradient norm = 0.0041891 (5
0 iterations in 2.322s)
[t-SNE] Iteration 250: error = 85.7971191, gradient norm = 0.0031332 (5
0 iterations in 2.357s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.
797119
[t-SNE] Iteration 300: error = 3.1548436, gradient norm = 0.0013879 (50
iterations in 2.385s)
[t-SNE] Iteration 350: error = 2.5092218, gradient norm = 0.0006531 (50
iterations in 2.197s)
[t-SNE] Iteration 400: error = 2.1877387, gradient norm = 0.0004240 (50
iterations in 2.263s)
[t-SNE] Iteration 450: error = 2.0021565, gradient norm = 0.0003138 (50
iterations in 2.269s)
[t-SNE] Iteration 500: error = 1.8830646, gradient norm = 0.0002536 (50
iterations in 2.279s)
[t-SNE] Iteration 550: error = 1.7991781, gradient norm = 0.0002103 (50
iterations in 2.301s)
[t-SNE] Iteration 600: error = 1.7363416, gradient norm = 0.0001830 (50
iterations in 2.313s)
[t-SNE] Iteration 650: error = 1.6869342, gradient norm = 0.0001601 (50
iterations in 2.281s)
```

```
[t-SNE] Iteration 700: error = 1.6472588, gradient norm = 0.0001427 (50
iterations in 2.293s)
[t-SNE] Iteration 750: error = 1.6144001, gradient norm = 0.0001289 (50
iterations in 2.238s)
[t-SNE] Iteration 800: error = 1.5868902, gradient norm = 0.0001189 (50
iterations in 2.237s)
[t-SNE] Iteration 850: error = 1.5634907, gradient norm = 0.0001119 (50
iterations in 2.247s)
[t-SNE] Iteration 900: error = 1.5432428, gradient norm = 0.0001020 (50
iterations in 2.224s)
[t-SNE] Iteration 950: error = 1.5257078, gradient norm = 0.0000975 (50
iterations in 2.242s)
[t-SNE] Iteration 1000: error = 1.5105479, gradient norm = 0.0000925 (5
0 iterations in 2.273s)
[t-SNE] KL divergence after 1000 iterations: 1.510548
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```



```
Done
```

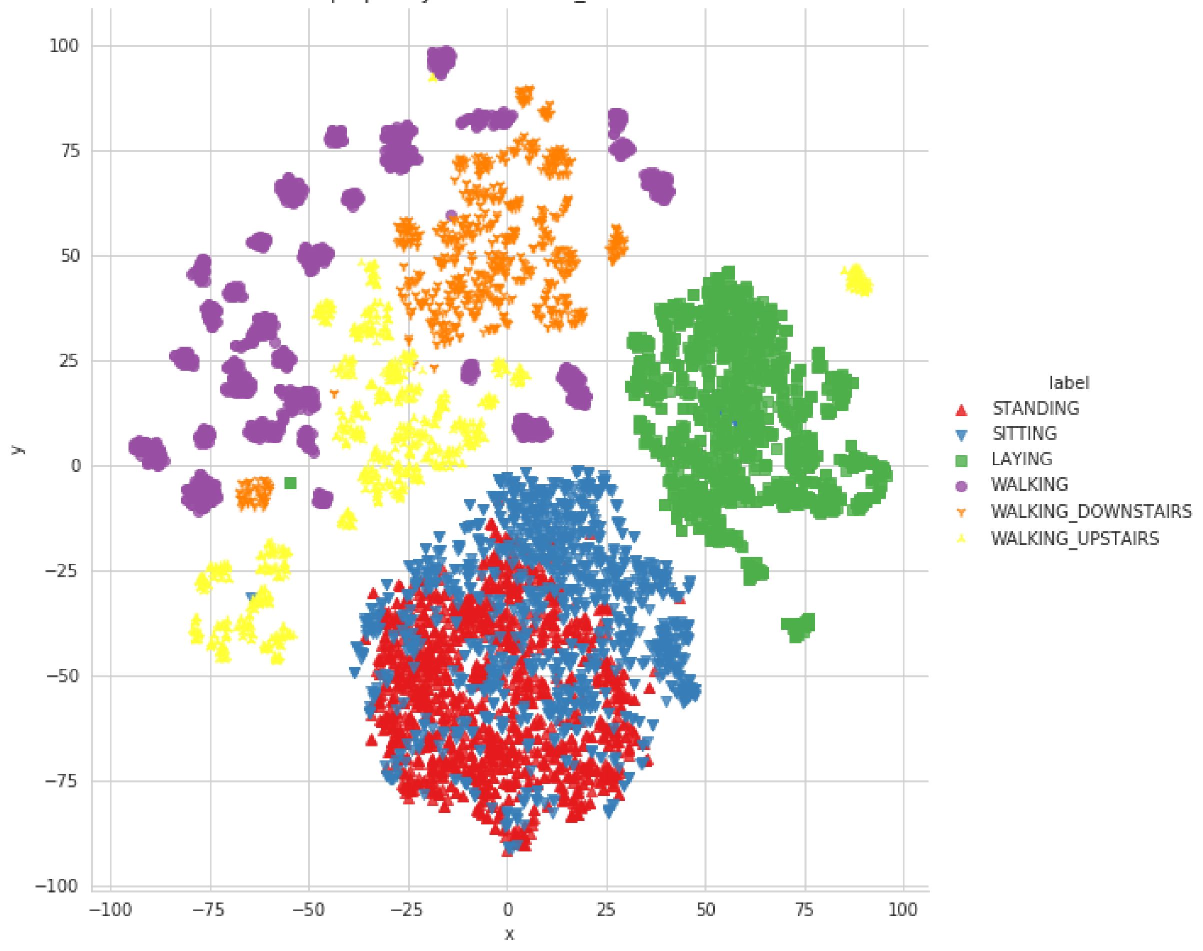
In [24]:
```
# t-sne with perplexity of 25
X_pre_tsne = train.drop(['subject', 'Activity','ActivityName'], axis=1)
y_pre_tsne = train['ActivityName']
perform_tsne(X_data = X_pre_tsne,y_data=y_pre_tsne, perplexities =[20])
```

```
performing tsne with perplexity 20 and with 1000 iterations at max
[t-SNE] Computing 61 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.261s...
```

```
[t-SNE] Computed neighbors for 7352 samples in 48.759s...
[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 0.260s
[t-SNE] Iteration 50: error = 97.0629578, gradient norm = 0.0240686 (50
iterations in 5.807s)
[t-SNE] Iteration 100: error = 83.6764297, gradient norm = 0.0068066 (5
0 iterations in 3.125s)
[t-SNE] Iteration 150: error = 81.8056870, gradient norm = 0.0043995 (5
0 iterations in 2.779s)
[t-SNE] Iteration 200: error = 81.1309891, gradient norm = 0.0021701 (5
0 iterations in 2.697s)
[t-SNE] Iteration 250: error = 80.7697144, gradient norm = 0.0018926 (5
0 iterations in 2.699s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 80.
769714
[t-SNE] Iteration 300: error = 2.6951613, gradient norm = 0.0013035 (50
iterations in 2.497s)
[t-SNE] Iteration 350: error = 2.1615787, gradient norm = 0.0005747 (50
iterations in 2.356s)
[t-SNE] Iteration 400: error = 1.9129652, gradient norm = 0.0003465 (50
iterations in 2.350s)
[t-SNE] Iteration 450: error = 1.7670560, gradient norm = 0.0002459 (50
iterations in 2.387s)
[t-SNE] Iteration 500: error = 1.6734955, gradient norm = 0.0001912 (50
iterations in 2.396s)
[t-SNE] Iteration 550: error = 1.6093736, gradient norm = 0.0001598 (50
iterations in 2.449s)
[t-SNE] Iteration 600: error = 1.5630196, gradient norm = 0.0001357 (50
iterations in 2.455s)
[t-SNE] Iteration 650: error = 1.5285522, gradient norm = 0.0001186 (50
iterations in 2.466s)
```

```
[t-SNE] Iteration 700: error = 1.5019333, gradient norm = 0.0001060 (50
iterations in 2.470s)
[t-SNE] Iteration 750: error = 1.4810089, gradient norm = 0.0001026 (50
iterations in 2.460s)
[t-SNE] Iteration 800: error = 1.4642619, gradient norm = 0.0000909 (50
iterations in 2.466s)
[t-SNE] Iteration 850: error = 1.4503953, gradient norm = 0.0000867 (50
iterations in 2.457s)
[t-SNE] Iteration 900: error = 1.4390157, gradient norm = 0.0000796 (50
iterations in 2.442s)
[t-SNE] Iteration 950: error = 1.4289066, gradient norm = 0.0000744 (50
iterations in 2.458s)
[t-SNE] Iteration 1000: error = 1.4199160, gradient norm = 0.0000723 (5
0 iterations in 2.488s)
[t-SNE] KL divergence after 1000 iterations: 1.419916
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```

perplexity : 20 and max_iter : 1000



```
Done
```

Insights from tsne plots across all perplexities

When we compare all the plots what we can infer is:

- All the classes are getting well separated except for standing and sitting classes. We find overlap only on standing and sitting classes. So given all 561 features we should be

able to separate all the classes except for standing and sitting.
- So take away point from tsne plots is even with different perplexity values we still face difficulties in separating standing and sitting classes. Otherwise all other classes are well separated.

6. Machine learning models

In [26]:
```
# Reading train & test dataset
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
print(train.shape, test.shape)
```

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

In [27]:
```
train.head(3)
```

Out[27]:

	tBodyAccmeanX	tBodyAccmeanY	tBodyAccmeanZ	tBodyAccstdX	tBodyAccstdY	tB
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.

3 rows × 564 columns

In [28]:
```
# Dropping subject,Activity & ActivityName columns from train dataset
X_train = train.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_train = train.ActivityName
```

In [29]:
```
# Dropping subject,Activity & ActivityName columns from train dataset
X_test = test.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_test = test.ActivityName
```

In [30]:
```
print('X_train and y_train : ({},{})'.format(X_train.shape, y_train.sha
pe))
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,))
```

Modelling the data

In [31]:
```
labels=['LAYING', 'SITTING','STANDING','WALKING','WALKING_DOWNSTAIRS',
'WALKING_UPSTAIRS']
```

In [32]:
```
# Function to plot the confusion matrix

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')
```

In [33]:
```
# Generic function to run any model.
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['trainin
g_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, tit
le='Normalized confusion matrix', cmap = cm_cmap)
    plt.show()

    # get classification report
    print('-------------------------')
    print('| Classifiction Report |')
    print('-------------------------')
    classification_report = metrics.classification_report(y_test, y_pre
d)
    # store report in results
    results['classification_report'] = classification_report
    print(classification_report)
    # add the trained  model to the results
    results['model'] = model

    return results
```

In [34]:

```
# Method to print grid search attribute
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.be
st_params_))


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


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

Logistic regression with Grid Search

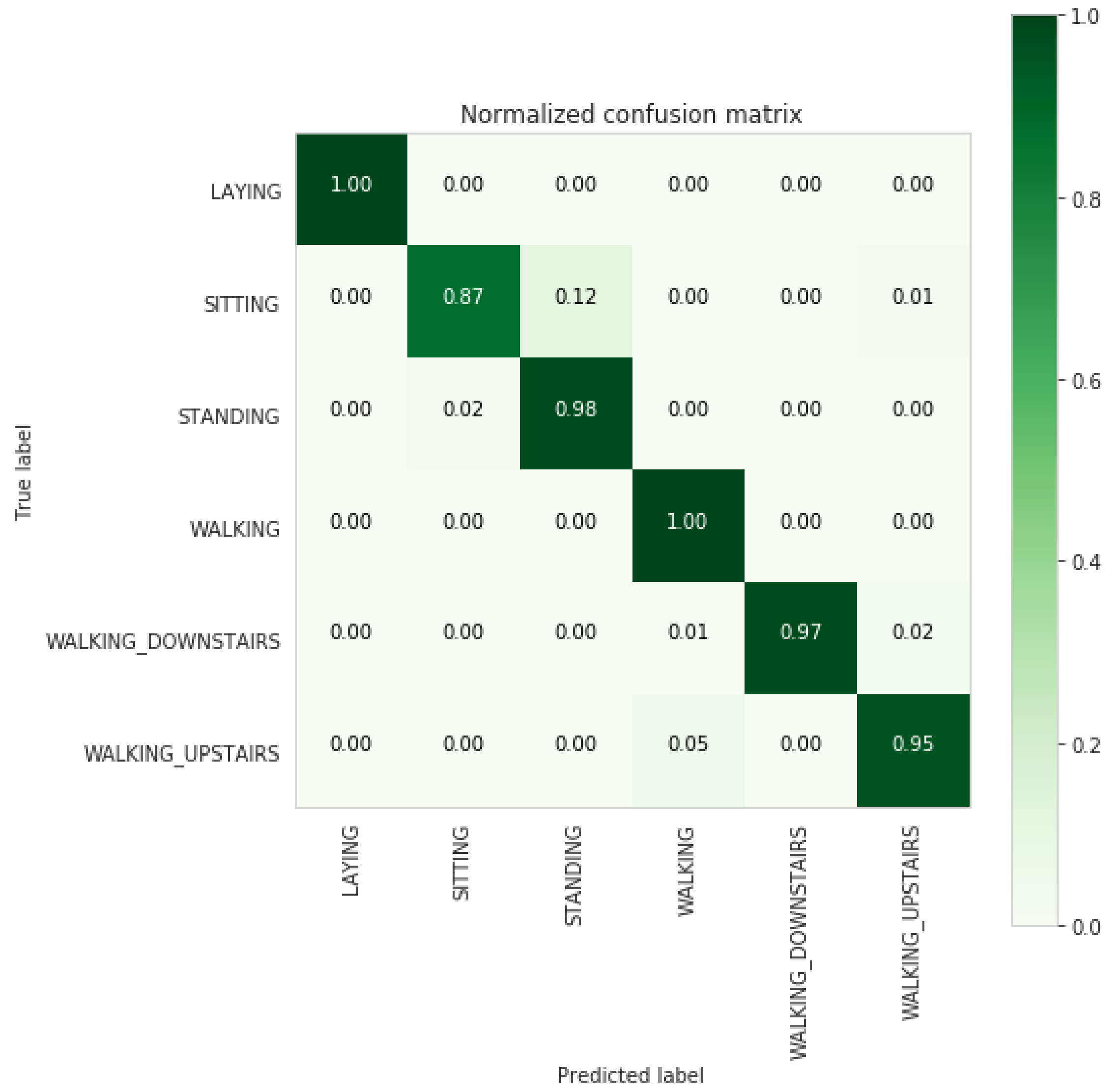
In [35]:
```
from sklearn import linear_model
from sklearn import metrics

from sklearn.model_selection import GridSearchCV

# 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, verbo
se=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 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  36 out of  36 | elapsed:   54.3s finished
```

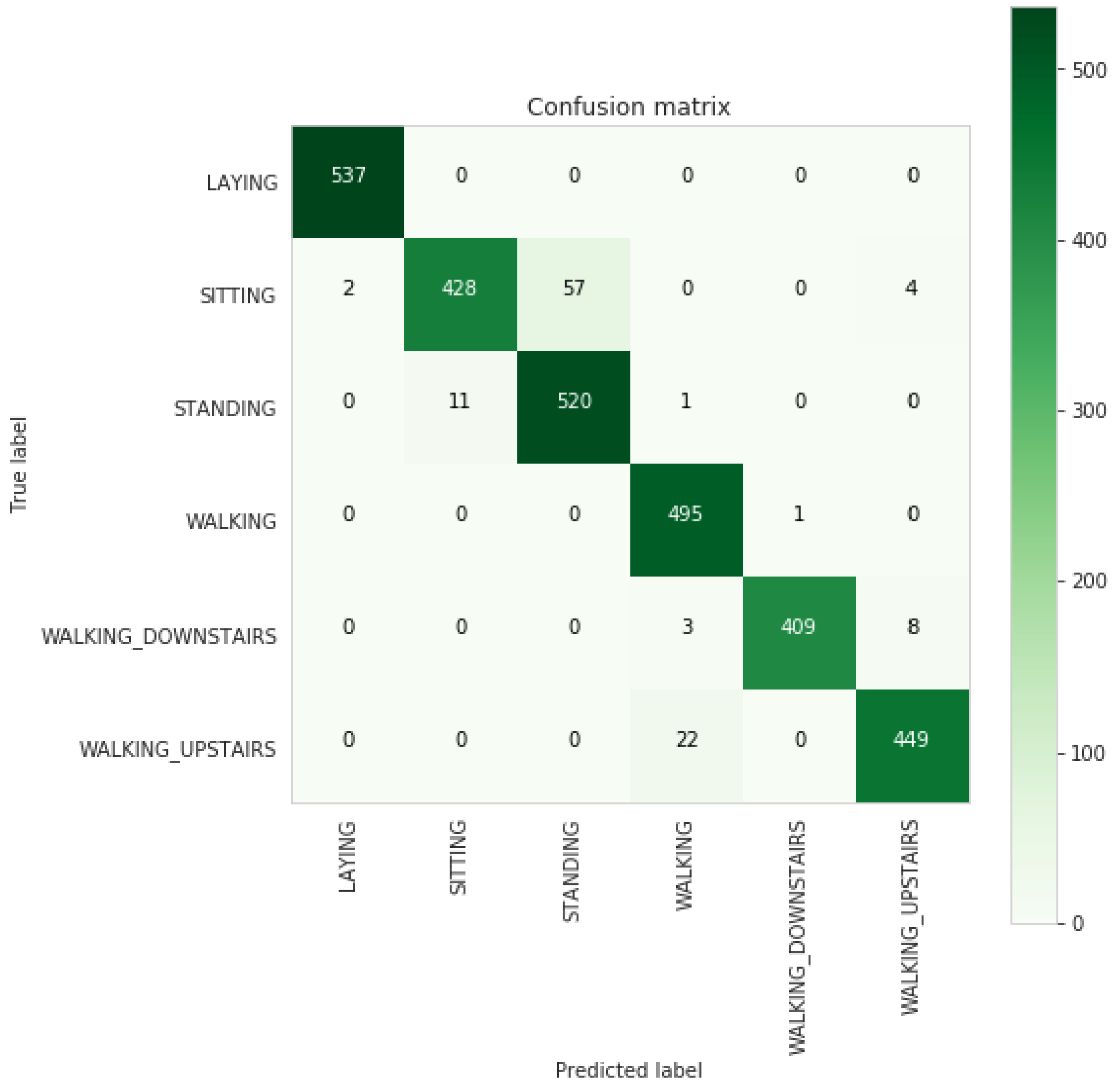
```
Done


training_time(HH:MM:SS.ms) - 0:01:09.316093


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.009731
```

```
---------------------
|      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]]
```



```
--------------------------
| Classification 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

         micro avg       0.96      0.96      0.96      2947
         macro avg       0.97      0.96      0.96      2947
      weighted avg       0.96      0.96      0.96      2947
```

In [36]:
```
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 [37]: # Observing 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_int
ercept=True,
          intercept_scaling=1, 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 : 

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

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator : 

        0.9461371055495104
```

Linear SVC with GridSearch

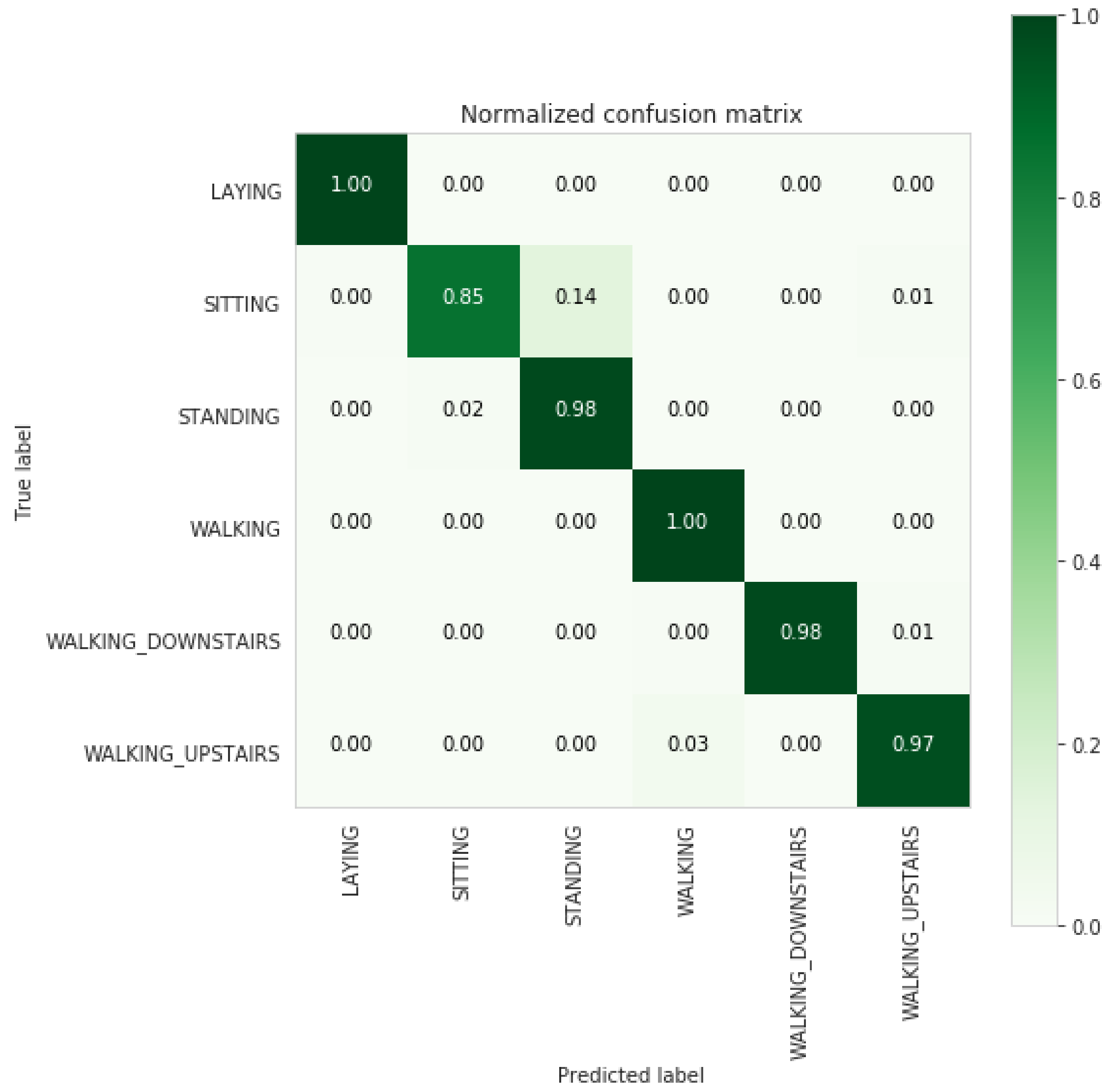
In [38]:
```
from sklearn.svm import LinearSVC

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, ve
rbose=1)
```

```
lr_svc_grid_results = perform_model(lr_svc_grid, X_train, y_train, X_te
st, y_test, class_labels=labels)
```

```
training the model..
Fitting 3 folds for each of 6 candidates, totalling 18 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  18 out of  18 | elapsed:   15.1s finished
```

```
Done


training_time(HH:MM:SS.ms) - 0:00:20.861278


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.005341


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

    0.9640312181879878


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

 [[537   0   0   0   0   0]
 [  2 417  67   0   0   5]
 [  0   8 523   1   0   0]
 [  0   0   0 496   0   0]
 [  0   0   0   2 413   5]
 [  0   0   0  15   1 455]]
```



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

            LAYING       1.00      1.00      1.00       537
```

```
           SITTING       0.98      0.85      0.91       491
          STANDING       0.89      0.98      0.93       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

         micro avg       0.96      0.96      0.96      2947
         macro avg       0.97      0.96      0.96      2947
      weighted avg       0.97      0.96      0.96      2947
```

In [39]:
```python
print_grid_search_attributes(lr_svc_grid_results['model'])
```

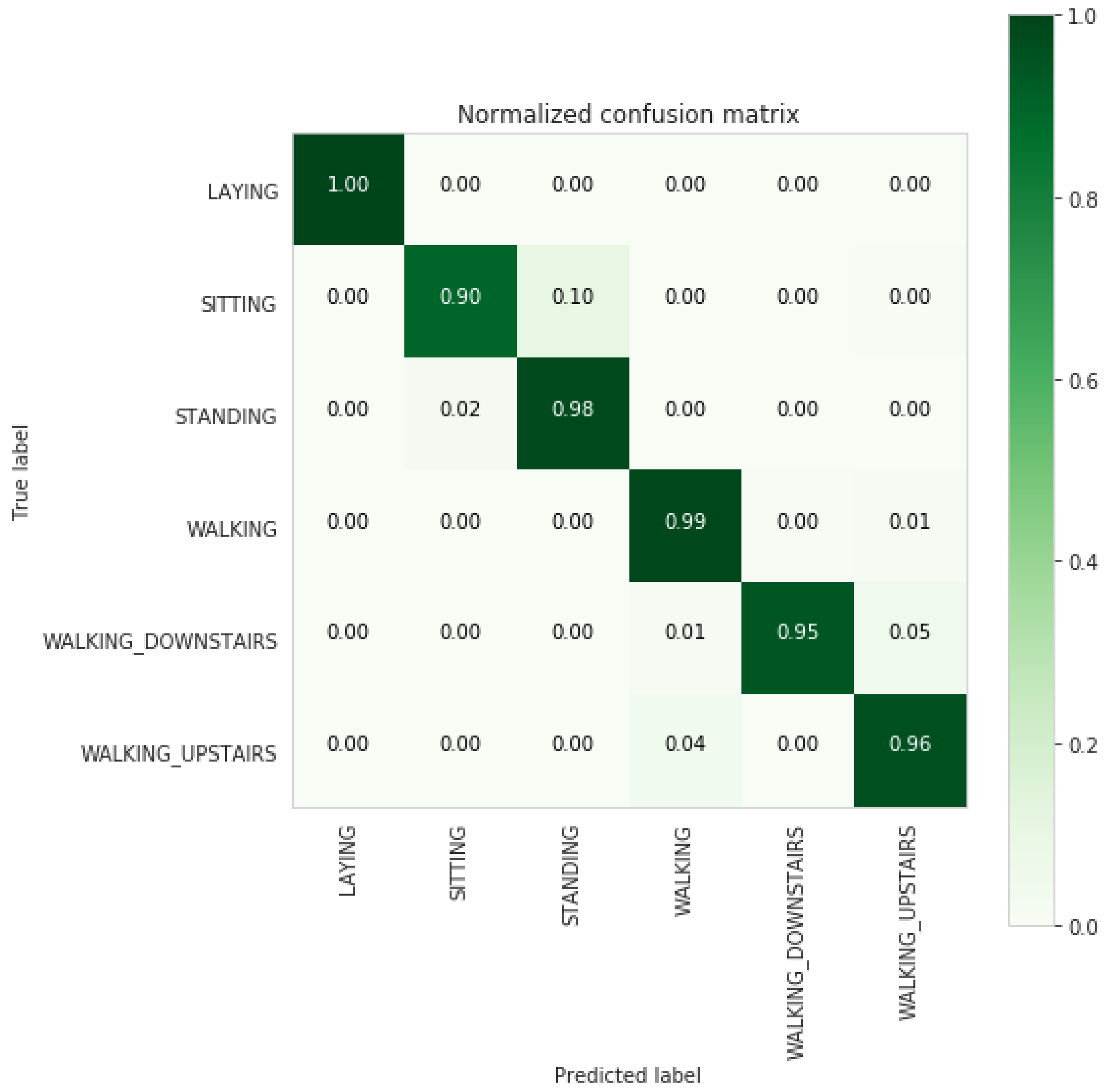
```
--------------------------
|      Best Estimator     |
--------------------------

        LinearSVC(C=2, class_weight=None, dual=True, fit_intercept=Tru
e,
     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': 2}

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

        Total numbre of cross validation sets: 3

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

```
        Average Cross Validate scores of best estimator :

        0.9462731229597389
```

Kernel SVM with GridSearch

In [40]:
```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=labels)
```

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:03:43.396983


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:03.041255


---------------------
|      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]]
```



	LAYING	SITTING	STANDING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
LAYING	1.00	0.00	0.00	0.00	0.00	0.00
SITTING	0.00	0.90	0.10	0.00	0.00	0.00
STANDING	0.00	0.02	0.98	0.00	0.00	0.00
WALKING	0.00	0.00	0.00	0.99	0.00	0.01
WALKING_DOWNSTAIRS	0.00	0.00	0.00	0.01	0.95	0.05
WALKING_UPSTAIRS	0.00	0.00	0.00	0.04	0.00	0.96

```
--------------------------
| Classification 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

         micro avg       0.96      0.96      0.96      2947
         macro avg       0.96      0.96      0.96      2947
      weighted avg       0.96      0.96      0.96      2947
```

In [41]:
```
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='rb
f',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

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

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

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

        Total numbre of cross validation sets: 3

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

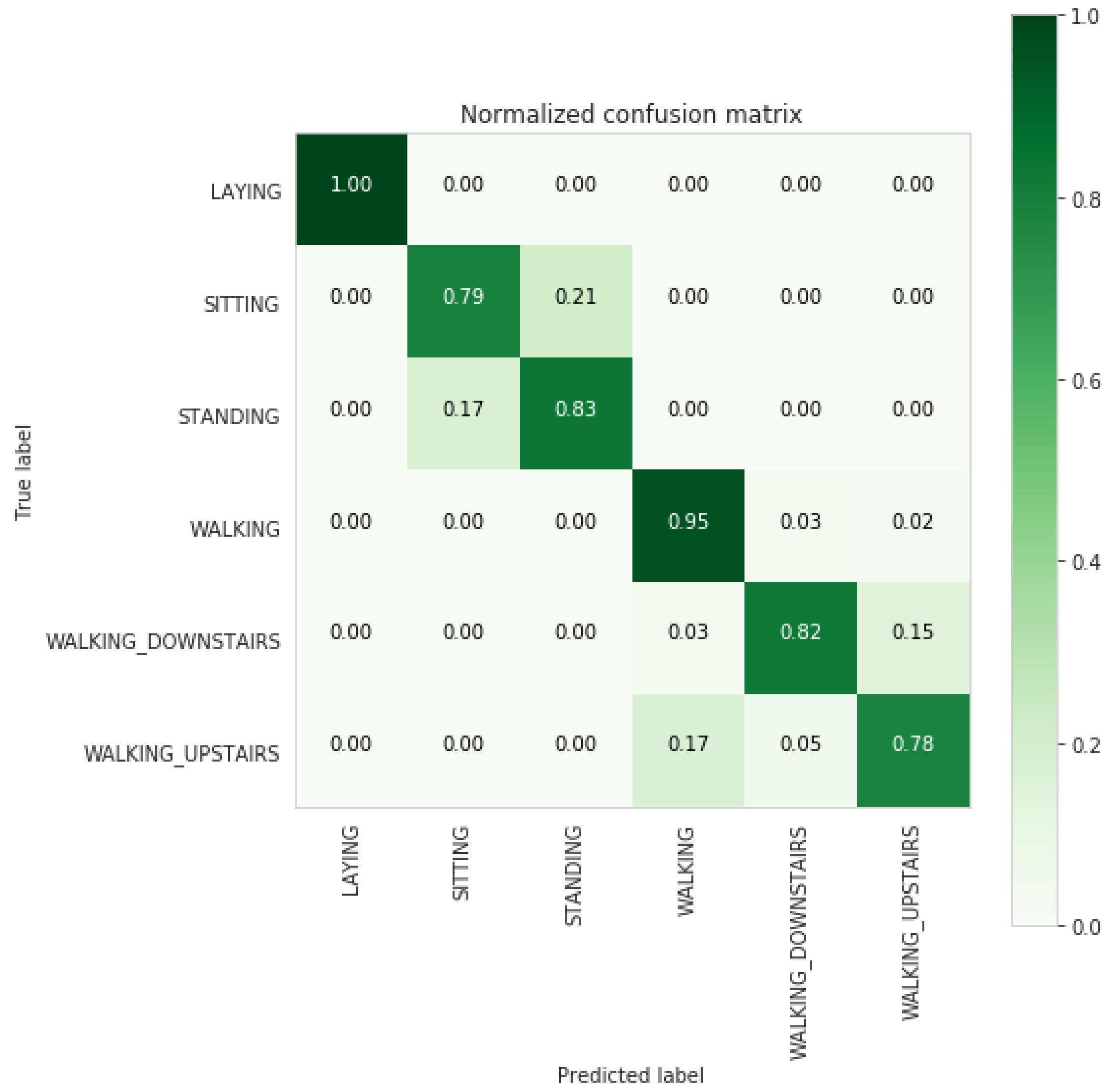
```
        Average Cross Validate scores of best estimator :

        0.9440968443960827
```

Decision Trees with GridSearchCV

In [42]:
```python
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_te
st, class_labels=labels)
print_grid_search_attributes(dt_grid_results['model'])
```

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:00:08.374110


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.007144


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

    0.8649474041398032


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

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

 [[537   0   0   0   0   0]
 [  0 388 103   0   0   0]
 [  0  93 439   0   0   0]
 [  0   0   0 471  17   8]
 [  0   0   0  14 345  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       0.84      0.95      0.89       496
WALKING_DOWNSTAIRS       0.89      0.82      0.86       420
  WALKING_UPSTAIRS       0.84      0.78      0.81       471

         micro avg       0.86      0.86      0.86      2947
         macro avg       0.86      0.86      0.86      2947
      weighted avg       0.87      0.86      0.86      2947

--------------------------
|      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=N
one,
            splitter='best')

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

        {'max_depth': 7}

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

        Total numbre of cross validation sets: 3

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

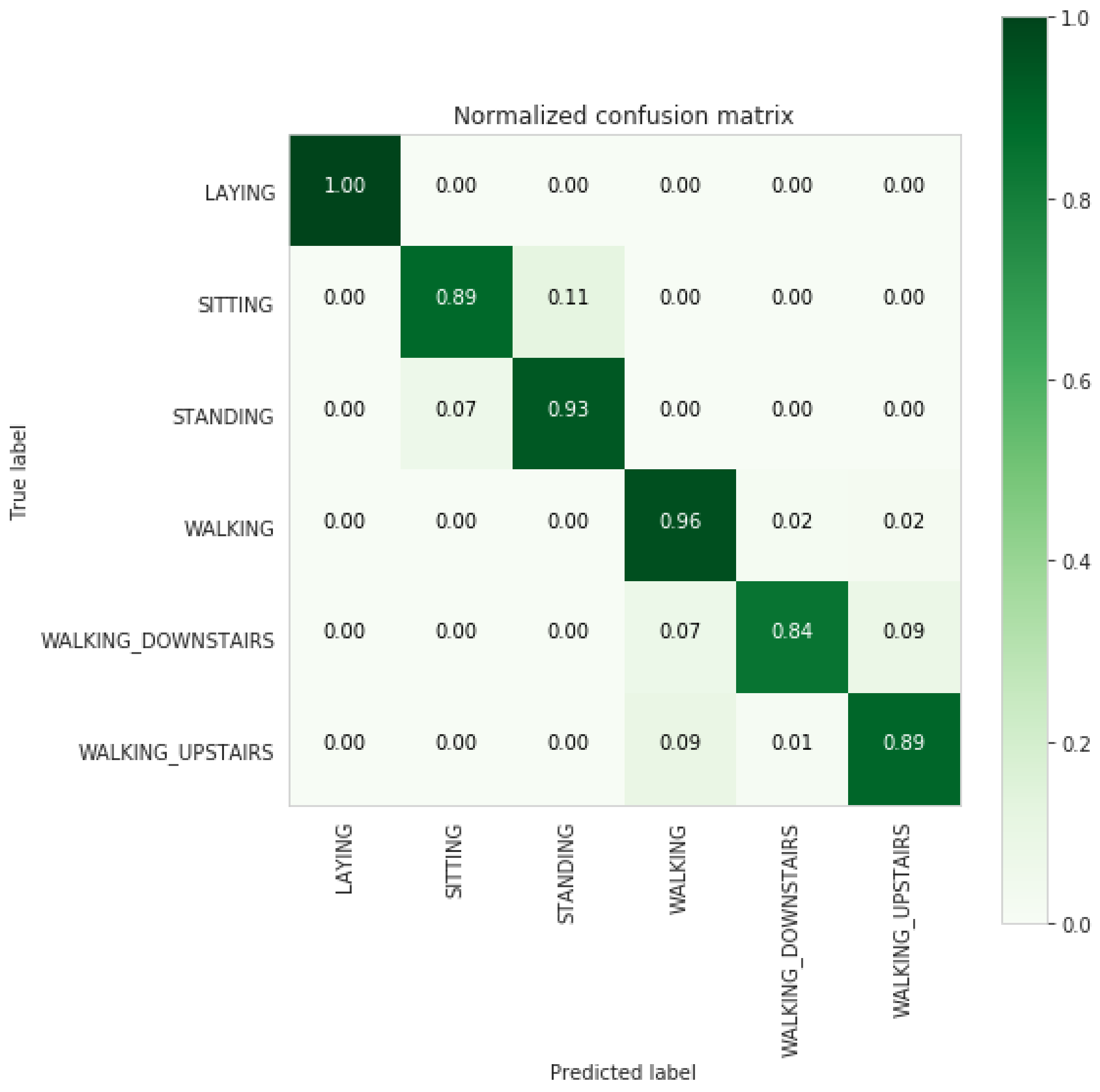
```
        Average Cross Validate scores of best estimator :

        0.8388193688792165
```

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

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:02:57.951732


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.034543


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

    0.9229725144214456
```

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

 [[537   0   0   0   0   0]
 [  0 435  56   0   0   0]
 [  0  35 497   0   0   0]
 [  0   0   0 478   8  10]
 [  0   0   0  31 353  36]
 [  0   0   0  44   7 420]]
```



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

            LAYING       1.00      1.00      1.00       537
```

```
           SITTING       0.93      0.89      0.91       491
          STANDING       0.90      0.93      0.92       532
           WALKING       0.86      0.96      0.91       496
WALKING_DOWNSTAIRS       0.96      0.84      0.90       420
  WALKING_UPSTAIRS       0.90      0.89      0.90       471

         micro avg       0.92      0.92      0.92      2947
         macro avg       0.92      0.92      0.92      2947
      weighted avg       0.92      0.92      0.92      2947

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

        RandomForestClassifier(bootstrap=True, class_weight=None, crite
rion='gini',
            max_depth=9, 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=50, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)

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

        {'n_estimators': 50, 'max_depth': 9}

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

        Total numbre of cross validation sets: 3

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

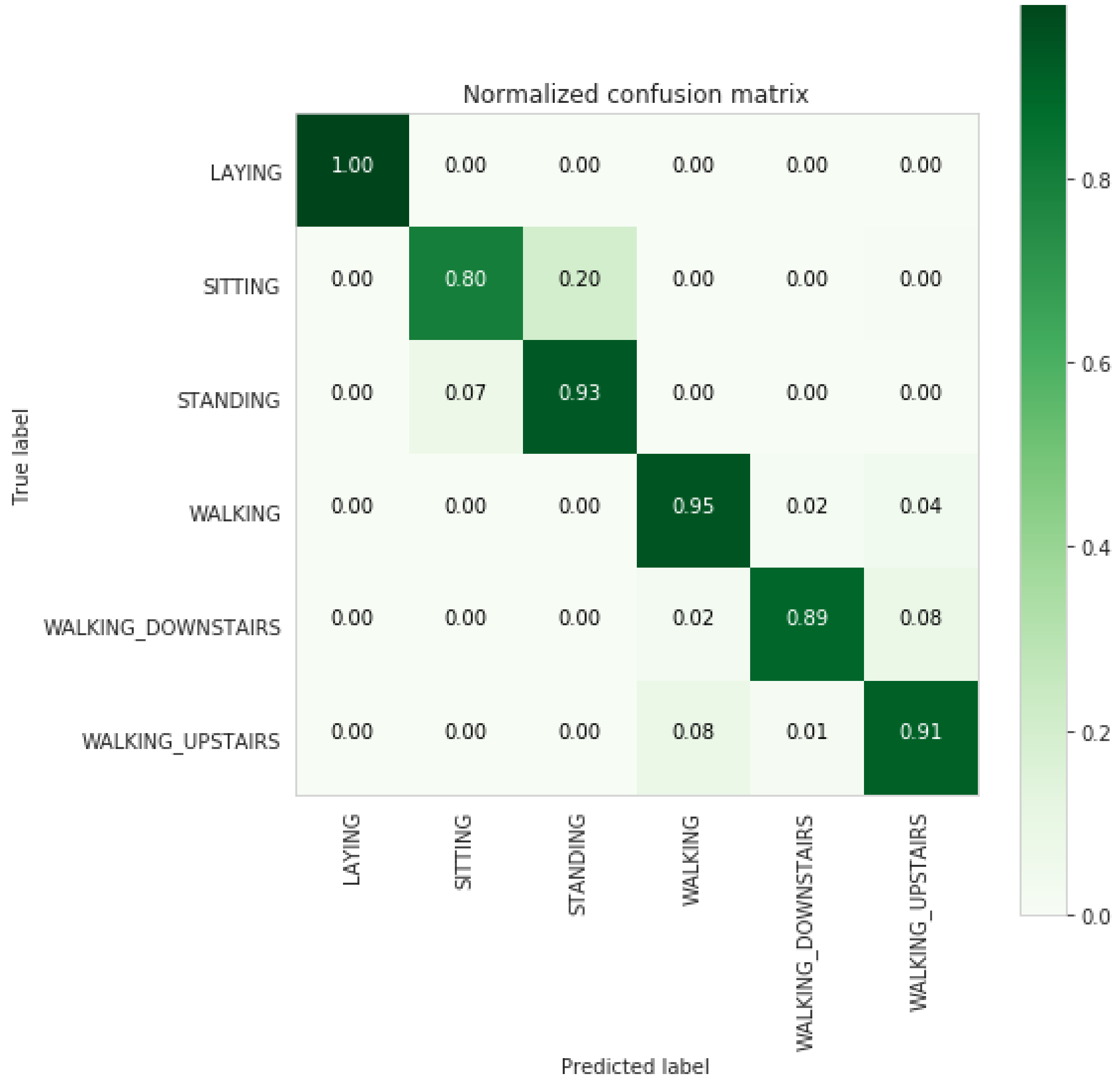
```
        Average Cross Validate scores of best estimator :

        0.9149891186071817
```

Gradient Boosted Decision Trees With GridSearch

In [44]:
```python
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)
print_grid_search_attributes(gbdt_grid_results['model'])
```

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:30:19.258358


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.077830


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

    0.9158466236851035
```

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

 [[537   0   0   0   0   0]
 [  0 393  96   0   0   2]
 [  0  37 495   0   0   0]
 [  0   0   0 470   8  18]
 [  0   0   0  10 375  35]
 [  0   1   0  37   4 429]]
```



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

            LAYING       1.00      1.00      1.00       537
           SITTING       0.91      0.80      0.85       491
```

```
          STANDING       0.84      0.93      0.88       532
           WALKING       0.91      0.95      0.93       496
WALKING_DOWNSTAIRS       0.97      0.89      0.93       420
  WALKING_UPSTAIRS       0.89      0.91      0.90       471

         micro avg       0.92      0.92      0.92      2947
         macro avg       0.92      0.91      0.91      2947
      weighted avg       0.92      0.92      0.92      2947

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

        GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=6,
              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 :

        {'n_estimators': 150, 'max_depth': 6}

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

        Total numbre of cross validation sets: 3

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

```
Average Cross Validate scores of best estimator :

0.904379760609358
```

In [45]:
```
print('\n                          Accuracy     Error')
print('                          ----------   --------')
print('Logistic Regression : {:.04}%       {:.04}%'.format(log_reg_grid
_results['accuracy'] * 100,\
                                                  100-(log_reg_grid_res
ults['accuracy'] * 100)))

print('Linear SVC          : {:.04}%       {:.04}% '.format(lr_svc_grid
_results['accuracy'] * 100,\
                                                        100-(lr_svc_gri
d_results['accuracy'] * 100)))

print('rbf SVM classifier  : {:.04}%      {:.04}% '.format(rbf_svm_grid
_results['accuracy'] * 100,\
                                                          100-(rbf_svm_
grid_results['accuracy'] * 100)))

print('DecisionTree        : {:.04}%      {:.04}% '.format(dt_grid_resu
lts['accuracy'] * 100,\
                                                        100-(dt_grid_re
sults['accuracy'] * 100)))

print('Random Forest       : {:.04}%      {:.04}% '.format(rfc_grid_res
ults['accuracy'] * 100,\
                                                           100-(rfc_gri
d_results['accuracy'] * 100)))
print('GradientBoosting DT : {:.04}%      {:.04}% '.format(rfc_grid_res
ults['accuracy'] * 100,\
                                                        100-(rfc_grid_r
esults['accuracy'] * 100)))
```

```
                          Accuracy     Error
                          ----------   --------
```

```
Logistic Regression : 96.3%        3.699%
Linear SVC          : 96.4%        3.597%
rbf SVM classifier  : 96.27%       3.733%
DecisionTree        : 86.49%       13.51%
Random Forest       : 92.3%       7.703%
GradientBoosting DT : 92.3%       7.703%
```

Insights and conclusion is provided in the Human Activity Recognition Part 2