Human Activity Recognition

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

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

How data was recorded

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

> prefix 't' in those metrics denotes time.

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

Feature names

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

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

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

    - tBodyAcc-XYZ
    - tGravityAcc-XYZ
    - tBodyAccJerk-XYZ
    - tBodyGyro-XYZ
    - tBodyGyroJerk-XYZ
    - tBodyAccMag

- tGravityAccMag
- tBodyAccJerkMag
- tBodyGyroMag
- tBodyGyroJerkMag
- fBodyAcc-XYZ
- fBodyAccJerk-XYZ
- fBodyGyro-XYZ
- fBodyAccMag
- fBodyAccJerkMag
- fBodyGyroMag
- fBodyGyroJerkMag

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

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

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

   - gravityMean
   - tBodyAccMean
   - tBodyAccJerkMean
   - tBodyGyroMean
   - tBodyGyroJerkMean

Y_Labels(Encoded)

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

Train and test data were saperated

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

Data

- All the data is present in 'UCI_HAR_dataset/' folder in present working directory.
  - Feature names are present in 'UCI_HAR_dataset/features.txt'
  - ***Train Data***
    - 'UCI_HAR_dataset/train/X_train.txt'
    - 'UCI_HAR_dataset/train/subject_train.txt'
    - 'UCI_HAR_dataset/train/y_train.txt'
  - ***Test Data***
    - 'UCI_HAR_dataset/test/X_test.txt'
    - 'UCI_HAR_dataset/test/subject_test.txt'
    - 'UCI_HAR_dataset/test/y_test.txt'

Data Size :

> 27 MB

Quick overview of the dataset :

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

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

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

Problem Framework

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

Problem Statement

- Given a new datapoint we have to predict the Activity

EDA

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
plt.rcParams['font.family'] = 'Dejavu Sans'
import warnings
warnings.filterwarnings("ignore")
import itertools

from sklearn.manifold import TSNE

# metrcis
from sklearn.metrics import confusion_matrix
from sklearn import metrics

# models
from sklearn import linear_model
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier

# datetime
from datetime import datetime
```

executed in 11.3s, finished 2018-11-06T12:36:31+05:30

In [2]:

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

executed in 23ms, finished 2018-11-06T12:36:43+05:30

```
No of Features: 561
```

Obtain the train data

In [3]:

```
# get the data from txt files to pandas dataffame
X_train = pd.read_csv('UCI_HAR_dataset/train/X_train.txt', delim_whitespace=True, head

# add subject column to the dataframe
X_train['subject'] = pd.read_csv('UCI_HAR_dataset/train/subject_train.txt', header=Non

y_train = pd.read_csv('UCI_HAR_dataset/train/y_train.txt', names=['Activity'], squeeze
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.head()
```

executed in 1.71s, finished 2018-11-06T12:36:46+05:30

Out[3]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y
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 [4]:

```
train.shape
```

executed in 5ms, finished 2018-11-06T12:36:46+05:30

Out[4]:

```
(7352, 564)
```

Obtain the test data

In [5]:

```
# get the data from txt files to pandas dataffame
X_test = pd.read_csv('UCI_HAR_dataset/test/X_test.txt', delim_whitespace=True, header=

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

# get y labels from the txt file
y_test = pd.read_csv('UCI_HAR_dataset/test/y_test.txt', names=['Activity'], squeeze=Tr
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.head()
```

executed in 725ms, finished 2018-11-06T12:37:11+05:30

Out[5]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y
0	0.257178	-0.023285	-0.014654	-0.938404	-0.920091	-0.667683	-0.952501	-0.925249
1	0.286027	-0.013163	-0.119083	-0.975415	-0.967458	-0.944958	-0.986799	-0.968401
2	0.275485	-0.026050	-0.118152	-0.993819	-0.969926	-0.962748	-0.994403	-0.970735
3	0.270298	-0.032614	-0.117520	-0.994743	-0.973268	-0.967091	-0.995274	-0.974471
4	0.274833	-0.027848	-0.129527	-0.993852	-0.967445	-0.978295	-0.994111	-0.965953

5 rows × 564 columns

In [6]:

```
test.shape
```

executed in 6ms, finished 2018-11-06T12:37:11+05:30

Out[6]:

```
(2947, 564)
```

Data Cleaning

1. Check for Duplicates

In [7]:

```
print("Numbers of duplicates in train data: {}".format(sum(train.duplicated())))
print("Numbers of duplicates in test data: {}".format(sum(test.duplicated())))

```

executed in 730ms, finished 2018-11-06T12:37:13+05:30

```
Numbers of duplicates in train data: 0
Numbers of duplicates in test data: 0
```

2. Checking for NaN/null values

In [8]:

```
print("Number of NaN value in train data is: {}".format(train.isnull().values.sum()))
print("Number of NaN value in test data is: {}".format(test.isnull().values.sum()))

```

executed in 34ms, finished 2018-11-06T12:37:13+05:30

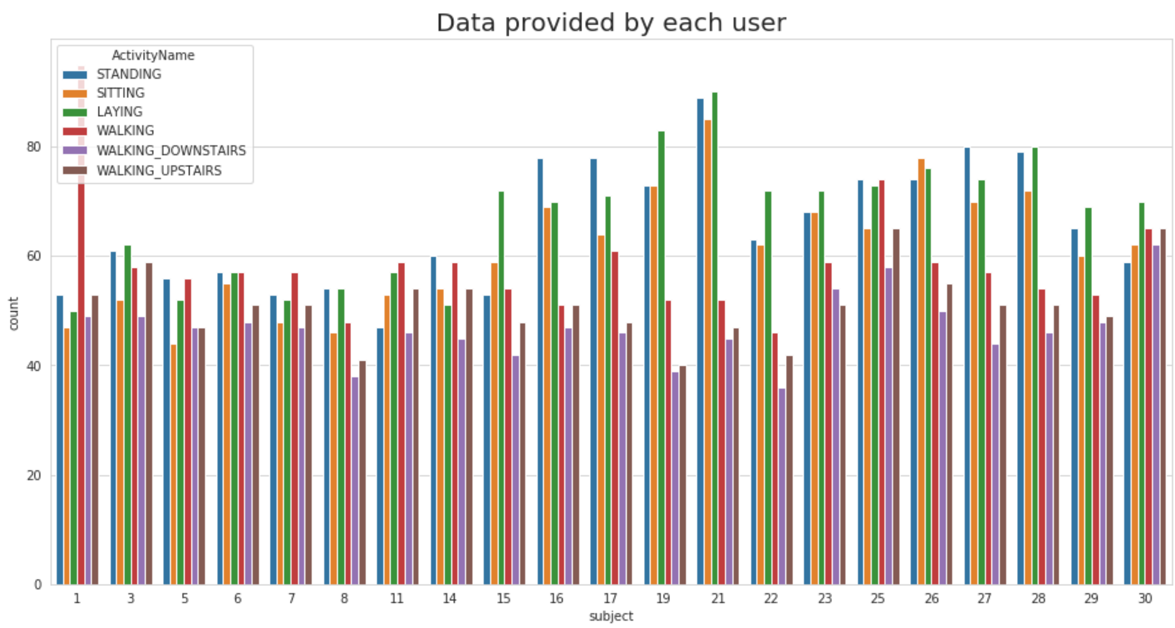
```
Number of NaN value in train data is: 0
Number of NaN value in test data is: 0
```

3. Check for data imbalance

In [9]:

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

executed in 572ms, finished 2018-11-06T12:37:14+05:30

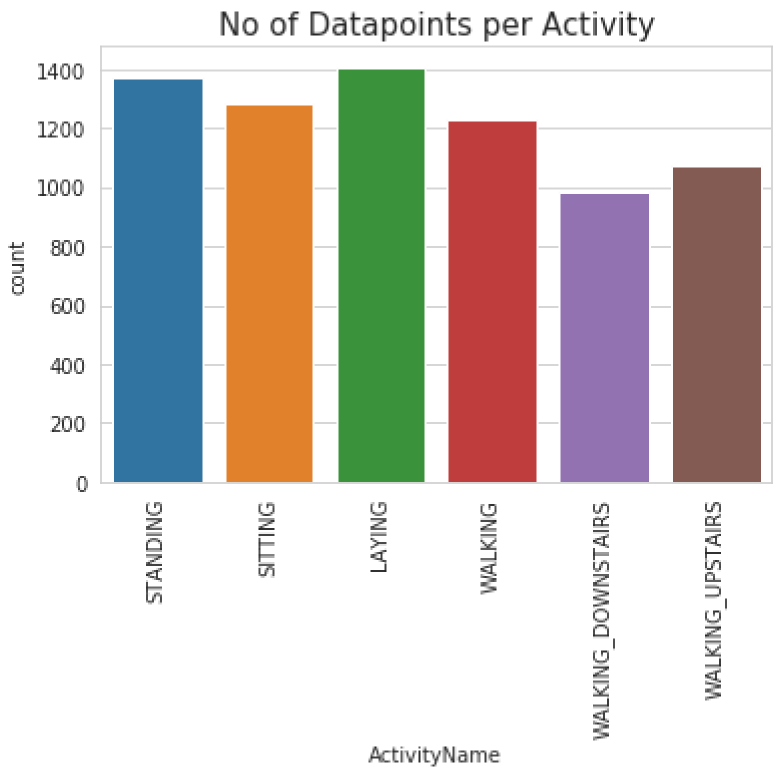


We have got almost same number of reading from all the subjects

In [10]:

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

executed in 146ms, finished 2018-11-06T12:37:15+05:30



Observation

Our data is well balanced (almost)

4. Changing feature names

In [11]:

```
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
```

executed in 11ms, finished 2018-11-06T12:37:16+05:30

Out[11]:

```
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 [12]:

```
train.to_csv('UCI_HAR_Dataset/csv_files/train.csv', index=False)
test.to_csv('UCI_HAR_Dataset/csv_files/test.csv', index=False)
```

executed in 8.58s, finished 2018-11-06T12:37:25+05:30

Exploratory Data Analysis

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

1. Featuring Engineering from Domain Knowledge

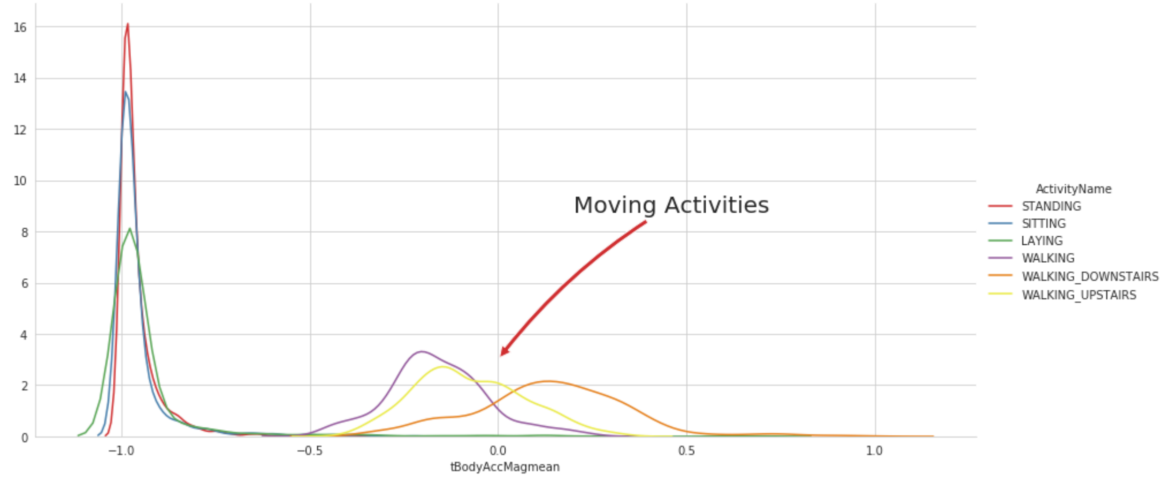
- **Static and Dynamic Activities**
  - In static activities (sit, stand, lie down) motion information will not be very useful.
  - In the dynamic activities (Walking, WalkingUpstairs,WalkingDownstairs) motion info will be significant.

2. Stationary and Moving activities are completely different

In [13]:

```
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()
```

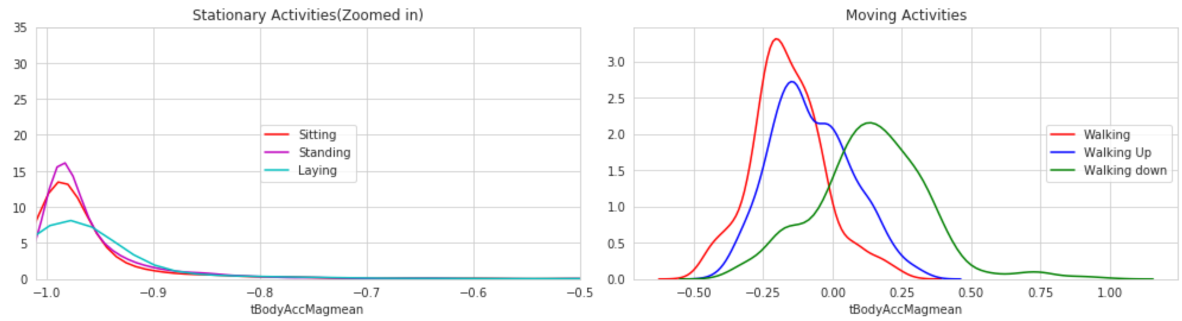
executed in 717ms, finished 2018-11-06T12:37:26+05:30



In [14]:

```
# 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 dow
plt.legend(loc='center right')


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

executed in 429ms, finished 2018-11-06T12:37:26+05:30

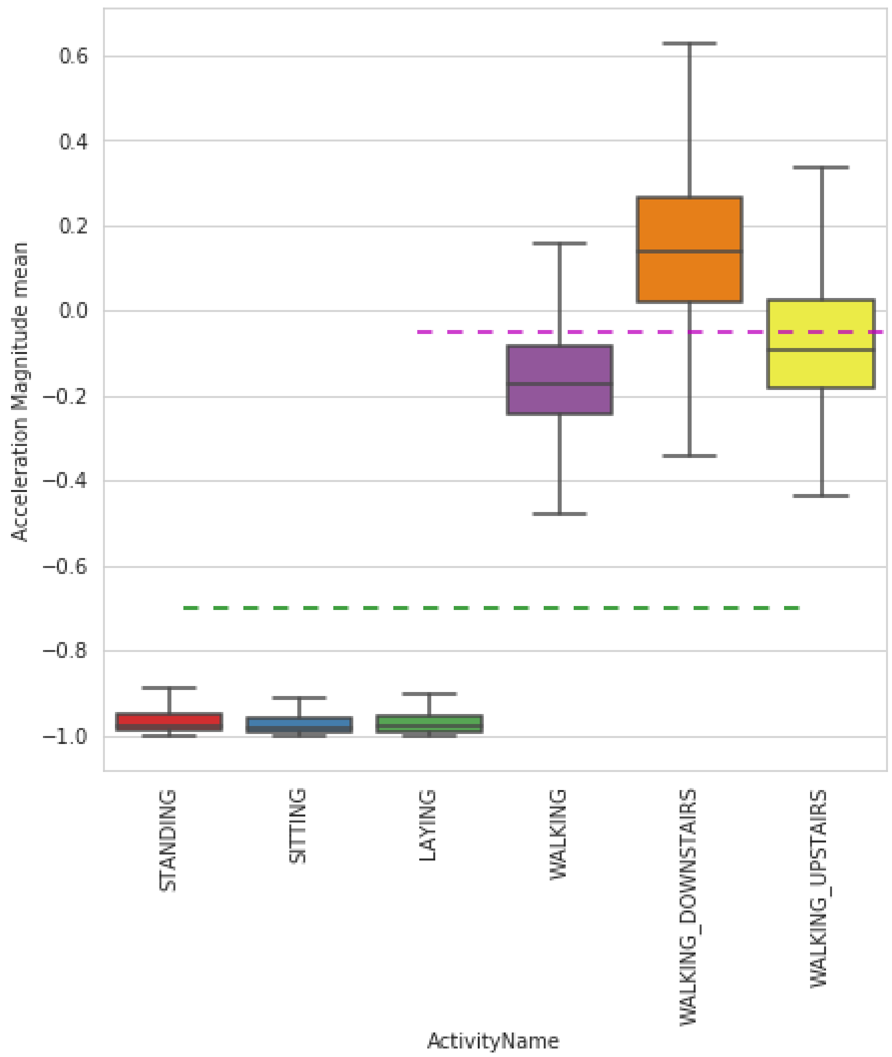


3. Magnitude of an acceleration can saperate it well

In [15]:

```
plt.figure(figsize=(7,7))
sns.boxplot(x='ActivityName', y='tBodyAccMagmean',data=train, showfliers=False, satura
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()
```

executed in 217ms, finished 2018-11-06T12:37:26+05:30



Observations:

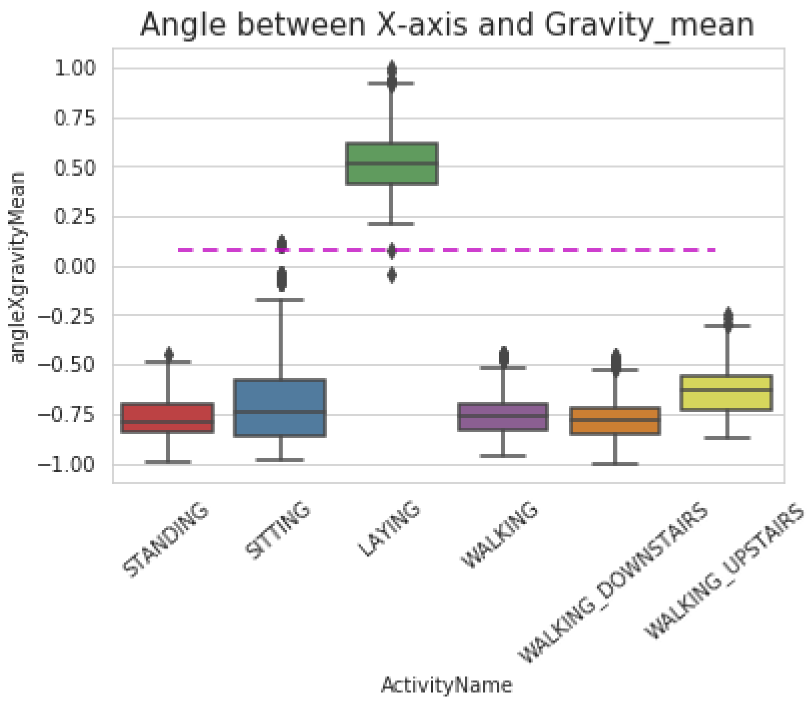
- If tAccMean is < -0.8 then the Activities are either Standing or Sitting or Laying.
- If tAccMean is > -0.6 then the Activities are either Walking or WalkingDownstairs or WalkingUpstairs.
- If tAccMean > 0.0 then the Activity is WalkingDownstairs.
- We can classify 75% the Acitivity labels with some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

4. Position of GravityAccelerationComponants also matters

In [16]:

```
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()
```

executed in 212ms, finished 2018-11-06T12:37:38+05:30



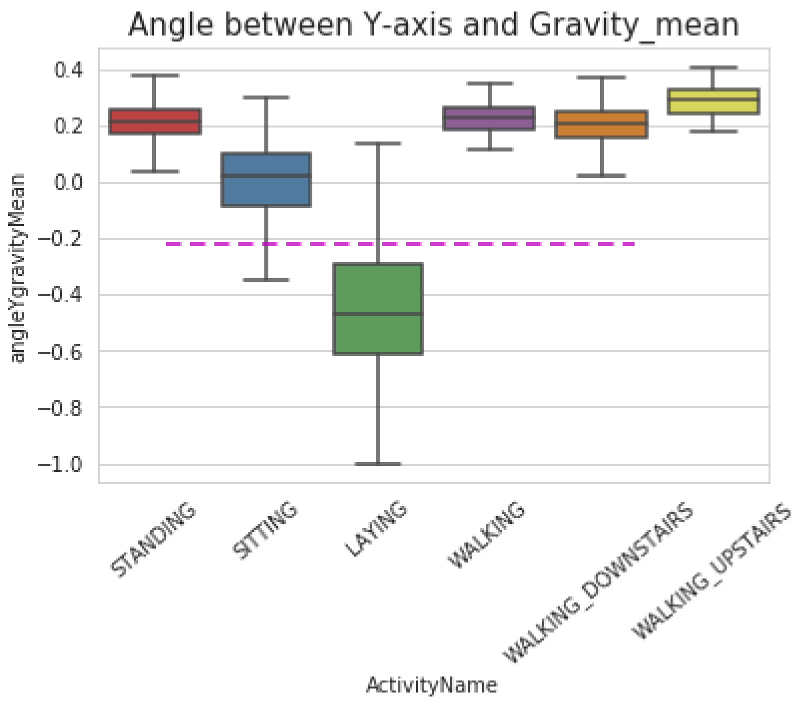
Observations:

- 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 [17]:

```
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()
```

executed in 247ms, finished 2018-11-06T12:37:38+05:30



Apply t-sne on the data

In [18]:

```
# 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'.for
        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')

```

executed in 7ms, finished 2018-11-06T12:37:39+05:30

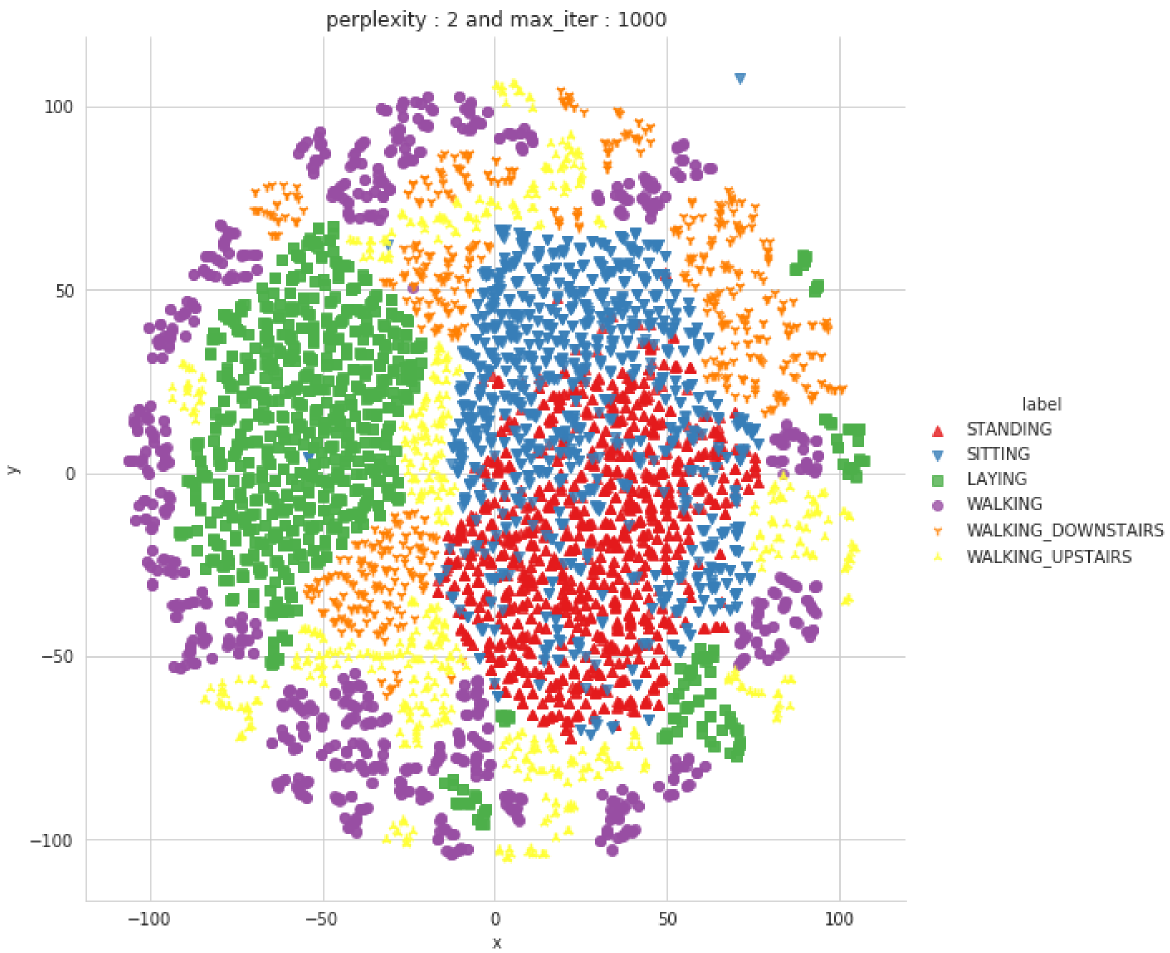
In [19]:

```
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])
```

executed in 20m 8s, finished 2018-11-06T12:57:47+05:30

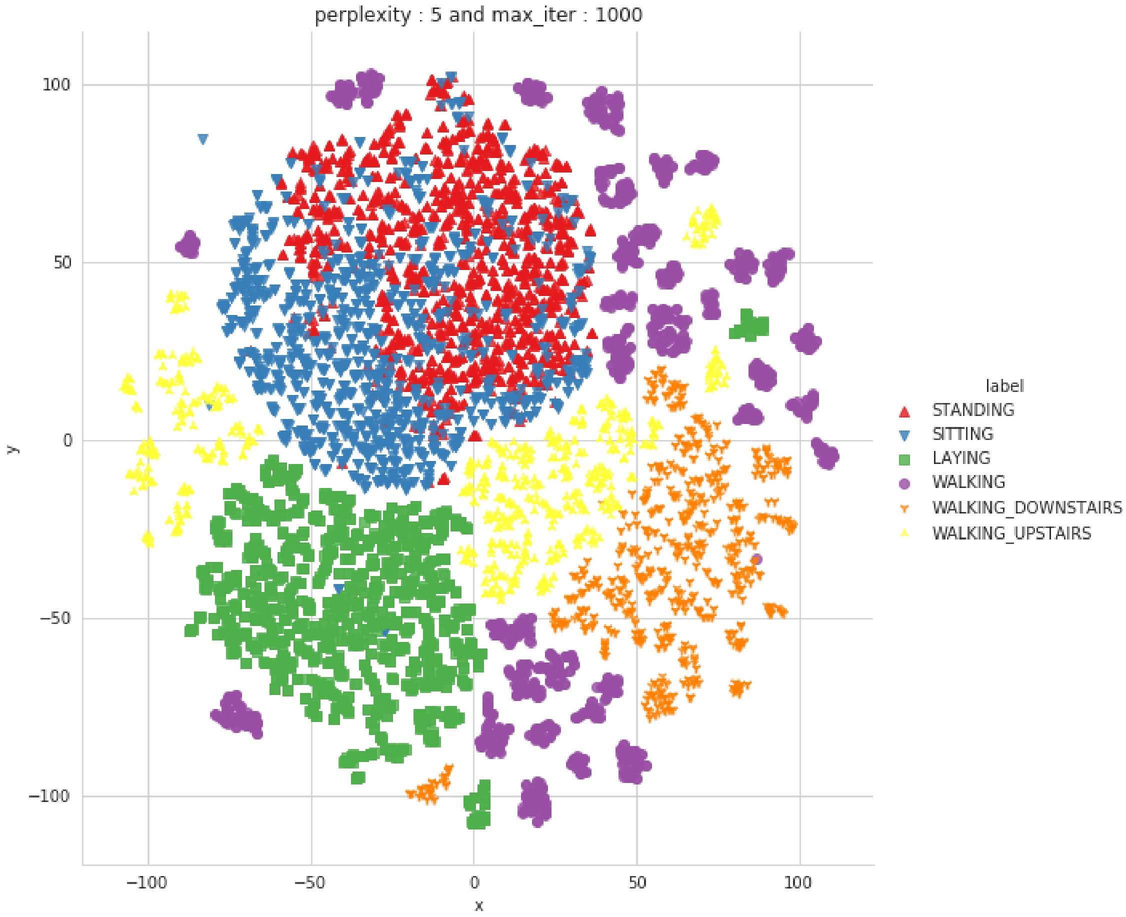
```
performing tsne with perplexity 2 and with 1000 iterations at max
[t-SNE] Computing 7 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.370s...
[t-SNE] Computed neighbors for 7352 samples in 50.562s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional 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 0.052s
[t-SNE] Iteration 50: error = 124.6932220, gradient norm = 0.0253406 (50 ite
rations in 10.903s)
[t-SNE] Iteration 100: error = 107.2629623, gradient norm = 0.0254685 (50 it
erations in 8.206s)
[t-SNE] Iteration 150: error = 101.0334396, gradient norm = 0.0174004 (50 it
erations in 6.906s)
[t-SNE] Iteration 200: error = 97.6049271, gradient norm = 0.0177322 (50 ite
rations in 6.263s)
[t-SNE] Iteration 250: error = 95.2795944, gradient norm = 0.0148403 (50 ite
rations in 6.261s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 95.27959
4
[t-SNE] Iteration 300: error = 4.1226091, gradient norm = 0.0015667 (50 iter
ations in 5.907s)
[t-SNE] Iteration 350: error = 3.2120683, gradient norm = 0.0010031 (50 iter
ations in 5.973s)
[t-SNE] Iteration 400: error = 2.7828777, gradient norm = 0.0007168 (50 iter
ations in 6.334s)
[t-SNE] Iteration 450: error = 2.5186248, gradient norm = 0.0005832 (50 iter
ations in 6.413s)
[t-SNE] Iteration 500: error = 2.3353820, gradient norm = 0.0004748 (50 iter
ations in 6.822s)
[t-SNE] Iteration 550: error = 2.1974690, gradient norm = 0.0004167 (50 iter
ations in 6.194s)
[t-SNE] Iteration 600: error = 2.0879962, gradient norm = 0.0003687 (50 iter
ations in 6.359s)
[t-SNE] Iteration 650: error = 1.9980706, gradient norm = 0.0003283 (50 iter
ations in 6.517s)
[t-SNE] Iteration 700: error = 1.9224700, gradient norm = 0.0003030 (50 iter
ations in 6.651s)
[t-SNE] Iteration 750: error = 1.8575609, gradient norm = 0.0002759 (50 iter
ations in 6.153s)
[t-SNE] Iteration 800: error = 1.8008357, gradient norm = 0.0002590 (50 iter
ations in 6.016s)
[t-SNE] Iteration 850: error = 1.7507610, gradient norm = 0.0002399 (50 iter
ations in 6.014s)
[t-SNE] Iteration 900: error = 1.7060949, gradient norm = 0.0002259 (50 iter
ations in 6.018s)
```

```
[t-SNE] Iteration 950: error = 1.6659936, gradient norm = 0.0002084 (50 iter
ations in 6.026s)
[t-SNE] Iteration 1000: error = 1.6296664, gradient norm = 0.0001995 (50 ite
rations in 6.067s)
[t-SNE] Error after 1000 iterations: 1.629666
Done..
Creating plot for this t-sne visualization..
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 0.274s...
[t-SNE] Computed neighbors for 7352 samples in 51.031s...
[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 0.063s
[t-SNE] Iteration 50: error = 113.9096527, gradient norm = 0.0217785 (50 i
terations in 11.809s)
[t-SNE] Iteration 100: error = 97.6285477, gradient norm = 0.0153195 (50 i
terations in 7.652s)
[t-SNE] Iteration 150: error = 93.0926895, gradient norm = 0.0088957 (50 i
terations in 6.426s)
[t-SNE] Iteration 200: error = 91.1182632, gradient norm = 0.0069188 (50 i
```

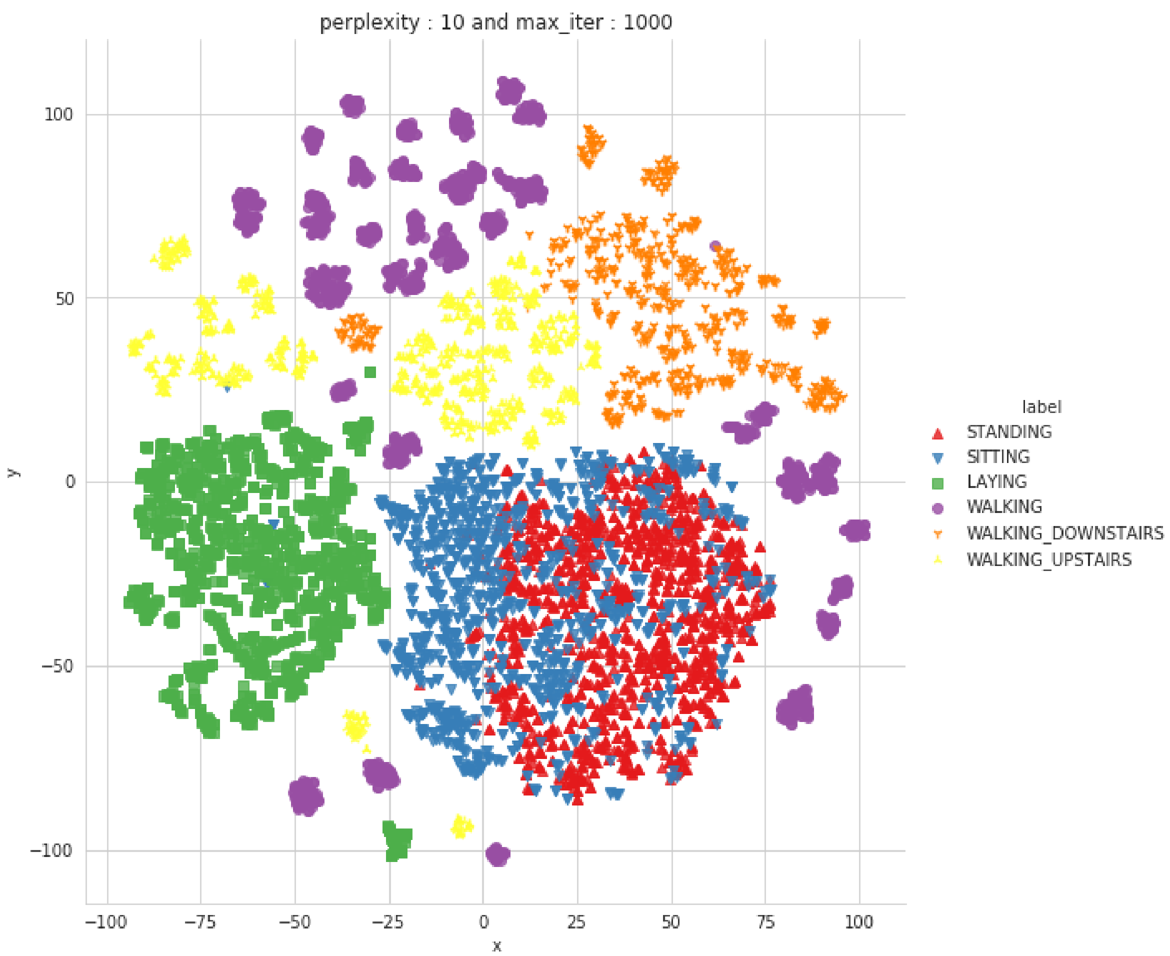
```
terations in 6.268s)
[t-SNE] Iteration 250: error = 89.9500961, gradient norm = 0.0051981 (50 i
terations in 6.252s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 89.950
096
[t-SNE] Iteration 300: error = 3.5718865, gradient norm = 0.0014644 (50 it
erations in 6.283s)
[t-SNE] Iteration 350: error = 2.8142099, gradient norm = 0.0007516 (50 it
erations in 6.396s)
[t-SNE] Iteration 400: error = 2.4338355, gradient norm = 0.0005309 (50 it
erations in 6.495s)
[t-SNE] Iteration 450: error = 2.2165146, gradient norm = 0.0004018 (50 it
erations in 6.552s)
[t-SNE] Iteration 500: error = 2.0715866, gradient norm = 0.0003355 (50 it
erations in 6.558s)
[t-SNE] Iteration 550: error = 1.9666973, gradient norm = 0.0002838 (50 it
erations in 6.609s)
[t-SNE] Iteration 600: error = 1.8857353, gradient norm = 0.0002474 (50 it
erations in 6.605s)
[t-SNE] Iteration 650: error = 1.8206962, gradient norm = 0.0002208 (50 it
erations in 6.649s)
[t-SNE] Iteration 700: error = 1.7669537, gradient norm = 0.0001977 (50 it
erations in 6.616s)
[t-SNE] Iteration 750: error = 1.7216936, gradient norm = 0.0001815 (50 it
erations in 6.631s)
[t-SNE] Iteration 800: error = 1.6828806, gradient norm = 0.0001667 (50 it
erations in 6.635s)
[t-SNE] Iteration 850: error = 1.6491964, gradient norm = 0.0001514 (50 it
erations in 6.630s)
[t-SNE] Iteration 900: error = 1.6195487, gradient norm = 0.0001409 (50 it
erations in 6.698s)
[t-SNE] Iteration 950: error = 1.5930126, gradient norm = 0.0001332 (50 it
erations in 6.724s)
[t-SNE] Iteration 1000: error = 1.5691556, gradient norm = 0.0001258 (50 i
terations in 6.629s)
[t-SNE] Error after 1000 iterations: 1.569156
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```



Done

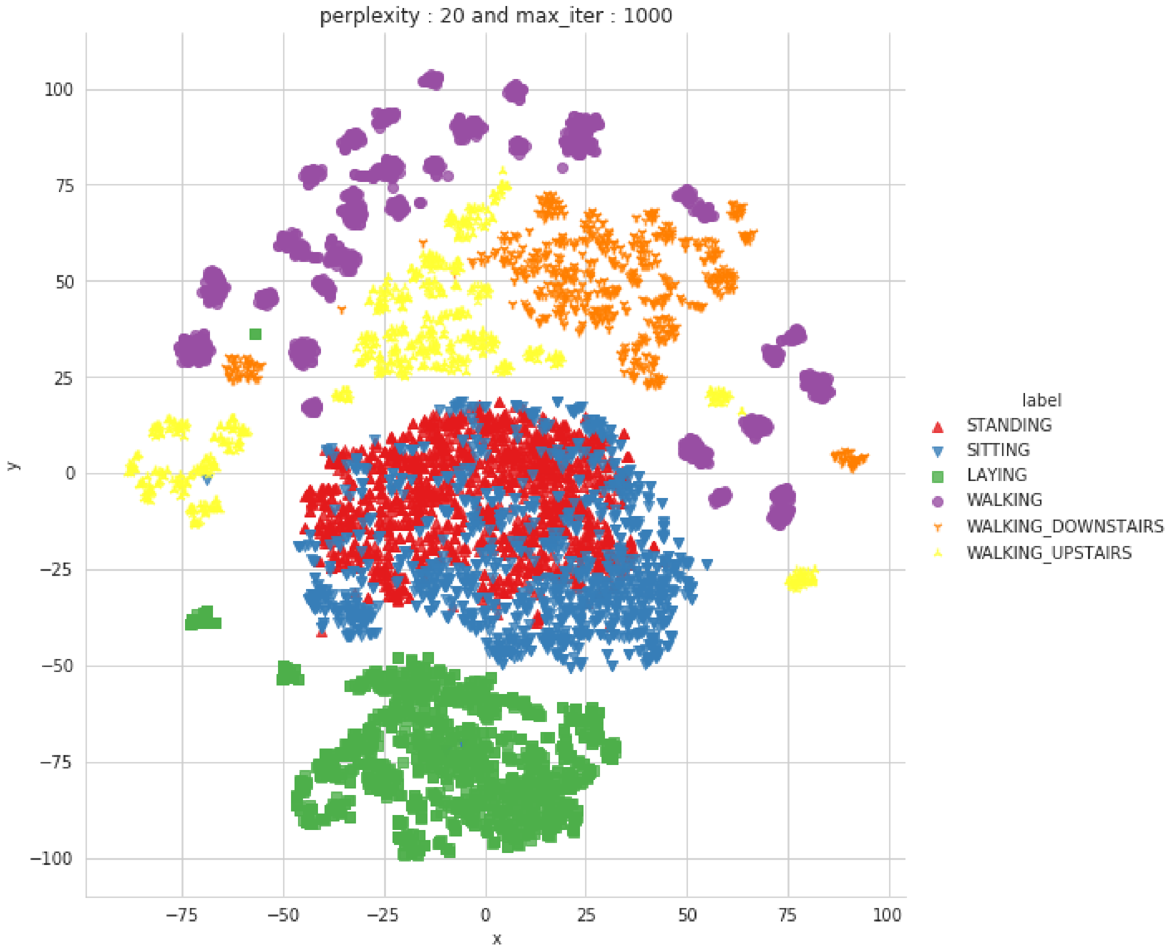
```
performing tsne with perplexity 10 and with 1000 iterations at max
[t-SNE] Computing 31 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.291s...
[t-SNE] Computed neighbors for 7352 samples in 52.395s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional 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 0.119s
[t-SNE] Iteration 50: error = 105.7489166, gradient norm = 0.0194669 (50 ite
rations in 11.964s)
[t-SNE] Iteration 100: error = 90.8696899, gradient norm = 0.0107236 (50 ite
rations in 8.564s)
[t-SNE] Iteration 150: error = 87.5438080, gradient norm = 0.0053936 (50 ite
rations in 7.625s)
[t-SNE] Iteration 200: error = 86.2203979, gradient norm = 0.0045790 (50 ite
rations in 7.592s)
[t-SNE] Iteration 250: error = 85.4012451, gradient norm = 0.0034881 (50 ite
rations in 7.615s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.40124
5
[t-SNE] Iteration 300: error = 3.1326840, gradient norm = 0.0013898 (50 iter
ations in 7.687s)
[t-SNE] Iteration 350: error = 2.4882658, gradient norm = 0.0006501 (50 iter
ations in 7.527s)
[t-SNE] Iteration 400: error = 2.1683755, gradient norm = 0.0004237 (50 iter
```

```
ations in 7.647s)
[t-SNE] Iteration 450: error = 1.9845148, gradient norm = 0.0003113 (50 iter
ations in 7.768s)
[t-SNE] Iteration 500: error = 1.8665744, gradient norm = 0.0002517 (50 iter
ations in 7.520s)
[t-SNE] Iteration 550: error = 1.7830403, gradient norm = 0.0002150 (50 iter
ations in 7.583s)
[t-SNE] Iteration 600: error = 1.7208303, gradient norm = 0.0001812 (50 iter
ations in 7.826s)
[t-SNE] Iteration 650: error = 1.6717302, gradient norm = 0.0001590 (50 iter
ations in 7.879s)
[t-SNE] Iteration 700: error = 1.6321598, gradient norm = 0.0001432 (50 iter
ations in 7.814s)
[t-SNE] Iteration 750: error = 1.5993425, gradient norm = 0.0001292 (50 iter
ations in 7.947s)
[t-SNE] Iteration 800: error = 1.5718443, gradient norm = 0.0001180 (50 iter
ations in 7.683s)
[t-SNE] Iteration 850: error = 1.5483818, gradient norm = 0.0001091 (50 iter
ations in 7.583s)
[t-SNE] Iteration 900: error = 1.5279934, gradient norm = 0.0001017 (50 iter
ations in 7.553s)
[t-SNE] Iteration 950: error = 1.5102819, gradient norm = 0.0000947 (50 iter
ations in 7.350s)
[t-SNE] Iteration 1000: error = 1.4947616, gradient norm = 0.0000913 (50 ite
rations in 7.402s)
[t-SNE] Error after 1000 iterations: 1.494762
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```



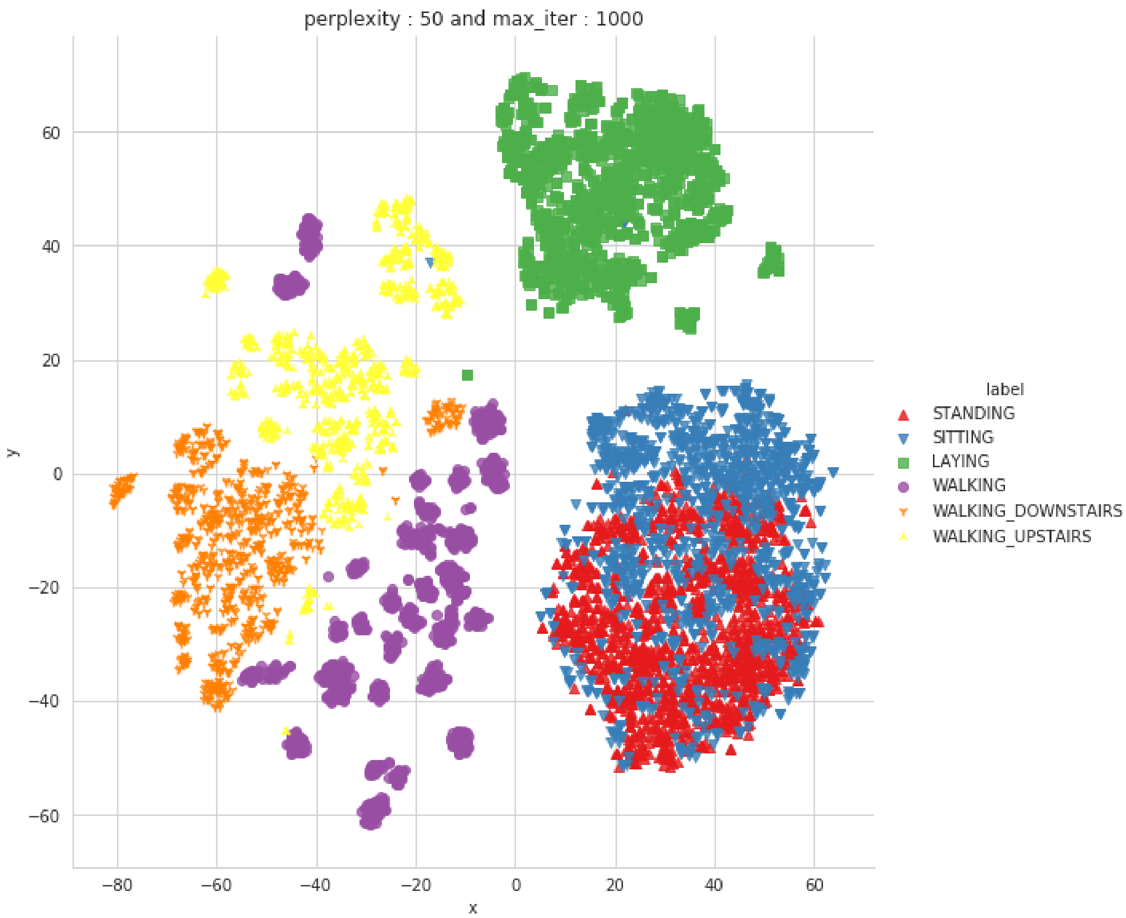
Done

```
performing tsne with perplexity 20 and with 1000 iterations at max
[t-SNE] Computing 61 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.257s...
[t-SNE] Computed neighbors for 7352 samples in 49.075s...
[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 0.214s
[t-SNE] Iteration 50: error = 96.5512390, gradient norm = 0.0274330 (50 it
erations in 15.430s)
[t-SNE] Iteration 100: error = 84.4019012, gradient norm = 0.0067155 (50 i
terations in 10.704s)
[t-SNE] Iteration 150: error = 82.2217407, gradient norm = 0.0033478 (50 i
terations in 9.706s)
[t-SNE] Iteration 200: error = 81.4570160, gradient norm = 0.0026197 (50 i
terations in 10.049s)
[t-SNE] Iteration 250: error = 81.0493698, gradient norm = 0.0020515 (50 i
terations in 10.646s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 81.049
370
[t-SNE] Iteration 300: error = 2.7297406, gradient norm = 0.0012999 (50 it
erations in 10.732s)
[t-SNE] Iteration 350: error = 2.1957550, gradient norm = 0.0005851 (50 it
erations in 10.156s)
[t-SNE] Iteration 400: error = 1.9434785, gradient norm = 0.0003512 (50 it
erations in 10.193s)
[t-SNE] Iteration 450: error = 1.7956692, gradient norm = 0.0002500 (50 it
erations in 10.736s)
[t-SNE] Iteration 500: error = 1.7011566, gradient norm = 0.0001953 (50 it
erations in 10.089s)
[t-SNE] Iteration 550: error = 1.6364098, gradient norm = 0.0001598 (50 it
erations in 9.766s)
[t-SNE] Iteration 600: error = 1.5893759, gradient norm = 0.0001361 (50 it
erations in 10.348s)
[t-SNE] Iteration 650: error = 1.5534849, gradient norm = 0.0001198 (50 it
erations in 9.686s)
[t-SNE] Iteration 700: error = 1.5254400, gradient norm = 0.0001061 (50 it
erations in 9.436s)
[t-SNE] Iteration 750: error = 1.5031372, gradient norm = 0.0000981 (50 it
erations in 9.695s)
[t-SNE] Iteration 800: error = 1.4852676, gradient norm = 0.0000921 (50 it
erations in 9.510s)
[t-SNE] Iteration 850: error = 1.4706868, gradient norm = 0.0000830 (50 it
erations in 9.773s)
[t-SNE] Iteration 900: error = 1.4582170, gradient norm = 0.0000786 (50 it
erations in 9.832s)
[t-SNE] Iteration 950: error = 1.4474396, gradient norm = 0.0000759 (50 it
erations in 9.507s)
[t-SNE] Iteration 1000: error = 1.4382648, gradient norm = 0.0000715 (50 i
terations in 9.489s)
[t-SNE] Error after 1000 iterations: 1.438265
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```




```
Done

performing tsne with perplexity 50 and with 1000 iterations at max
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.266s...
[t-SNE] Computed neighbors for 7352 samples in 50.699s...
[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 0.524s
[t-SNE] Iteration 50: error = 86.4923019, gradient norm = 0.0199595 (50 iter
ations in 20.173s)
[t-SNE] Iteration 100: error = 75.7472382, gradient norm = 0.0041427 (50 ite
rations in 17.551s)
[t-SNE] Iteration 150: error = 74.6251755, gradient norm = 0.0026428 (50 ite
rations in 15.963s)
[t-SNE] Iteration 200: error = 74.2391052, gradient norm = 0.0016393 (50 ite
rations in 16.253s)
[t-SNE] Iteration 250: error = 74.0540009, gradient norm = 0.0012340 (50 ite
rations in 16.277s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 74.05400
1
[t-SNE] Iteration 300: error = 2.1543148, gradient norm = 0.0011814 (50 iter
ations in 15.784s)
[t-SNE] Iteration 350: error = 1.7571723, gradient norm = 0.0004904 (50 iter
ations in 15.240s)
```

```
[t-SNE] Iteration 400: error = 1.5893952, gradient norm = 0.0002822 (50 iter
ations in 15.395s)
[t-SNE] Iteration 450: error = 1.4956979, gradient norm = 0.0001895 (50 iter
ations in 15.630s)
[t-SNE] Iteration 500: error = 1.4357437, gradient norm = 0.0001392 (50 iter
ations in 14.982s)
[t-SNE] Iteration 550: error = 1.3947712, gradient norm = 0.0001133 (50 iter
ations in 15.455s)
[t-SNE] Iteration 600: error = 1.3655816, gradient norm = 0.0000931 (50 iter
ations in 15.114s)
[t-SNE] Iteration 650: error = 1.3443761, gradient norm = 0.0000809 (50 iter
ations in 15.209s)
[t-SNE] Iteration 700: error = 1.3287190, gradient norm = 0.0000764 (50 iter
ations in 15.146s)
[t-SNE] Iteration 750: error = 1.3174671, gradient norm = 0.0000734 (50 iter
ations in 15.303s)
[t-SNE] Iteration 800: error = 1.3087639, gradient norm = 0.0000645 (50 iter
ations in 15.489s)
[t-SNE] Iteration 850: error = 1.3019392, gradient norm = 0.0000596 (50 iter
ations in 15.271s)
[t-SNE] Iteration 900: error = 1.2962723, gradient norm = 0.0000577 (50 iter
ations in 15.304s)
[t-SNE] Iteration 950: error = 1.2914181, gradient norm = 0.0000546 (50 iter
ations in 15.051s)
[t-SNE] Iteration 1000: error = 1.2872577, gradient norm = 0.0000536 (50 ite
rations in 14.988s)
[t-SNE] Error after 1000 iterations: 1.287258
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```



Done

Obtain the train and test data

In [20]:

```
train = pd.read_csv('UCI_HAR_dataset/csv_files/train.csv')
test = pd.read_csv('UCI_HAR_dataset/csv_files/test.csv')
print("Shape of train data",train.shape)
print("Shape of test data", test.shape)

```

executed in 1.96s, finished 2018-11-06T12:57:49+05:30

```
Shape of train data (7352, 564)
Shape of test data (2947, 564)
```

In [21]:

```
train.head(2)
```

executed in 24ms, finished 2018-11-06T12:57:49+05:30

Out[21]:

	tBodyAccmeanX	tBodyAccmeanY	tBodyAccmeanZ	tBodyAccstdX	tBodyAccstdY	tBodyAccs
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960

2 rows × 564 columns

In [22]:

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

executed in 27ms, finished 2018-11-06T12:57:49+05:30

In [23]:

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

executed in 27ms, finished 2018-11-06T12:57:49+05:30

In [24]:

```
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))
```

executed in 17ms, finished 2018-11-06T12:57:49+05:30

```
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 [25]:

```
labels=['LAYING', 'SITTING','STANDING','WALKING','WALKING_DOWNSTAIRS','WALKING_UPSTAIR
```

executed in 16ms, finished 2018-11-06T12:57:49+05:30

Function to plot the confusion matrix

In [26]:

```
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')
```

executed in 7ms, finished 2018-11-06T12:58:21+05:30

Generic function to run any model specified

In [27]:

```
from datetime import datetime
def perform_model(model, X_train, y_train, X_test, y_test, class_labels, cm_normalize=
                 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 
    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


```

executed in 9ms, finished 2018-11-06T12:58:22+05:30

In [28]:

```
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(mo



```

executed in 5ms, finished 2018-11-06T12:58:27+05:30

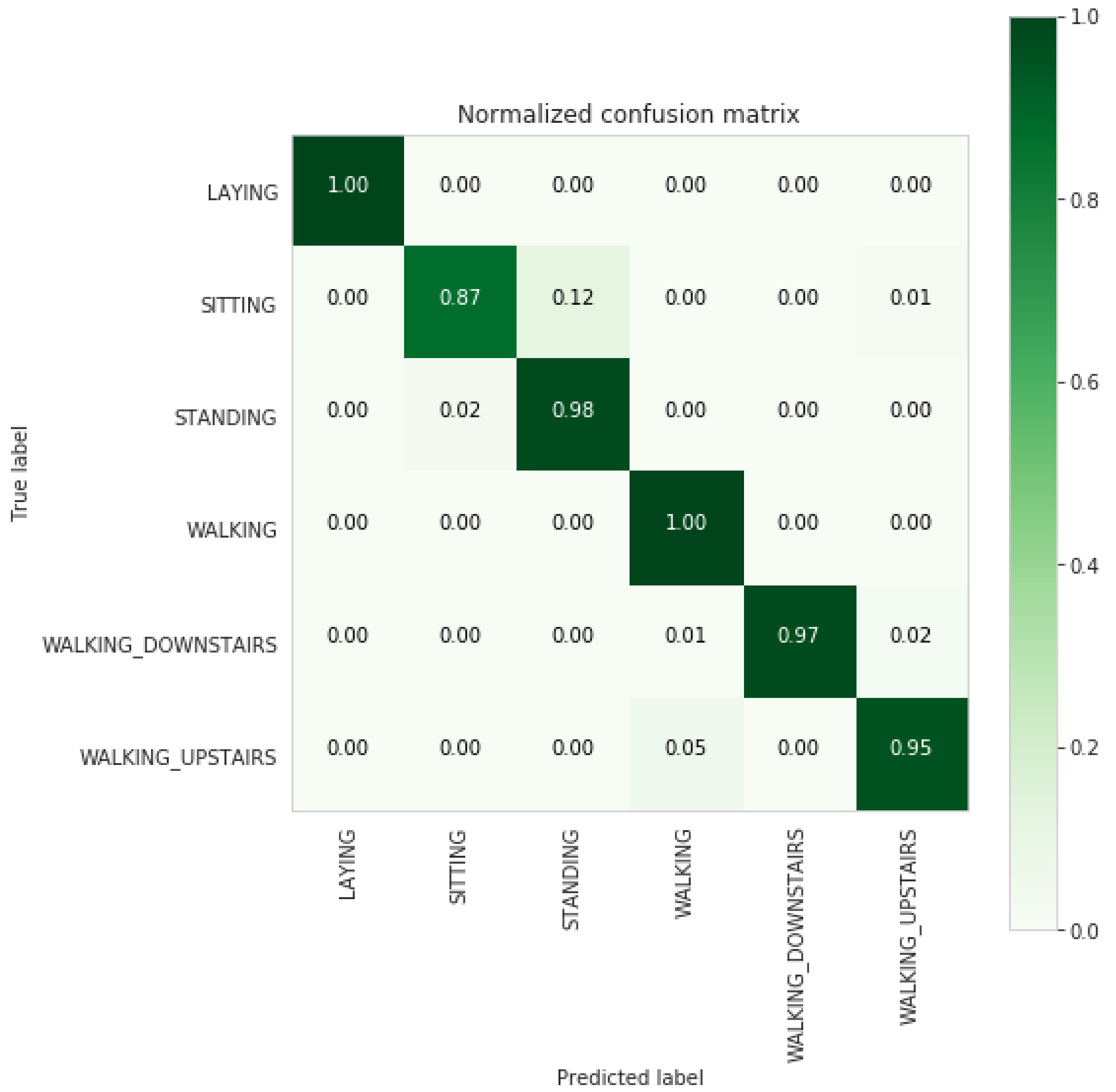
1. Logistic Regression with Grid Search

In [29]:

```
parameters = {'C':[0.01, 0.1, 1, 10, 20, 30], 'penalty':['l2','l1']}
log_reg = LogisticRegression()
log_reg_grid = GridSearchCV(log_reg,param_grid=parameters,cv=3,verbose=1,n_jobs=1)
log_reg_grid_result = perform_model(log_reg_grid,X_train,y_train,X_test,y_test,class_l
```

executed in 2m 58s, finished 2018-11-06T13:01:28+05:30

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

[Parallel(n_jobs=1)]: Done  36 out of  36 | elapsed:  2.8min finished

Done


training_time(HH:MM:SS.ms) - 0:02:57.863303


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.007976


---------------------
|      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

       avg / total       0.96      0.96      0.96      2947
```

In [30]:

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

executed in 5ms, finished 2018-11-06T13:01:28+05:30

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

        LogisticRegression(C=30, class_weight=None, dual=False, fit_intercep
t=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', 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
```

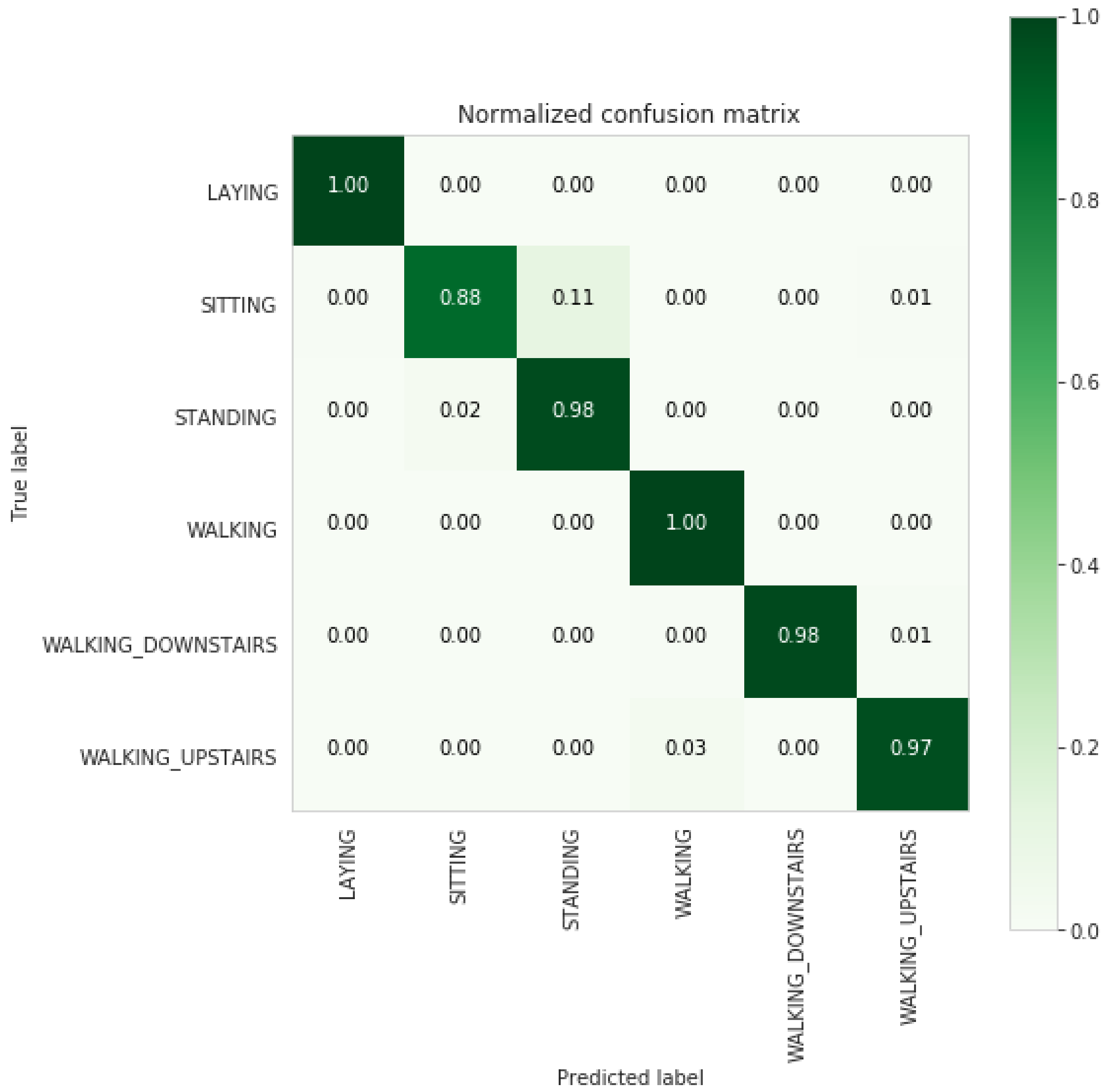
2. Linear SVC with GridSearch

In [31]:

```
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, cla
```

executed in 35.8s, finished 2018-11-06T13:02:04+05:30

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

[Parallel(n_jobs=-1)]: Done  18 out of  18 | elapsed:   27.6s finished

Done


training_time(HH:MM:SS.ms) - 0:00:35.128914


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.019948


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

    0.9681031557516118


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

 [[537   0   0   0   0   0]
 [  2 432  54   0   0   3]
 [  0  12 519   1   0   0]
 [  0   0   0 496   0   0]
 [  0   0   0   2 413   5]
 [  0   0   0  14   1 456]]
```




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

            LAYING       1.00      1.00      1.00       537
           SITTING       0.97      0.88      0.92       491
          STANDING       0.91      0.98      0.94       532
           WALKING       0.97      1.00      0.98       496
WALKING_DOWNSTAIRS       1.00      0.98      0.99       420
  WALKING_UPSTAIRS       0.98      0.97      0.98       471

       avg / total       0.97      0.97      0.97      2947
```

In [32]:

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

executed in 10ms, finished 2018-11-06T13:02:04+05:30

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

        LinearSVC(C=2, 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': 2}

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator : 

        0.9460010881392819
```

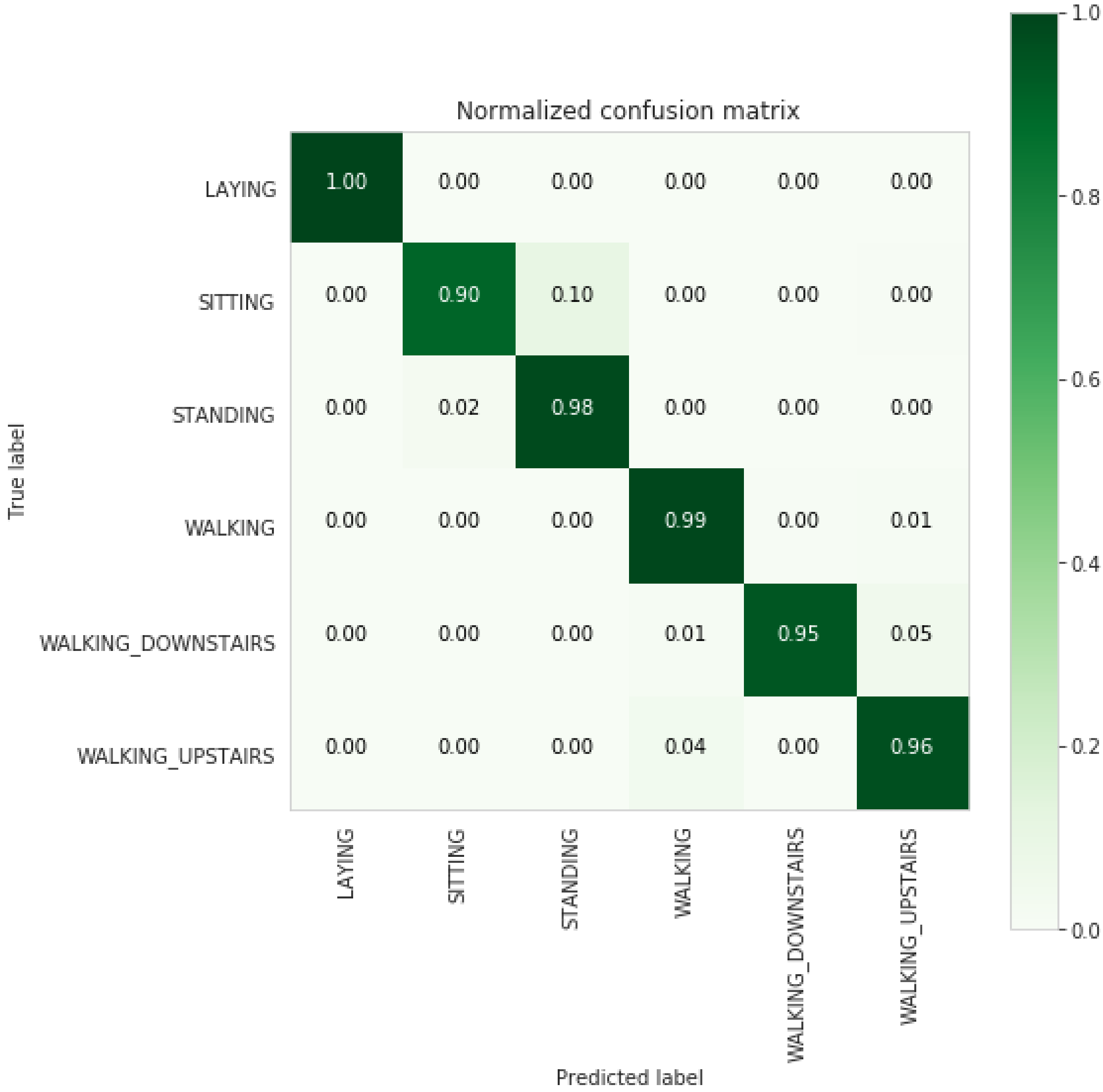
3. Kernel SVM with GridSearch

In [33]:

```
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, c
```

executed in 7m 12s, finished 2018-11-06T13:09:16+05:30

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:07:08.442032


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:03.142625


---------------------
|      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

       avg / total       0.96      0.96      0.96      2947
```

In [34]:

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

executed in 4ms, finished 2018-11-06T13:09:16+05:30

```
--------------------------
|      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 [35]:

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

executed in 12.9s, finished 2018-11-06T13:14:53+05:30

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:00:12.544452


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.008011


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

    0.8639294197488971


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

 [[537   0   0   0   0   0]
 [  0 385 106   0   0   0]
 [  0  93 439   0   0   0]
 [  0   0   0 472  16   8]
 [  0   0   0  15 344  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      0.78      0.79       491
          STANDING       0.81      0.83      0.82       532
           WALKING       0.84      0.95      0.89       496
WALKING_DOWNSTAIRS       0.90      0.82      0.86       420
  WALKING_UPSTAIRS       0.84      0.78      0.81       471

       avg / total       0.86      0.86      0.86      2947

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

        DecisionTreeClassifier(class_weight=None, criterion='gini', max_dept
h=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 : 

        0.8363710554951034
```

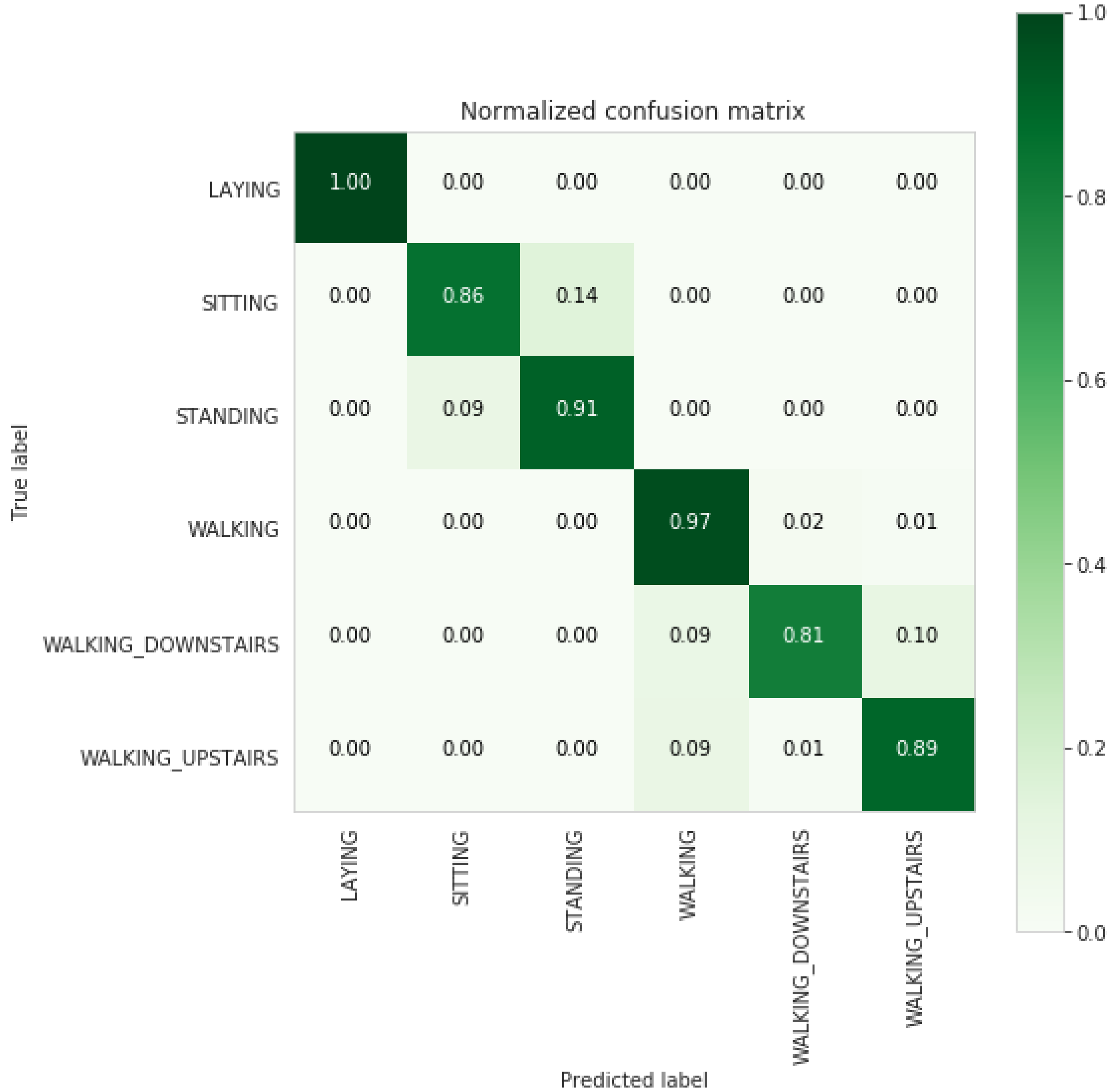
5. Random Forest Classifier with GridSearch

In [36]:

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

executed in 5m 49s, finished 2018-11-06T13:20:43+05:30

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:05:48.879640


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.057847


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

    0.9100780454699695


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

 [[537   0   0   0   0   0]
 [  0 420  71   0   0   0]
 [  0  48 484   0   0   0]
 [  0   0   0 481  10   5]
 [  0   0   0  37 339  44]
 [  0   0   0  44   6 421]]
```




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

            LAYING       1.00      1.00      1.00       537
           SITTING       0.90      0.86      0.88       491
          STANDING       0.87      0.91      0.89       532
           WALKING       0.86      0.97      0.91       496
WALKING_DOWNSTAIRS       0.95      0.81      0.87       420
  WALKING_UPSTAIRS       0.90      0.89      0.89       471

       avg / total       0.91      0.91      0.91      2947

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

        RandomForestClassifier(bootstrap=True, class_weight=None, criterio
n='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=130, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

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

        {'max_depth': 7, 'n_estimators': 130}

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator : 

        0.9144450489662677
```

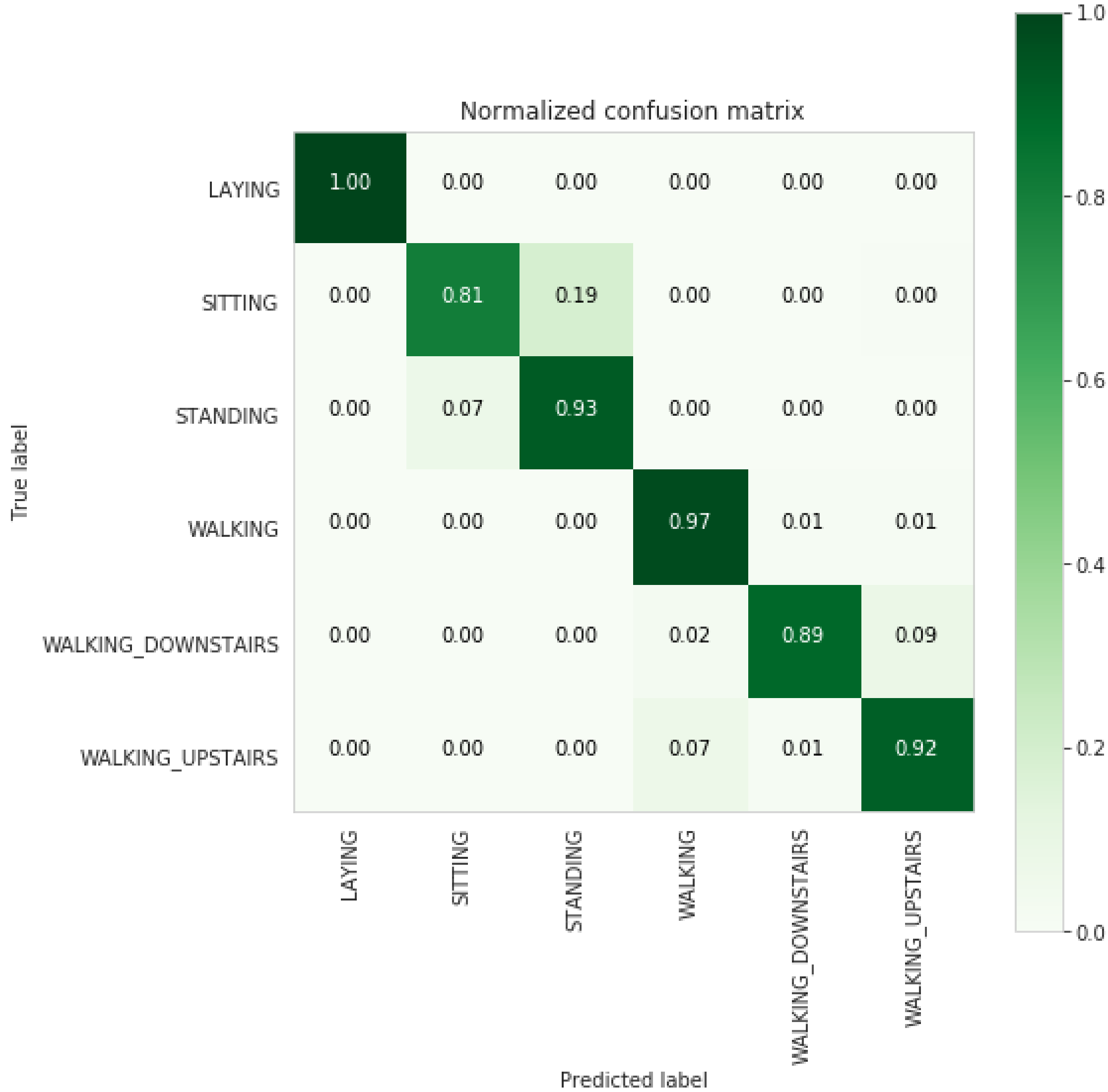
6. Gradient Boosted Decision Trees With GridSearch

In [38]:

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

executed in 37m 3s, finished 2018-11-06T14:58:56+05:30

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:37:02.972891


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.108744


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

    0.9219545300305395


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

 [[537   0   0   0   0   0]
 [  0 397  92   0   0   2]
 [  0  38 494   0   0   0]
 [  0   0   0 483   7   6]
 [  0   0   0  10 374  36]
 [  0   1   0  32   6 432]]
```




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

            LAYING       1.00      1.00      1.00       537
           SITTING       0.91      0.81      0.86       491
          STANDING       0.84      0.93      0.88       532
           WALKING       0.92      0.97      0.95       496
WALKING_DOWNSTAIRS       0.97      0.89      0.93       420
  WALKING_UPSTAIRS       0.91      0.92      0.91       471

       avg / total       0.92      0.92      0.92      2947

 --------------------------
|      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,
              presort='auto', random_state=None, subsample=1.0, 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 :

        0.9056039173014145
```

7. Comparing all models

In [40]:

```
print('\n                     Accuracy     Error')
print('                     ----------   --------')
print('Logistic Regression : {:.04}%       {:.04}%'.format(log_reg_grid_result['accura
                                                  100-(log_reg_grid_result['accuracy']

print('Linear SVC          : {:.04}%       {:.04}% '.format(lr_svc_grid_results['accur
                                                        100-(lr_svc_grid_results['accu

print('rbf SVM classifier  : {:.04}%      {:.04}% '.format(rbf_svm_grid_results['accur
                                                          100-(rbf_svm_grid_results['a

print('DecisionTree        : {:.04}%      {:.04}% '.format(dt_grid_results['accuracy']
                                                        100-(dt_grid_results['accuracy

print('Random Forest       : {:.04}%      {:.04}% '.format(rfc_grid_results['accuracy'
                                                           100-(rfc_grid_results['accu
print('GradientBoosting DT : {:.04}%      {:.04}% '.format(rfc_grid_results['accuracy'
                                                        100-(rfc_grid_results['accurac
```

executed in 14ms, finished 2018-11-06T15:01:20+05:30

```
                      Accuracy     Error
                      ----------   --------
Logistic Regression : 96.3%        3.699%
Linear SVC          : 96.81%        3.19%
rbf SVM classifier  : 96.27%       3.733%
DecisionTree        : 86.39%       13.61%
Random Forest       : 91.01%       8.992%
GradientBoosting DT : 91.01%       8.992%
```

In [ ]:

```
1
```