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

In [1]:

```
import numpy as np
import pandas as pd

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

```
No of Features: 561
```

Obtain the train data

In [2]:

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

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

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

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

```
D:\installed\Anaconda3\lib\site-packages\pandas\io\parsers.py:678: UserWar
ning: Duplicate names specified. This will raise an error in the future.
  return _read(filepath_or_buffer, kwds)
```

Out[2]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAc mad()
6015	0.2797	-0.004397	-0.10952	0.359081	0.119909	-0.177541	0.337963

1 rows × 564 columns

In [3]:

```
train.shape
```

Out[3]:

```
(7352, 564)
```

Obtain the test data

In [4]:

```
# 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=N
one, names=features)

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

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


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

```
D:\installed\Anaconda3\lib\site-packages\pandas\io\parsers.py:678: UserWar
ning: Duplicate names specified. This will raise an error in the future.
  return _read(filepath_or_buffer, kwds)
```

Out[4]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAc mad()
2261	0.279196	-0.018261	-0.103376	-0.996955	-0.982959	-0.988239	-0.9972

1 rows × 564 columns

In [5]:

```
test.shape
```

Out[5]:

```
(2947, 564)
```

Data Cleaning

1. Check for Duplicates

In [6]:

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

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

2. Checking for NaN/null values

In [7]:

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

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

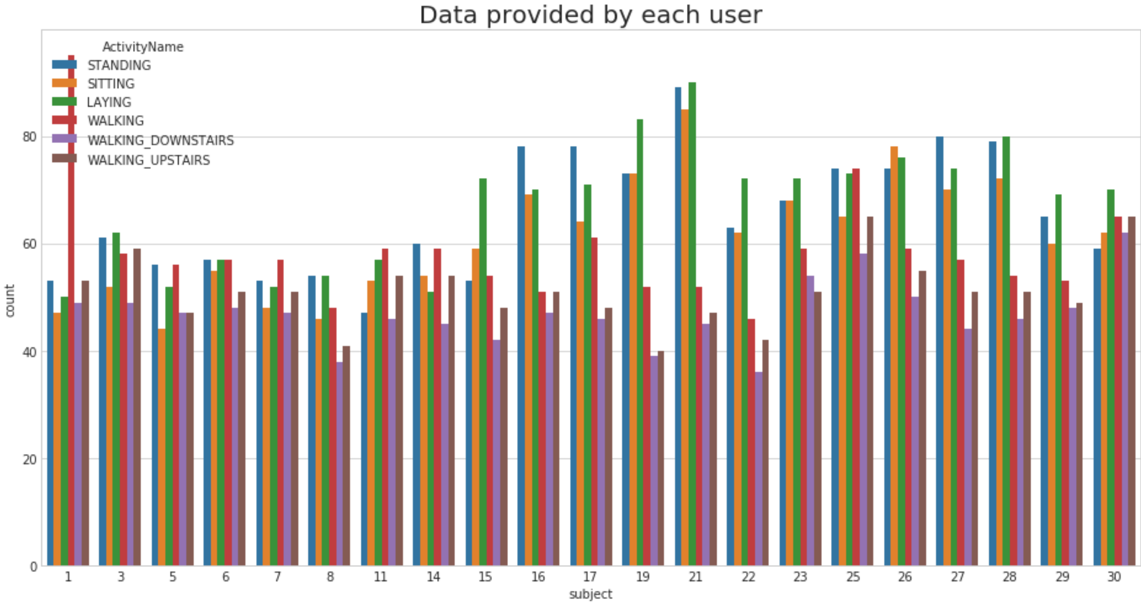
3. Check for data imbalance

In [8]:

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

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

In [9]:

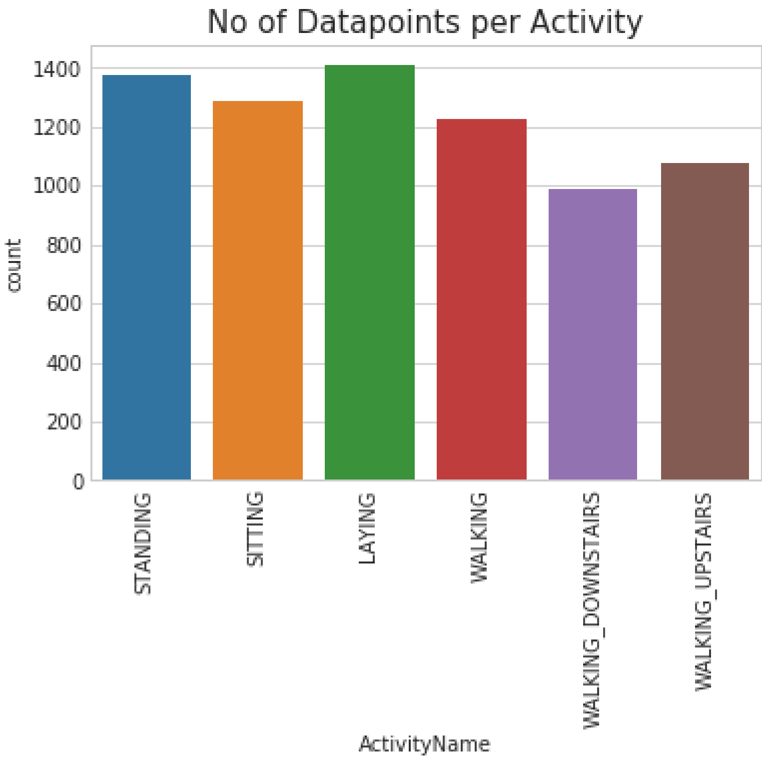
```
plt.figure(figsize=(16,8))
plt.title('Data provided by each user', fontsize=20)
sns.countplot(x='subject',hue='ActivityName', data = train)
plt.show()
```



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

In [10]:

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



Observation

> Our data is well balanced (almost)

4. Changing feature names

In [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
```

Out[11]:

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

5. Save this dataframe in a csv files

In [13]:

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

Exploratory Data Analysis

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

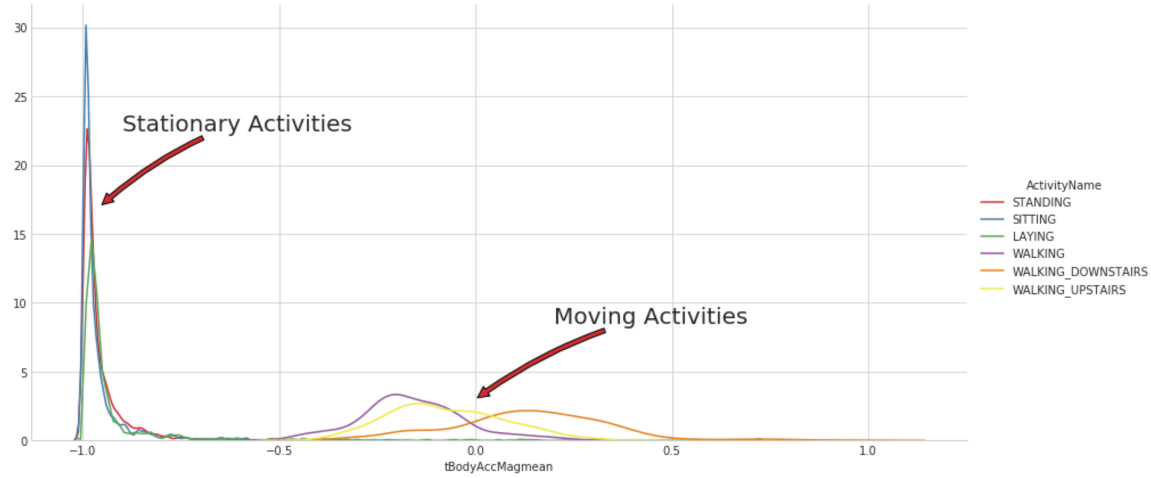
1. Featuring Engineering from Domain Knowledge

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

2. Stationary and Moving activities are completely different

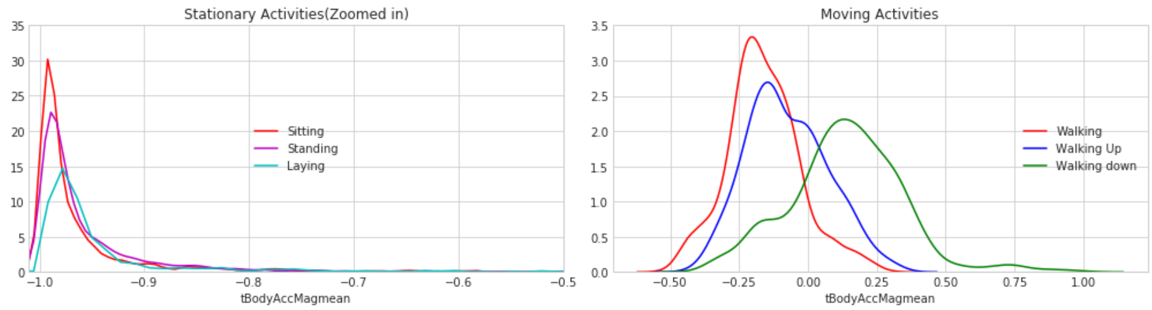
In [14]:

```
sns.set_palette("Set1", desat=0.80)
facetgrid = sns.FacetGrid(train, hue='ActivityName', size=6,aspect=2)
facetgrid.map(sns.distplot,'tBodyAccMagmean', hist=False)\
    .add_legend()
plt.annotate("Stationary Activities", 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()
```



In [15]:

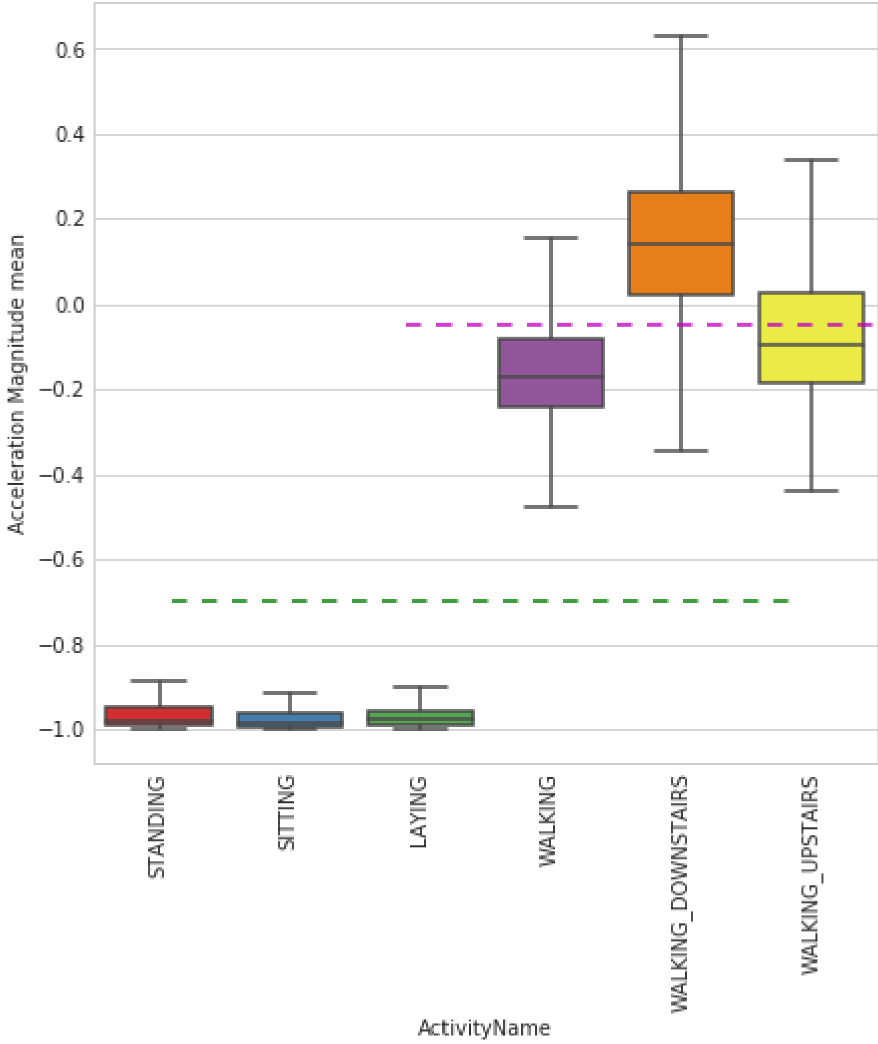
```
# for plotting purposes taking datapoints of each activity to a different dataframe
df1 = train[train['Activity']==1]
df2 = train[train['Activity']==2]
df3 = train[train['Activity']==3]
df4 = train[train['Activity']==4]
df5 = train[train['Activity']==5]
df6 = train[train['Activity']==6]

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

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


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



3. Magnitude of an acceleration can saperate it well

In [16]:

```
plt.figure(figsize=(7,7))
sns.boxplot(x='ActivityName', y='tBodyAccMagmean',data=train, showfliers=False, saturat
ion=1)
plt.ylabel('Acceleration Magnitude mean')
plt.axhline(y=-0.7, xmin=0.1, xmax=0.9,dashes=(5,5), c='g')
plt.axhline(y=-0.05, xmin=0.4, dashes=(5,5), c='m')
plt.xticks(rotation=90)
plt.show()
```



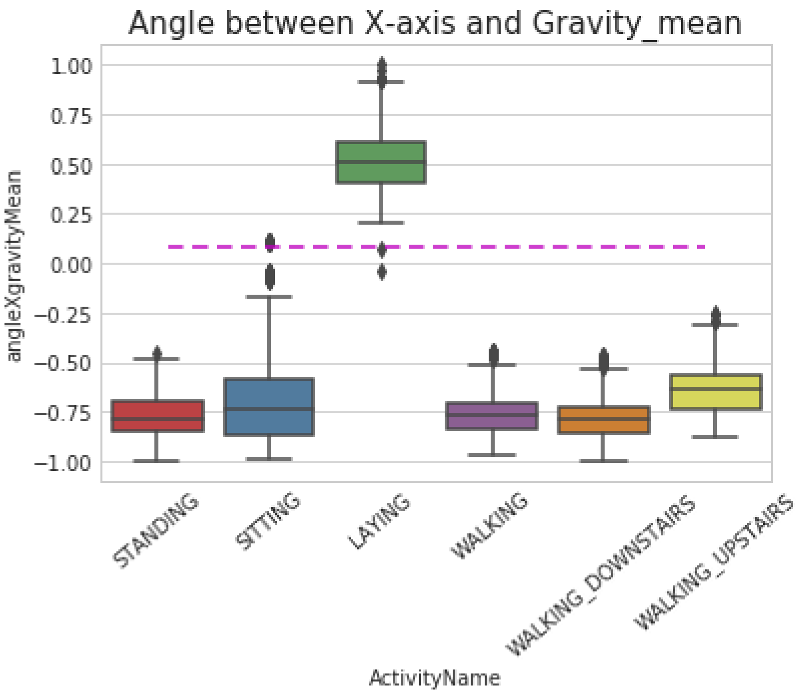
Observations:

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

4. Position of GravityAccelerationComponants also matters

In [17]:

```
sns.boxplot(x='ActivityName', y='angleXgravityMean', data=train)
plt.axhline(y=0.08, xmin=0.1, xmax=0.9,c='m',dashes=(5,3))
plt.title('Angle between X-axis and Gravity_mean', fontsize=15)
plt.xticks(rotation = 40)
plt.show()
```

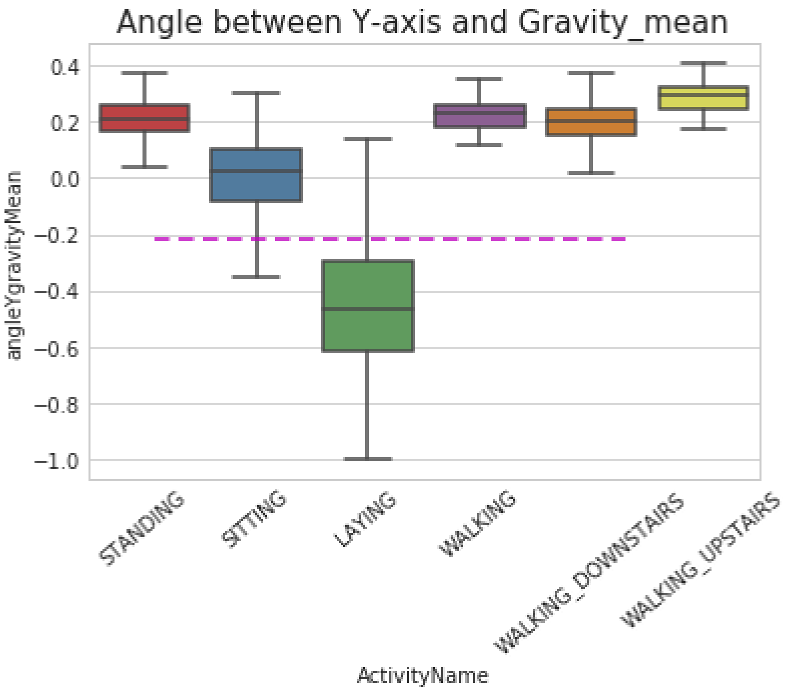


Observations:

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

In [18]:

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



Apply t-sne on the data

In [46]:

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

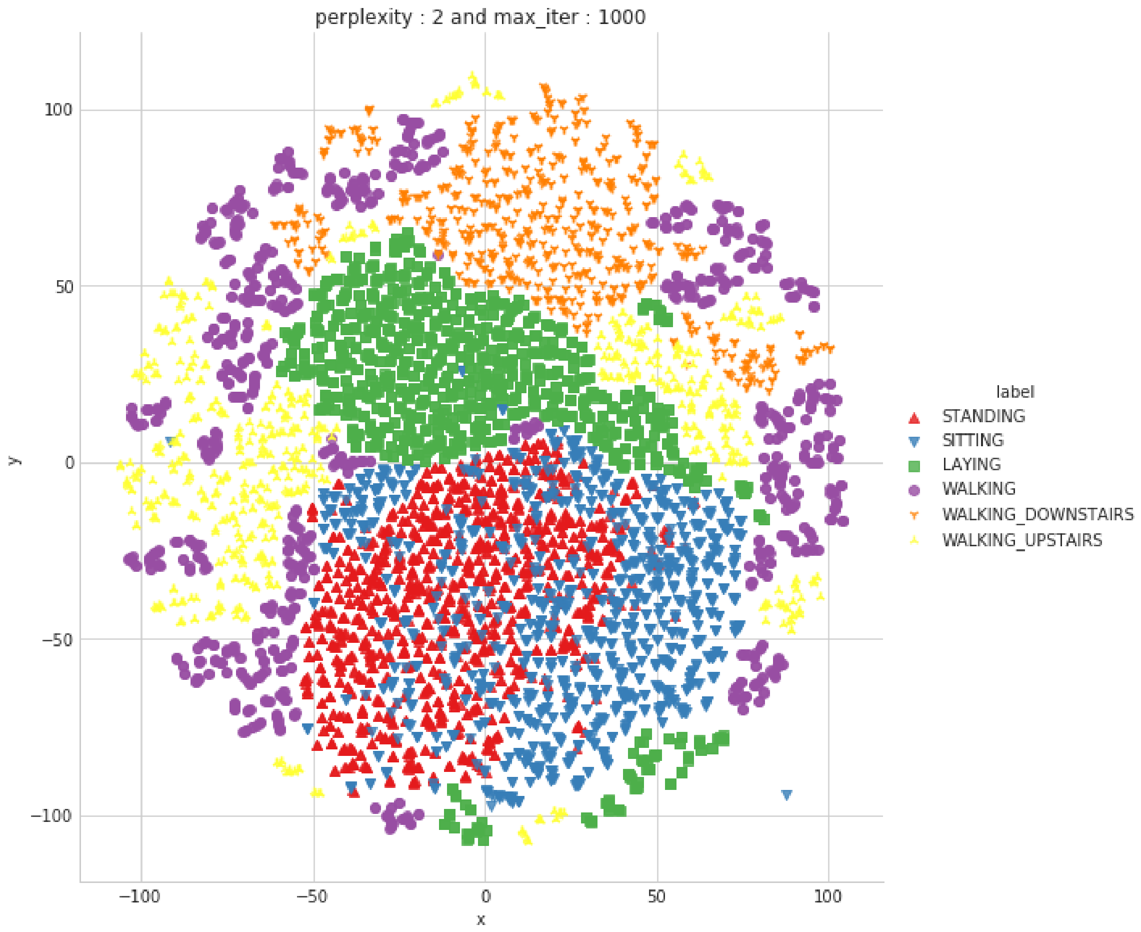
In [47]:

```
# 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'.form
at(perplexity, n_iter))
        X_reduced = TSNE(verbose=2, perplexity=perplexity).fit_transform(X_data)
        print('Done..')

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

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

In [48]:

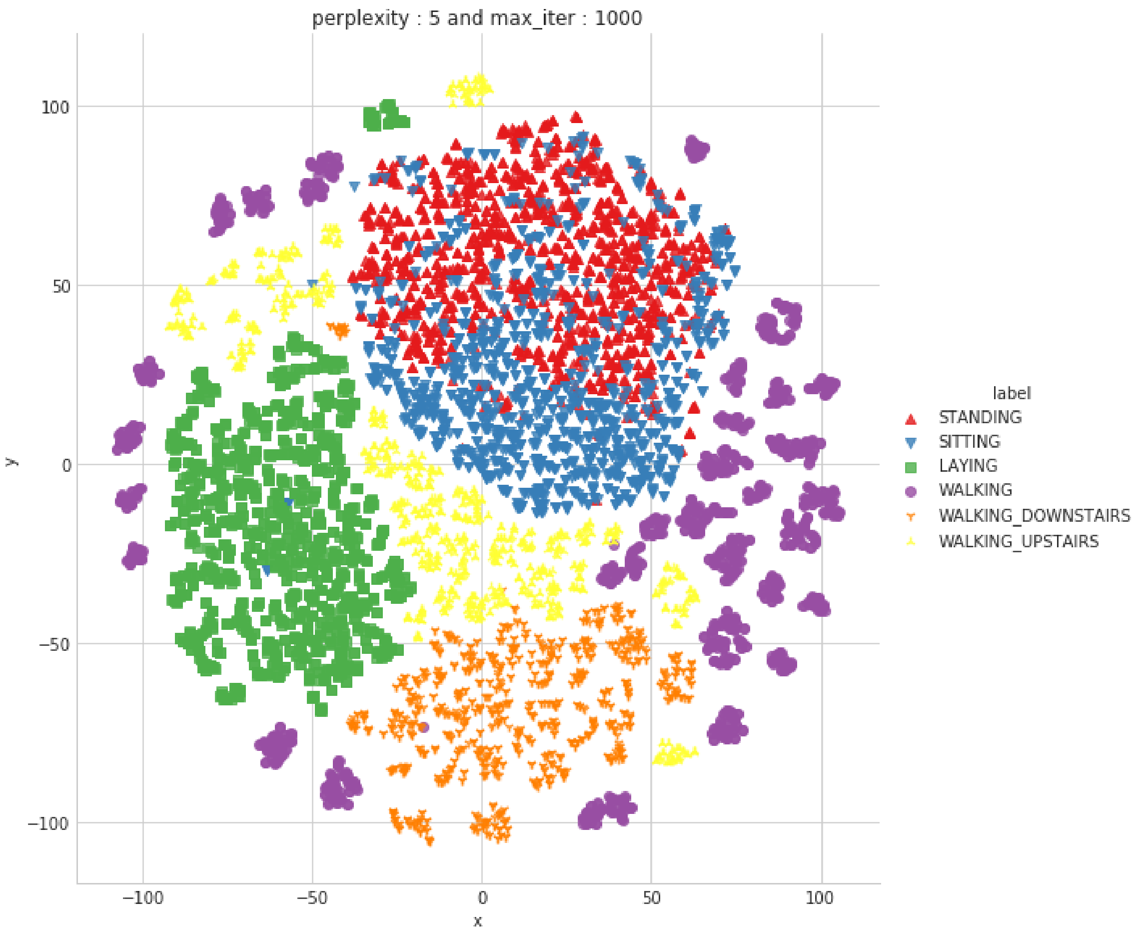
```
X_pre_tsne = train.drop(['subject', 'Activity','ActivityName'], axis=1)
y_pre_tsne = train['ActivityName']
perform_tsne(X_data = X_pre_tsne,y_data=y_pre_tsne, perplexities =[2,5,10,20,50])
```

```
performing tsne with perplexity 2 and with 1000 iterations at max
[t-SNE] Computing 7 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.426s...
[t-SNE] Computed neighbors for 7352 samples in 72.001s...
[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 0.071s
[t-SNE] Iteration 50: error = 124.8017578, gradient norm = 0.0253939 (50 i
terations in 16.625s)
[t-SNE] Iteration 100: error = 107.2019501, gradient norm = 0.0284782 (50
iterations in 9.735s)
[t-SNE] Iteration 150: error = 100.9872894, gradient norm = 0.0185151 (50
iterations in 5.346s)
[t-SNE] Iteration 200: error = 97.6054382, gradient norm = 0.0142084 (50 i
terations in 7.013s)
[t-SNE] Iteration 250: error = 95.3084183, gradient norm = 0.0132592 (50 i
terations in 5.703s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 95.308
418
[t-SNE] Iteration 300: error = 4.1209540, gradient norm = 0.0015668 (50 it
erations in 7.156s)
[t-SNE] Iteration 350: error = 3.2113254, gradient norm = 0.0009953 (50 it
erations in 8.022s)
[t-SNE] Iteration 400: error = 2.7819963, gradient norm = 0.0007203 (50 it
erations in 9.419s)
[t-SNE] Iteration 450: error = 2.5178111, gradient norm = 0.0005655 (50 it
erations in 9.370s)
[t-SNE] Iteration 500: error = 2.3341548, gradient norm = 0.0004804 (50 it
erations in 7.681s)
[t-SNE] Iteration 550: error = 2.1961622, gradient norm = 0.0004183 (50 it
erations in 7.097s)
[t-SNE] Iteration 600: error = 2.0867445, gradient norm = 0.0003664 (50 it
erations in 9.274s)
[t-SNE] Iteration 650: error = 1.9967778, gradient norm = 0.0003279 (50 it
erations in 7.697s)
[t-SNE] Iteration 700: error = 1.9210005, gradient norm = 0.0002984 (50 it
erations in 8.174s)
[t-SNE] Iteration 750: error = 1.8558111, gradient norm = 0.0002776 (50 it
erations in 9.747s)
[t-SNE] Iteration 800: error = 1.7989457, gradient norm = 0.0002569 (50 it
erations in 8.687s)
[t-SNE] Iteration 850: error = 1.7490212, gradient norm = 0.0002394 (50 it
erations in 8.407s)
[t-SNE] Iteration 900: error = 1.7043383, gradient norm = 0.0002224 (50 it
erations in 8.351s)
[t-SNE] Iteration 950: error = 1.6641431, gradient norm = 0.0002098 (50 it
erations in 7.841s)
[t-SNE] Iteration 1000: error = 1.6279151, gradient norm = 0.0001989 (50 i
terations in 5.623s)
[t-SNE] Error after 1000 iterations: 1.627915
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```


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

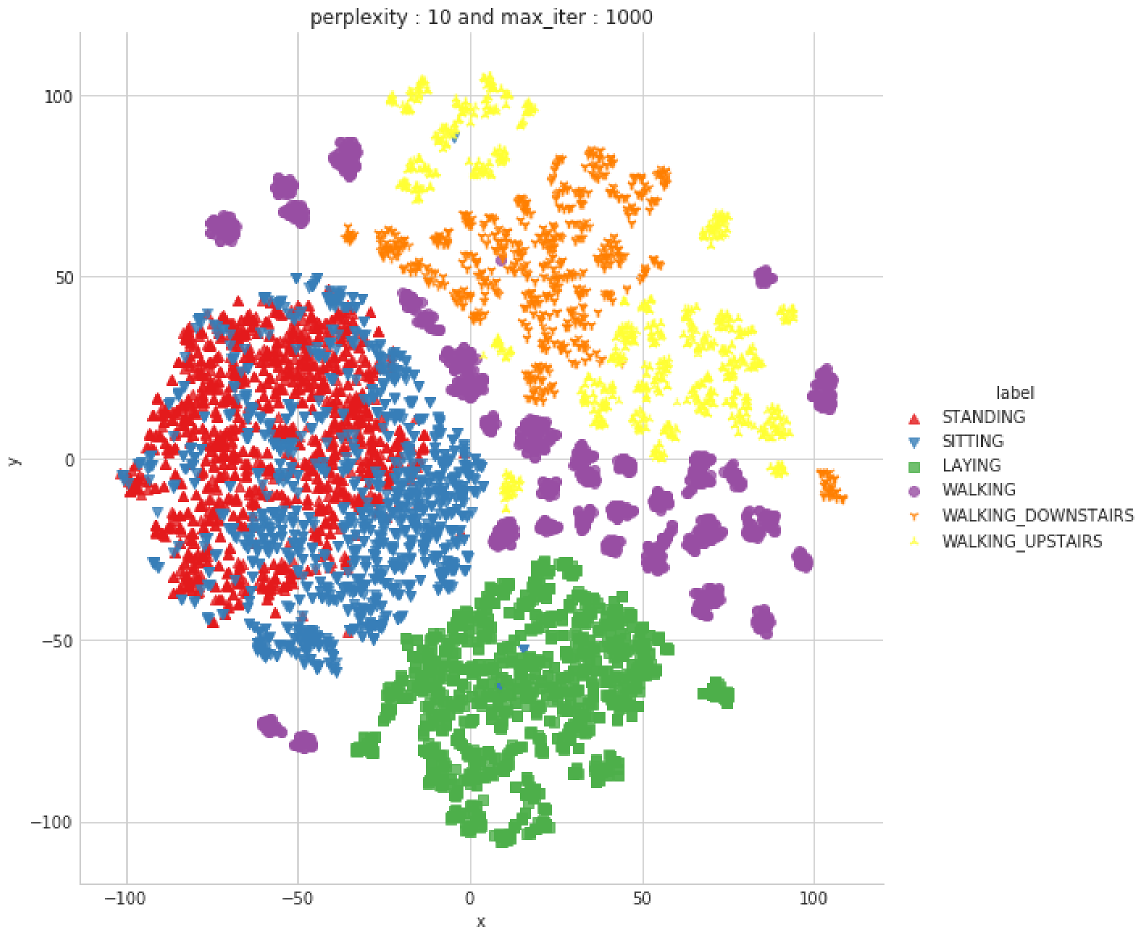
```
Done

performing tsne with perplexity 5 and with 1000 iterations at max
[t-SNE] Computing 16 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.263s...
[t-SNE] Computed neighbors for 7352 samples in 48.983s...
[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 0.122s
[t-SNE] Iteration 50: error = 114.1862640, gradient norm = 0.0184120 (50 i
terations in 55.655s)
[t-SNE] Iteration 100: error = 97.6535568, gradient norm = 0.0174309 (50 i
terations in 12.580s)
[t-SNE] Iteration 150: error = 93.1900101, gradient norm = 0.0101048 (50 i
terations in 9.180s)
[t-SNE] Iteration 200: error = 91.2315445, gradient norm = 0.0074560 (50 i
terations in 10.340s)
[t-SNE] Iteration 250: error = 90.0714417, gradient norm = 0.0057667 (50 i
terations in 9.458s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 90.071
442
[t-SNE] Iteration 300: error = 3.5796804, gradient norm = 0.0014691 (50 it
erations in 8.718s)
[t-SNE] Iteration 350: error = 2.8173938, gradient norm = 0.0007508 (50 it
erations in 10.180s)
[t-SNE] Iteration 400: error = 2.4344938, gradient norm = 0.0005251 (50 it
erations in 10.506s)
[t-SNE] Iteration 450: error = 2.2156141, gradient norm = 0.0004069 (50 it
erations in 10.072s)
[t-SNE] Iteration 500: error = 2.0703306, gradient norm = 0.0003340 (50 it
erations in 10.511s)
[t-SNE] Iteration 550: error = 1.9646366, gradient norm = 0.0002816 (50 it
erations in 9.792s)
[t-SNE] Iteration 600: error = 1.8835558, gradient norm = 0.0002471 (50 it
erations in 9.098s)
[t-SNE] Iteration 650: error = 1.8184001, gradient norm = 0.0002184 (50 it
erations in 8.656s)
[t-SNE] Iteration 700: error = 1.7647167, gradient norm = 0.0001961 (50 it
erations in 9.063s)
[t-SNE] Iteration 750: error = 1.7193680, gradient norm = 0.0001796 (50 it
erations in 9.754s)
[t-SNE] Iteration 800: error = 1.6803776, gradient norm = 0.0001655 (50 it
erations in 9.540s)
[t-SNE] Iteration 850: error = 1.6465144, gradient norm = 0.0001538 (50 it
erations in 9.953s)
[t-SNE] Iteration 900: error = 1.6166563, gradient norm = 0.0001421 (50 it
erations in 10.270s)
[t-SNE] Iteration 950: error = 1.5901035, gradient norm = 0.0001335 (50 it
erations in 6.609s)
[t-SNE] Iteration 1000: error = 1.5664237, gradient norm = 0.0001257 (50 i
terations in 8.553s)
[t-SNE] Error after 1000 iterations: 1.566424
Done..
```

```
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```



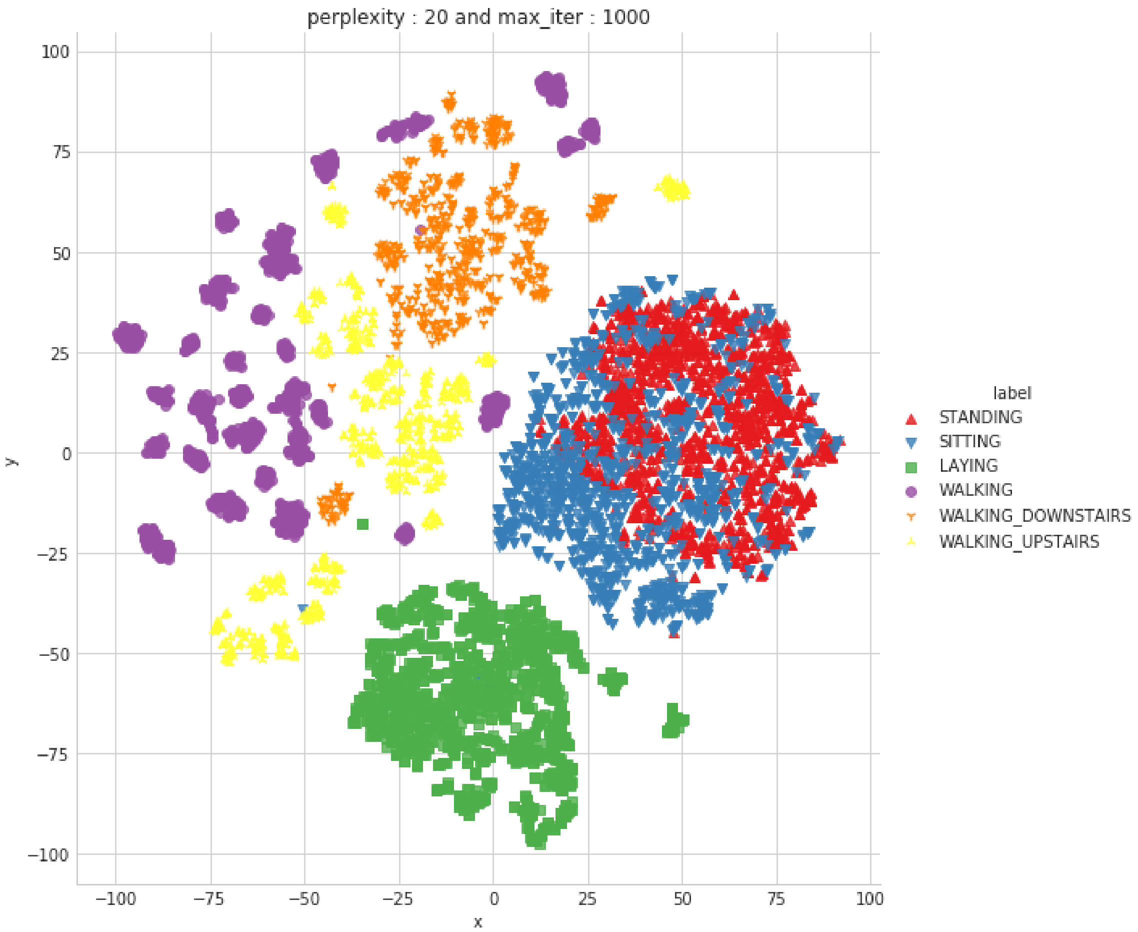
```
Done

performing tsne with perplexity 10 and with 1000 iterations at max
[t-SNE] Computing 31 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.410s...
[t-SNE] Computed neighbors for 7352 samples in 64.801s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.133828
[t-SNE] Computed conditional probabilities in 0.214s
[t-SNE] Iteration 50: error = 106.0169220, gradient norm = 0.0194293 (50 i
terations in 24.550s)
[t-SNE] Iteration 100: error = 90.3036194, gradient norm = 0.0097653 (50 i
terations in 11.936s)
[t-SNE] Iteration 150: error = 87.3132935, gradient norm = 0.0053059 (50 i
terations in 11.246s)
[t-SNE] Iteration 200: error = 86.1169128, gradient norm = 0.0035844 (50 i
terations in 11.864s)
[t-SNE] Iteration 250: error = 85.4133606, gradient norm = 0.0029100 (50 i
terations in 11.944s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.413
361
[t-SNE] Iteration 300: error = 3.1394315, gradient norm = 0.0013976 (50 it
erations in 11.742s)
[t-SNE] Iteration 350: error = 2.4929206, gradient norm = 0.0006466 (50 it
erations in 11.627s)
[t-SNE] Iteration 400: error = 2.1733041, gradient norm = 0.0004230 (50 it
erations in 11.846s)
[t-SNE] Iteration 450: error = 1.9884514, gradient norm = 0.0003124 (50 it
erations in 11.405s)
[t-SNE] Iteration 500: error = 1.8702440, gradient norm = 0.0002514 (50 it
erations in 11.320s)
[t-SNE] Iteration 550: error = 1.7870129, gradient norm = 0.0002107 (50 it
erations in 12.009s)
[t-SNE] Iteration 600: error = 1.7246909, gradient norm = 0.0001824 (50 it
erations in 10.632s)
[t-SNE] Iteration 650: error = 1.6758548, gradient norm = 0.0001590 (50 it
erations in 11.270s)
[t-SNE] Iteration 700: error = 1.6361949, gradient norm = 0.0001451 (50 it
erations in 12.072s)
[t-SNE] Iteration 750: error = 1.6034756, gradient norm = 0.0001305 (50 it
erations in 11.607s)
[t-SNE] Iteration 800: error = 1.5761518, gradient norm = 0.0001188 (50 it
erations in 9.409s)
[t-SNE] Iteration 850: error = 1.5527289, gradient norm = 0.0001113 (50 it
erations in 8.309s)
[t-SNE] Iteration 900: error = 1.5328671, gradient norm = 0.0001021 (50 it
erations in 9.433s)
[t-SNE] Iteration 950: error = 1.5152045, gradient norm = 0.0000974 (50 it
erations in 11.488s)
[t-SNE] Iteration 1000: error = 1.4999681, gradient norm = 0.0000933 (50 i
terations in 10.593s)
[t-SNE] Error after 1000 iterations: 1.499968
Done..
```

```
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```



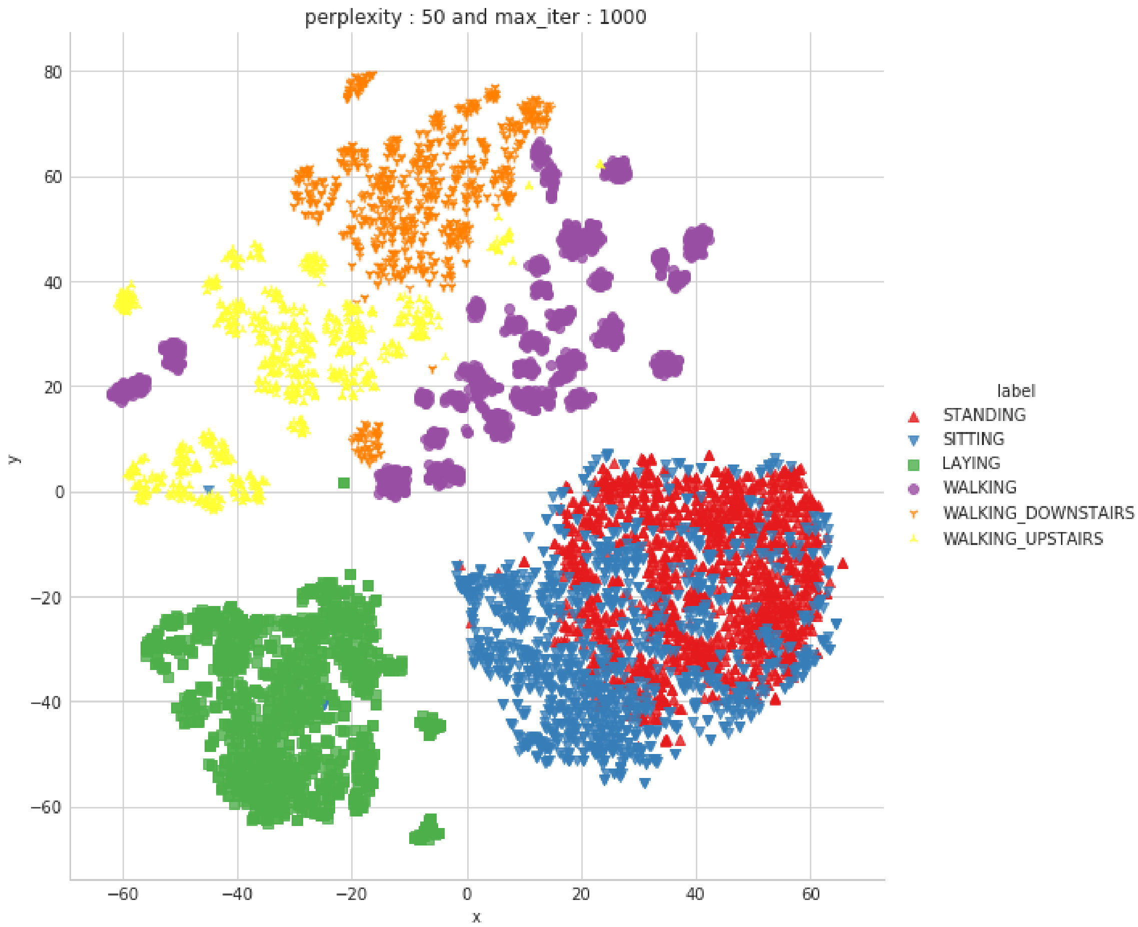
```
Done

performing tsne with perplexity 20 and with 1000 iterations at max
[t-SNE] Computing 61 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.425s...
[t-SNE] Computed neighbors for 7352 samples in 61.792s...
[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 0.355s
[t-SNE] Iteration 50: error = 97.5202179, gradient norm = 0.0223863 (50 it
erations in 21.168s)
[t-SNE] Iteration 100: error = 83.9500732, gradient norm = 0.0059110 (50 i
terations in 17.306s)
[t-SNE] Iteration 150: error = 81.8804779, gradient norm = 0.0035797 (50 i
terations in 14.258s)
[t-SNE] Iteration 200: error = 81.1615143, gradient norm = 0.0022536 (50 i
terations in 14.130s)
[t-SNE] Iteration 250: error = 80.7704086, gradient norm = 0.0018108 (50 i
terations in 15.340s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 80.770
409
[t-SNE] Iteration 300: error = 2.6957574, gradient norm = 0.0012993 (50 it
erations in 13.605s)
[t-SNE] Iteration 350: error = 2.1637220, gradient norm = 0.0005765 (50 it
erations in 13.248s)
[t-SNE] Iteration 400: error = 1.9143614, gradient norm = 0.0003474 (50 it
erations in 14.774s)
[t-SNE] Iteration 450: error = 1.7684202, gradient norm = 0.0002458 (50 it
erations in 15.502s)
[t-SNE] Iteration 500: error = 1.6744757, gradient norm = 0.0001923 (50 it
erations in 14.808s)
[t-SNE] Iteration 550: error = 1.6101606, gradient norm = 0.0001575 (50 it
erations in 14.043s)
[t-SNE] Iteration 600: error = 1.5641028, gradient norm = 0.0001344 (50 it
erations in 15.769s)
[t-SNE] Iteration 650: error = 1.5291905, gradient norm = 0.0001182 (50 it
erations in 15.834s)
[t-SNE] Iteration 700: error = 1.5024391, gradient norm = 0.0001055 (50 it
erations in 15.398s)
[t-SNE] Iteration 750: error = 1.4809053, gradient norm = 0.0000965 (50 it
erations in 14.594s)
[t-SNE] Iteration 800: error = 1.4631859, gradient norm = 0.0000884 (50 it
erations in 15.025s)
[t-SNE] Iteration 850: error = 1.4486470, gradient norm = 0.0000832 (50 it
erations in 14.060s)
[t-SNE] Iteration 900: error = 1.4367288, gradient norm = 0.0000804 (50 it
erations in 12.389s)
[t-SNE] Iteration 950: error = 1.4270191, gradient norm = 0.0000761 (50 it
erations in 10.392s)
[t-SNE] Iteration 1000: error = 1.4189968, gradient norm = 0.0000787 (50 i
terations in 12.355s)
[t-SNE] Error after 1000 iterations: 1.418997
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 0.376s...
[t-SNE] Computed neighbors for 7352 samples in 73.164s...
[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 0.844s
[t-SNE] Iteration 50: error = 86.1525574, gradient norm = 0.0242986 (50 it
erations in 36.249s)
[t-SNE] Iteration 100: error = 75.9874649, gradient norm = 0.0061005 (50 i
terations in 30.453s)
[t-SNE] Iteration 150: error = 74.7072296, gradient norm = 0.0024708 (50 i
terations in 28.461s)
[t-SNE] Iteration 200: error = 74.2736282, gradient norm = 0.0018644 (50 i
terations in 27.735s)
[t-SNE] Iteration 250: error = 74.0722427, gradient norm = 0.0014078 (50 i
terations in 26.835s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 74.072
243
[t-SNE] Iteration 300: error = 2.1539080, gradient norm = 0.0011796 (50 it
erations in 25.445s)
[t-SNE] Iteration 350: error = 1.7567128, gradient norm = 0.0004845 (50 it
erations in 21.282s)
[t-SNE] Iteration 400: error = 1.5888531, gradient norm = 0.0002798 (50 it
erations in 21.015s)
[t-SNE] Iteration 450: error = 1.4956820, gradient norm = 0.0001894 (50 it
erations in 23.332s)
[t-SNE] Iteration 500: error = 1.4359720, gradient norm = 0.0001420 (50 it
erations in 23.083s)
[t-SNE] Iteration 550: error = 1.3947564, gradient norm = 0.0001117 (50 it
erations in 19.626s)
[t-SNE] Iteration 600: error = 1.3653858, gradient norm = 0.0000949 (50 it
erations in 22.752s)
[t-SNE] Iteration 650: error = 1.3441534, gradient norm = 0.0000814 (50 it
erations in 23.972s)
[t-SNE] Iteration 700: error = 1.3284039, gradient norm = 0.0000742 (50 it
erations in 20.636s)
[t-SNE] Iteration 750: error = 1.3171139, gradient norm = 0.0000700 (50 it
erations in 20.407s)
[t-SNE] Iteration 800: error = 1.3085558, gradient norm = 0.0000657 (50 it
erations in 24.951s)
[t-SNE] Iteration 850: error = 1.3017821, gradient norm = 0.0000603 (50 it
erations in 24.719s)
[t-SNE] Iteration 900: error = 1.2962619, gradient norm = 0.0000586 (50 it
erations in 24.500s)
[t-SNE] Iteration 950: error = 1.2914882, gradient norm = 0.0000573 (50 it
erations in 24.132s)
[t-SNE] Iteration 1000: error = 1.2874244, gradient norm = 0.0000546 (50 i
terations in 22.840s)
[t-SNE] Error after 1000 iterations: 1.287424
Done..
```

```
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```




```
Done
```

Applying traditional ML models on the handcrafted features

In [1]:

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

Obtain the train and test data

In [2]:

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

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

In [3]:

```
train.head(3)
```

Out[3]:

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

3 rows × 564 columns

In [4]:

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

In [5]:

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

In [6]:

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

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

Let's model with our data

Labels that are useful in plotting confusion matrix

In [7]:

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

Function to plot the confusion matrix

In [8]:

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

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

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

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

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

Generic function to run any model specified

In [9]:

```
from datetime import datetime
def perform_model(model, X_train, y_train, X_test, y_test, class_labels, cm_normalize=T
rue, \
                 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 c
onfusion matrix', cmap = cm_cmap)
    plt.show()

    # get classification report
    print('-------------------------')
    print('| Classifiction Report |')
```

```
    print('-------------------------')
    classification_report = metrics.classification_report(y_test, y_pred)
    # store report in results
    results['classification_report'] = classification_report
    print(classification_report)

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

    return results
```

Method to print the gridsearch Attributes

In [10]:

```
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(mod
el.best_score_))
```

1. Logistic Regression with Grid Search

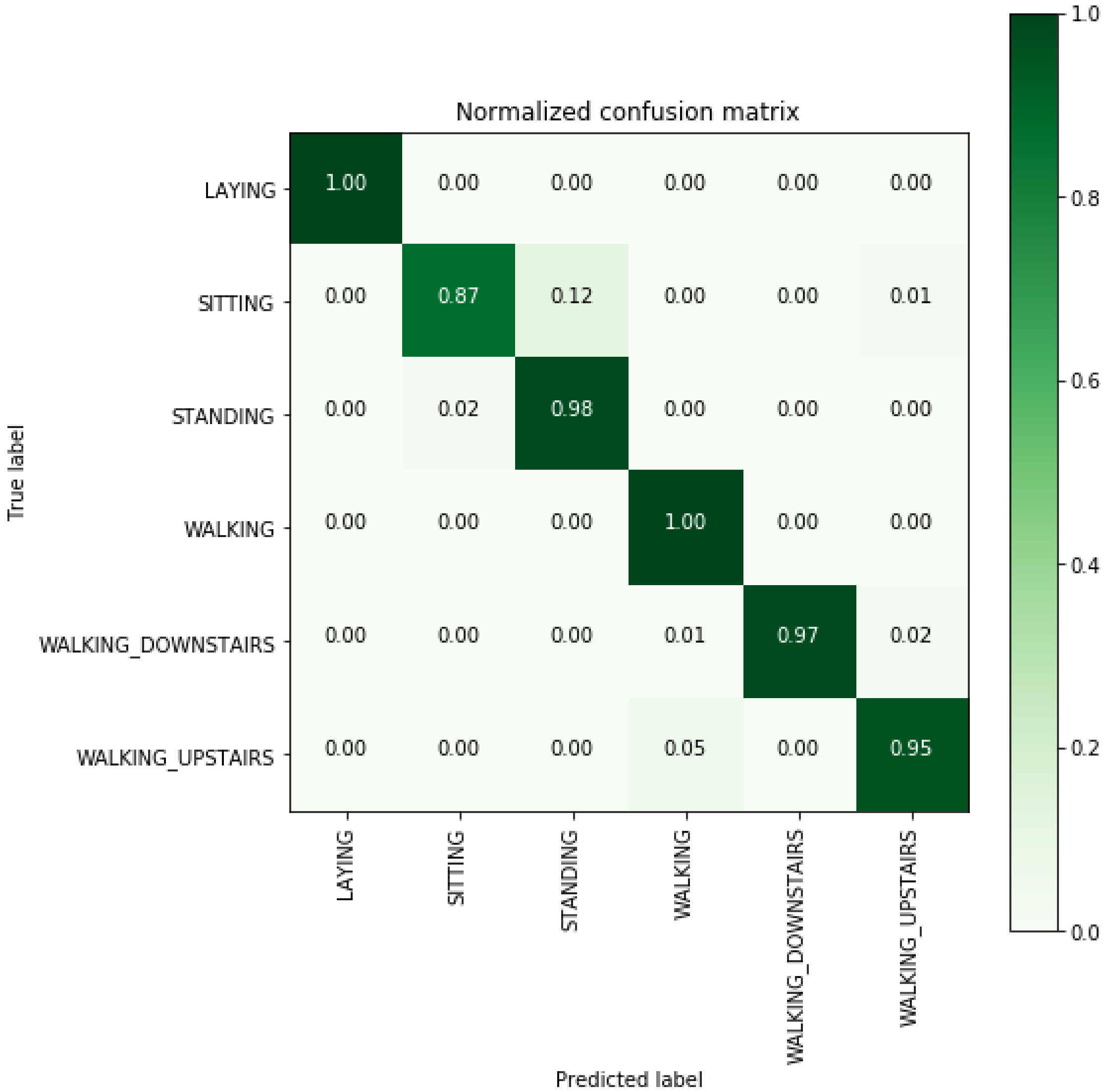
In [11]:

```
from sklearn import linear_model
from sklearn import metrics

from sklearn.model_selection import GridSearchCV
```

In [12]:

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

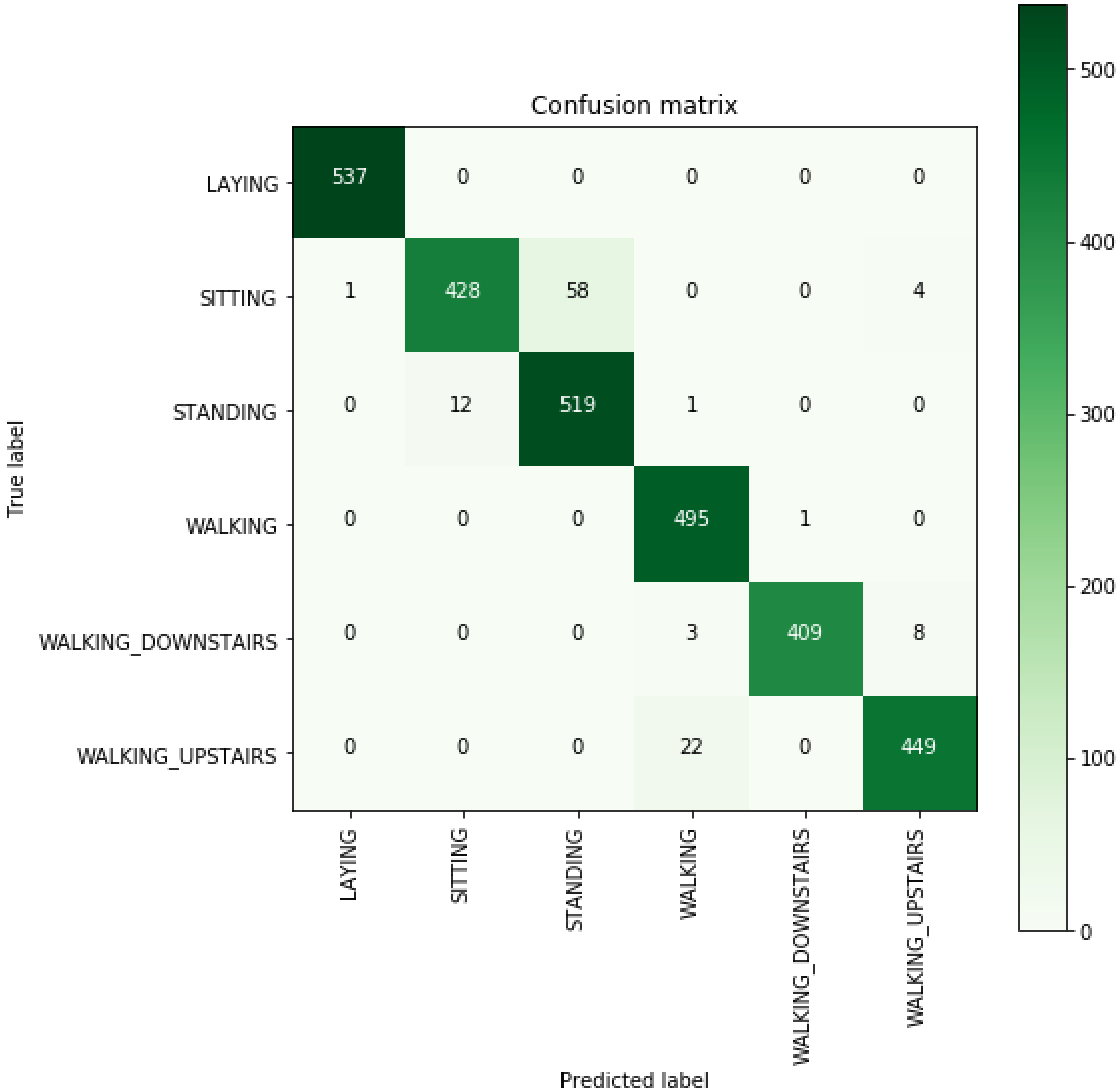
```
training the model..
Fitting 3 folds for each of 12 candidates, totalling 36 fits

[Parallel(n_jobs=-1)]: Done  36 out of  36 | elapsed:  1.2min finished

Done


training_time(HH:MM:SS.ms) - 0:01:25.843810


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.009192


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

    0.9626739056667798


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

 [[537   0   0   0   0   0]
 [  1 428  58   0   0   4]
 [  0  12 519   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 [13]:

```
plt.figure(figsize=(8,8))
plt.grid(b=False)
plot_confusion_matrix(log_reg_grid_results['confusion_matrix'], classes=labels, cmap=pl
t.cm.Greens, )
plt.show()
```



In [14]:

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

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

        LogisticRegression(C=30, class_weight=None, dual=False, fit_interc
ept=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 :

        0.9461371055495104
```

2. Linear SVC with GridSearch

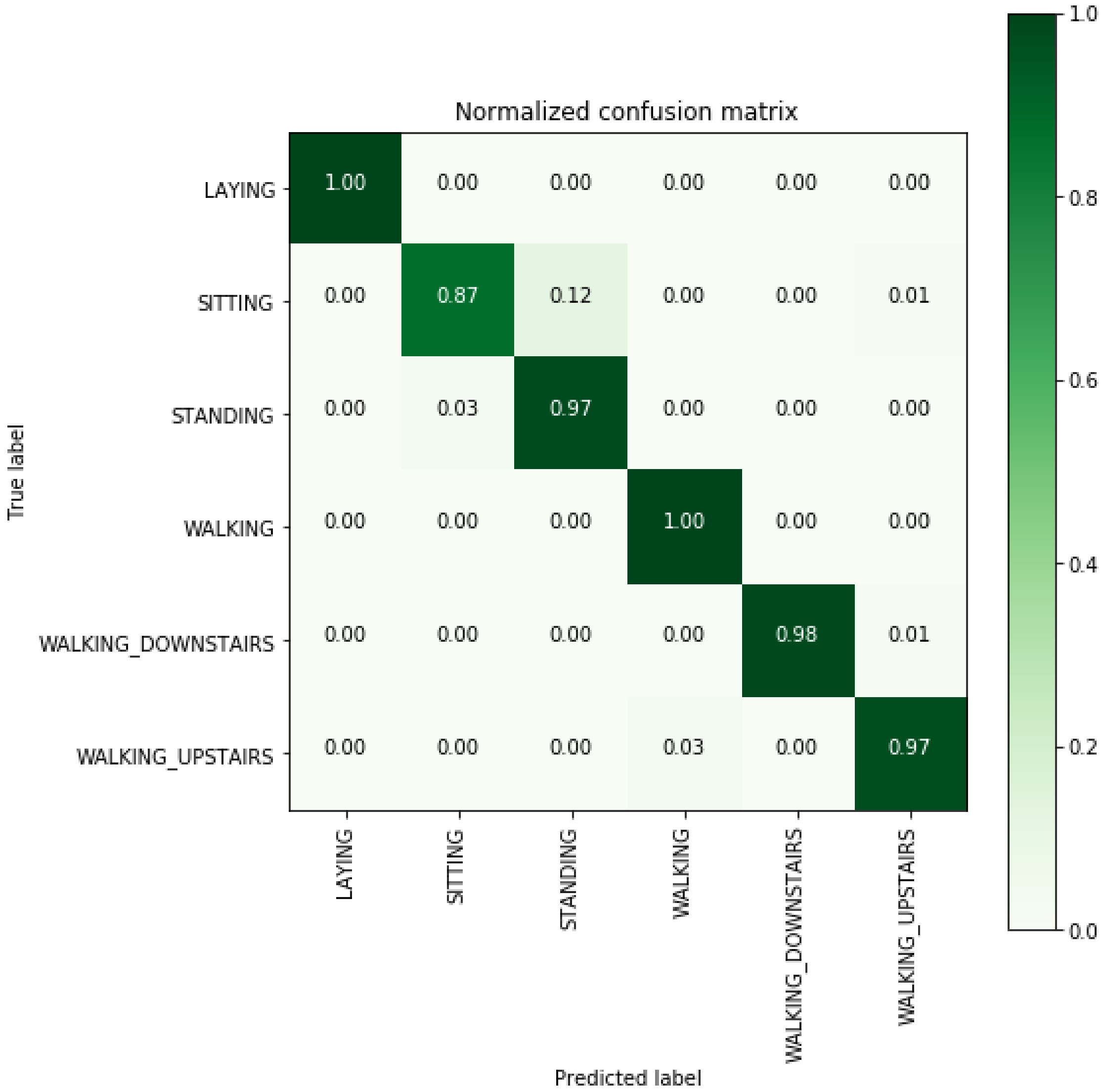
In [15]:

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

In [16]:

```
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, clas
s_labels=labels)
```

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

[Parallel(n_jobs=-1)]: Done  18 out of  18 | elapsed:   24.9s finished

Done


training_time(HH:MM:SS.ms) - 0:00:32.951942


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.012182


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

    0.9660671869697998


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

 [[537   0   0   0   0   0]
 [  2 426  58   0   0   5]
 [  0  14 518   0   0   0]
 [  0   0   0 495   0   1]
 [  0   0   0   2 413   5]
 [  0   0   0  12   1 458]]
```


Normalized confusion matrix
True label
Predicted label
LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
1.00 0.00 0.00 0.00 0.00 0.00
0.00 0.87 0.12 0.00 0.00 0.01
0.00 0.03 0.97 0.00 0.00 0.00
0.00 0.00 0.00 1.00 0.00 0.00
0.00 0.00 0.00 0.00 0.98 0.01
0.00 0.00 0.00 0.03 0.00 0.97
1.0
0.8
0.6
0.4
0.2
0.0

```
------------------------
| 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      0.97      0.94       532
           WALKING       0.97      1.00      0.99       496
WALKING_DOWNSTAIRS       1.00      0.98      0.99       420
  WALKING_UPSTAIRS       0.98      0.97      0.97       471

       avg / total       0.97      0.97      0.97      2947
```

In [17]:

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

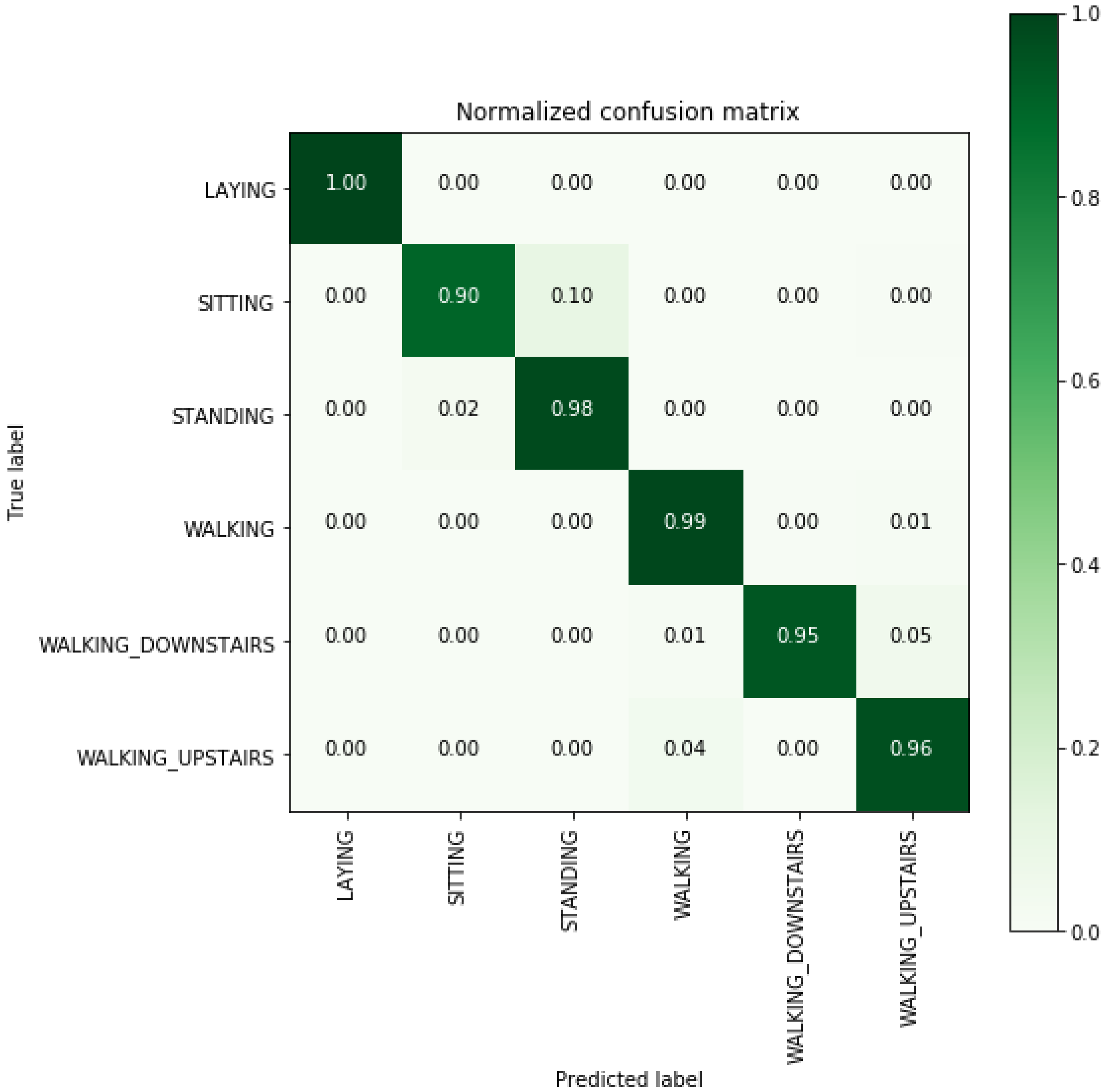
```
--------------------------
|      Best Estimator     |
--------------------------

        LinearSVC(C=8, 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': 8}

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.9465451577801959
```

3. Kernel SVM with GridSearch

In [18]:

```
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, cl
ass_labels=labels)
```

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:05:46.182889


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:05.221285


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


Normalized confusion matrix
True label
Predicted label
LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
1.00 0.00 0.00 0.00 0.00 0.00
0.00 0.90 0.10 0.00 0.00 0.00
0.00 0.02 0.98 0.00 0.00 0.00
0.00 0.00 0.00 0.99 0.00 0.01
0.00 0.00 0.00 0.01 0.95 0.05
0.00 0.00 0.00 0.04 0.00 0.96
1.0
0.8
0.6
0.4
0.2
0.0

```
-------------------------
| 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 [19]:

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

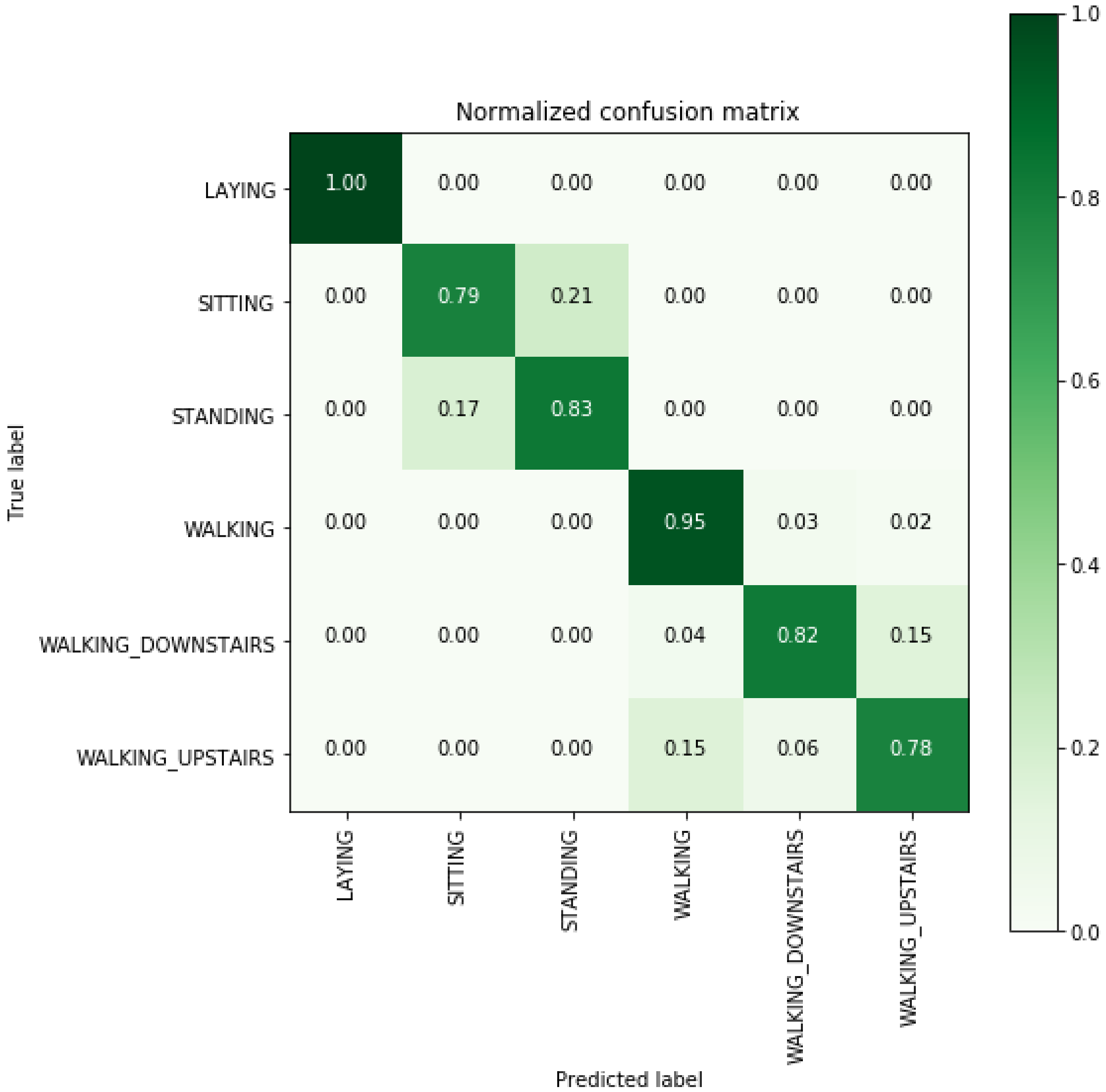
```
--------------------------
|      Best Estimator     |
--------------------------

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

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

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

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.9440968443960827
```

4. Decision Trees with GridSearchCV

In [20]:

```
from sklearn.tree import DecisionTreeClassifier
parameters = {'max_depth':np.arange(3,10,2)}
dt = DecisionTreeClassifier()
dt_grid = GridSearchCV(dt,param_grid=parameters, n_jobs=-1)
dt_grid_results = perform_model(dt_grid, X_train, y_train, X_test, y_test, class_labels
=labels)
print_grid_search_attributes(dt_grid_results['model'])
```

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:00:19.476858


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.012858


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

    0.8642687478791992


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

 [[537   0   0   0   0   0]
 [  0 386 105   0   0   0]
 [  0  93 439   0   0   0]
 [  0   0   0 472  16   8]
 [  0   0   0  15 344  61]
 [  0   0   0  73  29 369]]
```


Normalized confusion matrix
True label
Predicted label
LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
1.00 0.00 0.00 0.00 0.00 0.00
0.00 0.79 0.21 0.00 0.00 0.00
0.00 0.17 0.83 0.00 0.00 0.00
0.00 0.00 0.00 0.95 0.03 0.02
0.00 0.00 0.00 0.04 0.82 0.15
0.00 0.00 0.00 0.15 0.06 0.78
1.0
0.8
0.6
0.4
0.2
0.0

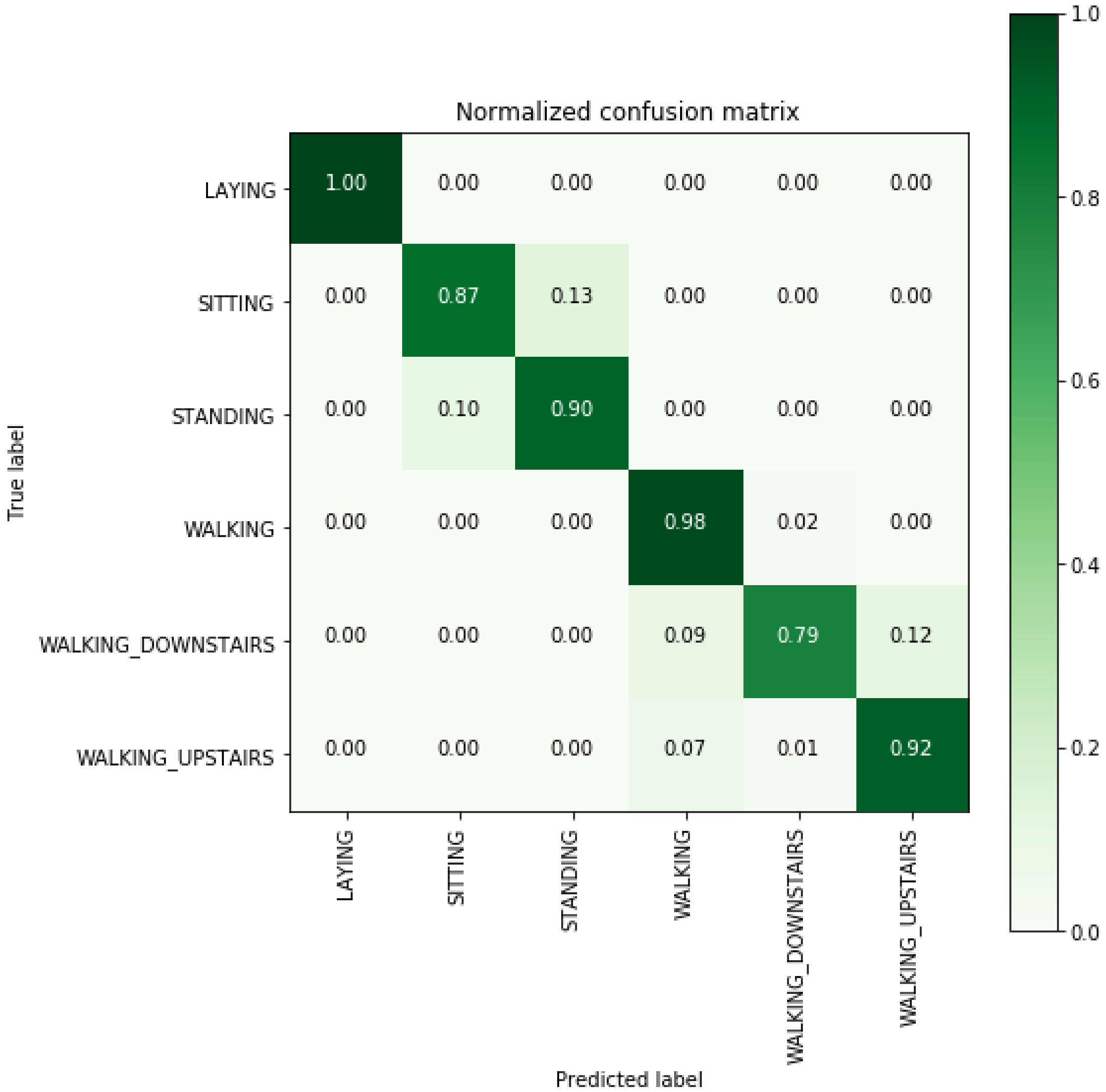
```
--------------------------
| Classifiction Report |
--------------------------
                    precision    recall  f1-score   support

            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       0.88      0.82      0.85       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_de
pth=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=Non
e,
            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 :

        0.8369151251360174
```

5. Random Forest Classifier with GridSearch

In [21]:

```
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_labe
ls=labels)
print_grid_search_attributes(rfc_grid_results['model'])
```

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:06:22.775270


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.025937


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

    0.9131319986426875


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

 [[537   0   0   0   0   0]
 [  0 427  64   0   0   0]
 [  0  52 480   0   0   0]
 [  0   0   0 484  10   2]
 [  0   0   0  38 332  50]
 [  0   0   0  34   6 431]]
```


Normalized confusion matrix
True label
Predicted label
LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
1.00 0.00 0.00 0.00 0.00 0.00
0.00 0.87 0.13 0.00 0.00 0.00
0.00 0.10 0.90 0.00 0.00 0.00
0.00 0.00 0.00 0.98 0.02 0.00
0.00 0.00 0.00 0.09 0.79 0.12
0.00 0.00 0.00 0.07 0.01 0.92
1.0
0.8
0.6
0.4
0.2
0.0

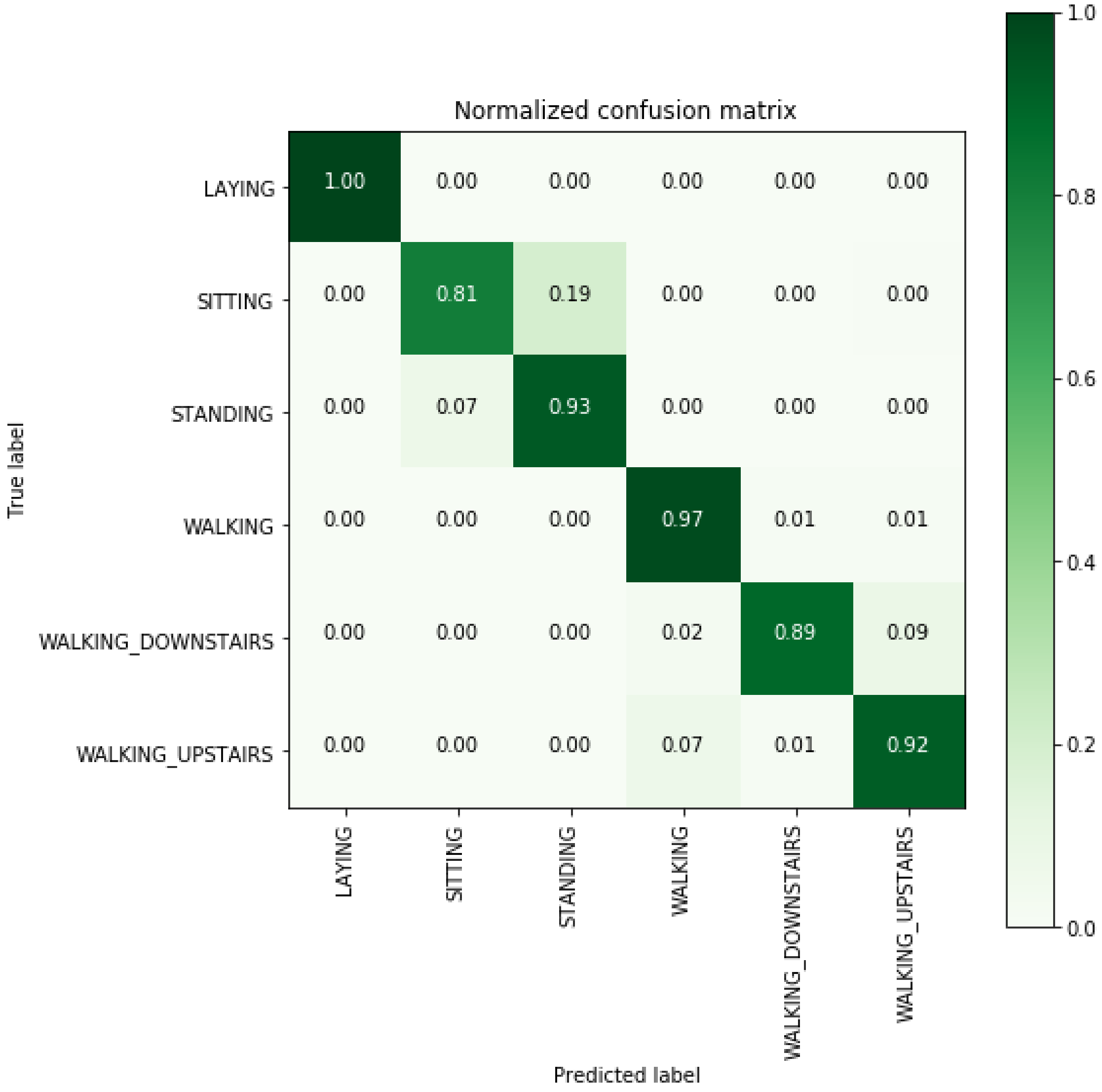
```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.89      0.87      0.88       491
          STANDING       0.88      0.90      0.89       532
           WALKING       0.87      0.98      0.92       496
WALKING_DOWNSTAIRS       0.95      0.79      0.86       420
  WALKING_UPSTAIRS       0.89      0.92      0.90       471

       avg / total       0.92      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=70, 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': 70}

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.9141730141458106
```

6. Gradient Boosted Decision Trees With GridSearch

In [22]:

```
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_la
bels=labels)
print_grid_search_attributes(gbdt_grid_results['model'])
```

```
training the model..
Done 


training_time(HH:MM:SS.ms) - 0:28:03.653432


Predicting test data
Done 


testing time(HH:MM:SS:ms) - 0:00:00.058843


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

    0.9222938581608415


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

 [[537   0   0   0   0   0]
 [  0 396  93   0   0   2]
 [  0  37 495   0   0   0]
 [  0   0   0 483   7   6]
 [  0   0   0  10 374  36]
 [  0   1   0  31   6 433]]
```


Normalized confusion matrix
True label
Predicted label
LAYING
SITTING
STANDING
WALKING
WALKING_DOWNSTAIRS
WALKING_UPSTAIRS
1.00 0.00 0.00 0.00 0.00 0.00
0.00 0.81 0.19 0.00 0.00 0.00
0.00 0.07 0.93 0.00 0.00 0.00
0.00 0.00 0.00 0.97 0.01 0.01
0.00 0.00 0.00 0.02 0.89 0.09
0.00 0.00 0.00 0.07 0.01 0.92
1.0
0.8
0.6
0.4
0.2
0.0

```
-------------------------
| 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=140,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False)

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

        {'max_depth': 5, 'n_estimators': 140}

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator : 

        0.904379760609358
```

7. Comparing all models

In [23]:

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

print('Linear SVC          : {:.04}%      {:.04}% '.format(lr_svc_grid_results['accura
cy'] * 100,\
                                                        100-(lr_svc_grid_results['accur
acy'] * 100)))

print('rbf SVM classifier  : {:.04}%     {:.04}% '.format(rbf_svm_grid_results['accura
cy'] * 100,\
                                                          100-(rbf_svm_grid_results['ac
curacy'] * 100)))

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

print('Random Forest       : {:.04}%     {:.04}% '.format(rfc_grid_results['accuracy']
* 100,\
                                                           100-(rfc_grid_results['accur
acy'] * 100)))
print('GradientBoosting DT : {:.04}%     {:.04}% '.format(rfc_grid_results['accuracy']
* 100,\
                                                        100-(rfc_grid_results['accurac
y'] * 100)))
```

```
                      Accuracy     Error
                      ----------   --------
Logistic Regression : 96.27%       3.733%
Linear SVC          : 96.61%       3.393%
rbf SVM classifier  : 96.27%      3.733%
DecisionTree        : 86.43%      13.57%
Random Forest       : 91.31%      8.687%
GradientBoosting DT : 91.31%      8.687%
```

We can choose ***Logistic regression*** or ***Linear SVC*** or ***rbf SVM***.

Conclusion :

In the real world, domain-knowledge, EDA and feature-engineering matter most.

Human activity recognition using LSTM

In [ ]:

```
# Importing Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from time import time
from datetime import datetime
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.wrappers.scikit_learn import KerasClassifier
from keras.constraints import maxnorm
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout
from keras.layers.normalization import BatchNormalization
```

In [0]:

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

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

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

1.0 Data

In [0]:

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

In [0]:

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

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

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

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

In [0]:

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

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

In [0]:

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

    return X_train, X_test, y_train, y_test
```

In [0]:

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

In [0]:

```
# Configuring a session
session_conf = tf.compat.v1.ConfigProto(
    intra_op_parallelism_threads=1,
    inter_op_parallelism_threads=1
)
```

In [0]:

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

In [0]:

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

In [0]:

```
# Loading the train and test data
import warnings
warnings.filterwarnings("ignore")

X_train, X_test, Y_train, Y_test = load_data()
```

In [0]:

```
type(X_train)
```

Out[0]:

```
numpy.ndarray
```

In [0]:

```
print((X_train[0][0]))
```

```
[ 1.808515e-04  1.076681e-02  5.556068e-02  3.019122e-02  6.601362e-02
  2.285864e-02  1.012817e+00 -1.232167e-01  1.029341e-01]
```

In [0]:

```
print((X_train[0]))
```

```
[[ 1.808515e-04  1.076681e-02  5.556068e-02 ...  1.012817e+00
  -1.232167e-01  1.029341e-01]
 [ 1.013856e-02  6.579480e-03  5.512483e-02 ...  1.022833e+00
  -1.268756e-01  1.056872e-01]
 [ 9.275574e-03  8.928878e-03  4.840473e-02 ...  1.022028e+00
  -1.240037e-01  1.021025e-01]
 ...
 [-1.147484e-03  1.714439e-04  2.647864e-03 ...  1.018445e+00
  -1.240696e-01  1.003852e-01]
 [-2.222655e-04  1.574181e-03  2.381057e-03 ...  1.019372e+00
  -1.227451e-01  9.987355e-02]
 [ 1.575500e-03  3.070189e-03 -2.269757e-03 ...  1.021171e+00
  -1.213260e-01  9.498741e-02]]
```

In [0]:

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

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

```
128
9
7352
```

In [0]:

```
print(n_classes)
```

```
6
```

In [0]:

```
np.save('X_train', X_train)
np.save('X_test', X_test)
np.save('Y_train', Y_train)
np.save('Y_test', Y_test)
```

In [3]:

```
from zipfile import ZipFile
file_name="/content/Colab.zip"

with ZipFile(file_name,'r') as zip:
    zip.extractall()
    print('Done')
```

```
Done
```

In [0]:

```
X_train= np.load('/content/Colab/X_train.npy')
X_test= np.load('/content/Colab/X_test.npy')
Y_train= np.load('/content/Colab/Y_train.npy')
Y_test= np.load('/content/Colab/Y_test.npy')
```

In [0]:

```
Y_test= np.load('/content/Colab/Y_test.npy')
```

2.0 Simple base model without hyperparameter tuning

In [0]:

```
# Initializing parameters
epochs = 30
batch_size = 16
n_hidden = 32
```

In [0]:

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 32)                5376
_________________________________________________________________
dropout_3 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 198
=================================================================
Total params: 5,574
Trainable params: 5,574
Non-trainable params: 0
_________________________________________________________________
```

In [0]:

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

In [0]:

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

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 92s 13ms/step - loss: 1.3018
- acc: 0.4395 - val_loss: 1.1254 - val_acc: 0.4662
Epoch 2/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.9666
- acc: 0.5880 - val_loss: 0.9491 - val_acc: 0.5714
Epoch 3/30
7352/7352 [==============================] - 97s 13ms/step - loss: 0.7812
- acc: 0.6408 - val_loss: 0.8286 - val_acc: 0.5850
Epoch 4/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.6941
- acc: 0.6574 - val_loss: 0.7297 - val_acc: 0.6128
Epoch 5/30
7352/7352 [==============================] - 92s 13ms/step - loss: 0.6336
- acc: 0.6912 - val_loss: 0.7359 - val_acc: 0.6787
Epoch 6/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.5859
- acc: 0.7134 - val_loss: 0.7015 - val_acc: 0.6939
Epoch 7/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.5692
- acc: 0.7477 - val_loss: 0.5995 - val_acc: 0.7387
Epoch 8/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.4899
- acc: 0.7809 - val_loss: 0.5762 - val_acc: 0.7387
Epoch 9/30
7352/7352 [==============================] - 90s 12ms/step - loss: 0.4482
- acc: 0.7886 - val_loss: 0.7413 - val_acc: 0.7126
Epoch 10/30
7352/7352 [==============================] - 90s 12ms/step - loss: 0.4132
- acc: 0.8077 - val_loss: 0.5048 - val_acc: 0.7513
Epoch 11/30
7352/7352 [==============================] - 89s 12ms/step - loss: 0.3985
- acc: 0.8274 - val_loss: 0.5234 - val_acc: 0.7452
Epoch 12/30
7352/7352 [==============================] - 91s 12ms/step - loss: 0.3378
- acc: 0.8638 - val_loss: 0.4114 - val_acc: 0.8833
Epoch 13/30
7352/7352 [==============================] - 91s 12ms/step - loss: 0.2947
- acc: 0.9051 - val_loss: 0.4386 - val_acc: 0.8731
Epoch 14/30
7352/7352 [==============================] - 90s 12ms/step - loss: 0.2448
- acc: 0.9291 - val_loss: 0.3768 - val_acc: 0.8921
Epoch 15/30
7352/7352 [==============================] - 91s 12ms/step - loss: 0.2157
- acc: 0.9331 - val_loss: 0.4441 - val_acc: 0.8931
Epoch 16/30
7352/7352 [==============================] - 90s 12ms/step - loss: 0.2053
- acc: 0.9366 - val_loss: 0.4162 - val_acc: 0.8968
Epoch 17/30
7352/7352 [==============================] - 89s 12ms/step - loss: 0.2028
- acc: 0.9404 - val_loss: 0.4538 - val_acc: 0.8962
Epoch 18/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.1911
- acc: 0.9419 - val_loss: 0.3964 - val_acc: 0.8999
Epoch 19/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1912
- acc: 0.9407 - val_loss: 0.3165 - val_acc: 0.9030
Epoch 20/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1732
- acc: 0.9446 - val_loss: 0.4546 - val_acc: 0.8904
```

```
Epoch 21/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1782
- acc: 0.9444 - val_loss: 0.3346 - val_acc: 0.9063
Epoch 22/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1812
- acc: 0.9418 - val_loss: 0.8164 - val_acc: 0.8582
Epoch 23/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1824
- acc: 0.9426 - val_loss: 0.4240 - val_acc: 0.9036
Epoch 24/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1726
- acc: 0.9429 - val_loss: 0.4067 - val_acc: 0.9148
Epoch 25/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1737
- acc: 0.9411 - val_loss: 0.3396 - val_acc: 0.9074
Epoch 26/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1650
- acc: 0.9461 - val_loss: 0.3806 - val_acc: 0.9019
Epoch 27/30
7352/7352 [==============================] - 89s 12ms/step - loss: 0.1925
- acc: 0.9415 - val_loss: 0.6464 - val_acc: 0.8850
Epoch 28/30
7352/7352 [==============================] - 91s 12ms/step - loss: 0.1965
- acc: 0.9425 - val_loss: 0.3363 - val_acc: 0.9203
Epoch 29/30
7352/7352 [==============================] - 92s 12ms/step - loss: 0.1889
- acc: 0.9431 - val_loss: 0.3737 - val_acc: 0.9158
Epoch 30/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1945
- acc: 0.9414 - val_loss: 0.3088 - val_acc: 0.9097
```

Out[0]:

```
<keras.callbacks.History at 0x29b5ee36a20>
```

In [0]:

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

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS
\
True
LAYING                 512        0        25        0                   0
SITTING                  3      410        75        0                   0
STANDING                 0       87       445        0                   0
WALKING                  0        0         0      481                   2
WALKING_DOWNSTAIRS       0        0         0        0                 382
WALKING_UPSTAIRS         0        0         0        2                  18

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            3
STANDING                           0
WALKING                           13
WALKING_DOWNSTAIRS                38
WALKING_UPSTAIRS                 451
```

In [0]:

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

```
2947/2947 [==============================] - 4s 2ms/step
```

In [0]:

```
score
```

Out[0]:

```
[0.3087582236972612, 0.9097387173396675]
```

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

3.0 Hyperparameter tuning a single layered LSTM using KerasClassifier & Grid search

In [5]:

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

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

```
128
9
7352
```

In [0]:

```
# Credits: https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning
-models-python-keras/
# Function to create model, required for KerasClassifier
def create_model(cells=1,dropout_rate=0.0):
    # create model
    model = Sequential()
    model.add(LSTM(cells, input_shape=(timesteps, input_dim)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(n_classes, activation='sigmoid'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accurac
y'])
    return model
```

In [0]:

```
model = KerasClassifier(build_fn=create_model, epochs=20, batch_size=50, verbose=0)
```

3.1 Grid Search

In [0]:

```
# defining the search parameters
import warnings
warnings.filterwarnings("ignore")
start = datetime.now()
cells=[64,128,150]
dropout_rate = [0.25, 0.35, 0.50]
param_grid = dict(cells= cells, dropout_rate=dropout_rate)
grid = GridSearchCV(estimator=model,param_grid=param_grid,cv=3)
grid_result = grid.fit(X_train, Y_train)
print('Time taken :', datetime.now() - start)
```

```
Time taken : 3:50:57.960481
```

3.2 Best estimator

In [0]:

```
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.653972 using {'cells': 64, 'dropout_rate': 0.35}
```

In [0]:

```
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
0.607726 (0.021996) with: {'cells': 64, 'dropout_rate': 0.25}
0.653972 (0.015658) with: {'cells': 64, 'dropout_rate': 0.35}
0.517818 (0.150945) with: {'cells': 64, 'dropout_rate': 0.5}
0.520947 (0.088254) with: {'cells': 128, 'dropout_rate': 0.25}
0.560800 (0.086814) with: {'cells': 128, 'dropout_rate': 0.35}
0.432127 (0.293107) with: {'cells': 128, 'dropout_rate': 0.5}
0.632345 (0.114650) with: {'cells': 150, 'dropout_rate': 0.25}
0.574129 (0.046813) with: {'cells': 150, 'dropout_rate': 0.35}
0.542029 (0.108520) with: {'cells': 150, 'dropout_rate': 0.5}
```

3.3 3-D Plot to visualize the metric for different values of hyperparameters

In [0]:

```
df=pd.DataFrame(grid.cv_results_)
df.head(2)
```

Out[0]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_cells	paran
0	473.090310	5.601252	5.374086	0.078879	64	0.25
1	476.258571	1.768062	5.696839	0.041143	64	0.35

In [0]:

```
df.to_csv('hyp.csv')
```

In [0]:

```
%matplotlib notebook
%matplotlib inline
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
def enable_plotly_in_cell():
    import IPython
    from plotly.offline import init_notebook_mode
    display(IPython.core.display.HTML('''<script src="/static/components/requirejs/requ
ire.js"></script>'''))
    init_notebook_mode(connected=False)
```

In [ ]:

```
# https://plot.ly/python/3d-axes/
#trace1 = go.Scatter3d(x=df['param_cells'],y=df['param_dropout_rate'],z=df['mean_test_s
core'], name = 'train')
trace2 = go.Scatter3d(x=df['param_cells'],y=df['param_dropout_rate'],z=df['mean_test_sc
ore'], name = 'Cross validation')
data = [trace2]
enable_plotly_in_cell()

layout = go.Layout(scene = dict(
        xaxis = dict(title='Number of LSTM cells'),
        yaxis = dict(title='Drop-out rate'),
        zaxis = dict(title='Accuracy'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

PLot

3.4 Applying the best hyperparameters on the network

In [0]:

```
n_hidden= 64
dropout_rate= 0.35
```

Architecture

In [10]:

```
# Initiliazing the sequential model
model1 = Sequential()

model1.add(LSTM(n_hidden,input_shape=(timesteps, input_dim)))
model1.add(BatchNormalization())
model1.add(Dropout(dropout_rate))

model1.add(Dense(n_classes, activation='sigmoid'))
model1.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_2 (LSTM)                (None, 64)                18944     
_________________________________________________________________
batch_normalization_2 (Batch (None, 64)                256       
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 390       
=================================================================
Total params: 19,590
Trainable params: 19,462
Non-trainable params: 128
_________________________________________________________________
```

In [0]:

```
# Compiling the model
model1.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

3.5 Checkpointing the model and creating the callback list

In [0]:

```
from keras.callbacks import ModelCheckpoint
from keras.callbacks import CSVLogger
import matplotlib.pyplot as plt
from keras.callbacks import TensorBoard
import tensorflow as tf
import datetime
import keras


filepath="weights-{epoch:02d}-{val_accuracy:.2f}.hdf5"
checkpoints = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_on
ly=True, mode='max')
train_results = CSVLogger('train_results_2.log') #storing the training results in a pan
das dataframe
callbacks_list = [checkpoints, train_results]
```

3.6 Fitting the model in batches

In [17]:

```
history= model1.fit(X_train,Y_train,batch_size=50,validation_data=(X_test, Y_test),nb_e
poch=30,verbose=1,
                    callbacks =callbacks_list)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: UserWarnin
g: The `nb_epoch` argument in `fit` has been renamed `epochs`.


Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.8737 -
acc: 0.5903 - val_loss: 0.8794 - val_acc: 0.5148
Epoch 2/30
```

```
/usr/local/lib/python3.6/dist-packages/keras/callbacks.py:707: RuntimeWarn
ing: Can save best model only with val_accuracy available, skipping.
  'skipping.' % (self.monitor), RuntimeWarning)
```

```
7352/7352 [==============================] - 44s 6ms/step - loss: 0.7757 -
acc: 0.6091 - val_loss: 0.8388 - val_acc: 0.6257
Epoch 3/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.7354 -
acc: 0.6138 - val_loss: 0.8674 - val_acc: 0.5589
Epoch 4/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.7585 -
acc: 0.5871 - val_loss: 0.7672 - val_acc: 0.5809
Epoch 5/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.7604 -
acc: 0.5632 - val_loss: 0.8021 - val_acc: 0.5107
Epoch 6/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.7076 -
acc: 0.5717 - val_loss: 0.7230 - val_acc: 0.5701
Epoch 7/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.7145 -
acc: 0.5690 - val_loss: 0.7387 - val_acc: 0.5304
Epoch 8/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.7102 -
acc: 0.5690 - val_loss: 0.7256 - val_acc: 0.5073
Epoch 9/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.6796 -
acc: 0.5846 - val_loss: 0.7081 - val_acc: 0.6091
Epoch 10/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.7015 -
acc: 0.6204 - val_loss: 0.6679 - val_acc: 0.6637
Epoch 11/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.5928 -
acc: 0.6959 - val_loss: 0.6539 - val_acc: 0.6610
Epoch 12/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.6295 -
acc: 0.7047 - val_loss: 1.9109 - val_acc: 0.4917
Epoch 13/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.6305 -
acc: 0.7311 - val_loss: 0.5935 - val_acc: 0.7448
Epoch 14/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.4553 -
acc: 0.8402 - val_loss: 0.4621 - val_acc: 0.8626
Epoch 15/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.2337 -
acc: 0.9241 - val_loss: 0.6692 - val_acc: 0.8544
Epoch 16/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.1950 -
acc: 0.9300 - val_loss: 0.4736 - val_acc: 0.8758
Epoch 17/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.2293 -
acc: 0.9221 - val_loss: 0.5560 - val_acc: 0.8690
Epoch 18/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.2485 -
acc: 0.9144 - val_loss: 0.3738 - val_acc: 0.8907
Epoch 19/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.1847 -
acc: 0.9309 - val_loss: 0.2657 - val_acc: 0.9053
Epoch 20/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.1825 -
acc: 0.9338 - val_loss: 0.2923 - val_acc: 0.9128
Epoch 21/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.1516 -
acc: 0.9411 - val_loss: 0.2962 - val_acc: 0.9111
Epoch 22/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.1465 -
```

```
acc: 0.9436 - val_loss: 0.2487 - val_acc: 0.9074
Epoch 23/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.1433 -
acc: 0.9396 - val_loss: 0.3190 - val_acc: 0.9094
Epoch 24/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.1792 -
acc: 0.9310 - val_loss: 0.2996 - val_acc: 0.9121
Epoch 25/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.1683 -
acc: 0.9353 - val_loss: 0.3410 - val_acc: 0.8819
Epoch 26/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.1682 -
acc: 0.9377 - val_loss: 0.2552 - val_acc: 0.9019
Epoch 27/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.1430 -
acc: 0.9402 - val_loss: 0.2351 - val_acc: 0.9141
Epoch 28/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.1309 -
acc: 0.9444 - val_loss: 0.2480 - val_acc: 0.9040
Epoch 29/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.1266 -
acc: 0.9463 - val_loss: 0.2544 - val_acc: 0.9070
Epoch 30/30
7352/7352 [==============================] - 43s 6ms/step - loss: 0.1200 -
acc: 0.9517 - val_loss: 0.2620 - val_acc: 0.9128
```

3.7 Confusion matrix

In [102]:

```
cm=confusion_matrix(Y_test, model1.predict(X_test))
cm
```

Out[102]:

Pred	LAYING	SITTING	STANDING	WALKING	WALKING_DOWN
True					
LAYING	537	0	0	0	0
SITTING	0	371	116	1	0
STANDING	0	77	452	2	0
WALKING	0	8	0	467	15
WALKING_DOWNSTAIRS	0	0	0	4	412
WALKING_UPSTAIRS	0	0	0	13	7

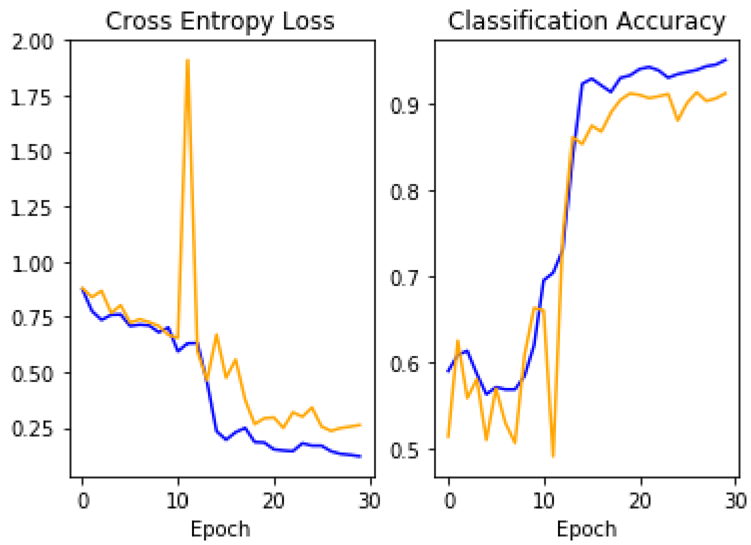
3.8 Plots on training results

In [0]:

```
# function to plot epoch vs loss
%matplotlib notebook
%matplotlib inline
from matplotlib import pyplot
def plot(history):
    # plot loss
    pyplot.subplot(121)
    pyplot.title('Cross Entropy Loss')
    pyplot.xlabel('Epoch')
    pyplot.plot(history.history['loss'], color='blue', label='train')
    pyplot.plot(history.history['val_loss'], color='orange', label='test')
    # plot accuracy
    pyplot.subplot(122)
    pyplot.title('\nClassification Accuracy')
    pyplot.xlabel('Epoch')
    pyplot.plot(history.history['acc'], color='blue', label='train')
    pyplot.plot(history.history['val_acc'], color='orange', label='test')
```

In [41]:

```
plot(history)
```



3.9 Model Testing

In [20]:

```
score = model1.evaluate(X_test, Y_test, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
2947/2947 [==============================] - 10s 3ms/step
Test loss: 0.2620189085359711
Test accuracy: 0.9127926705123854
```

4.0 Deep LSTM model

In [0]:

```
epochs = 50
batch_size= 50
n_hidden1 = 64
n_hidden2 =128
d1 = 0.50
d2 = 0.60 #using higher dropout rates
```

In [0]:

```
import keras.backend as K
K.clear_session()
```

4.1 Architecture

In [126]:

```
# Initiliazing the sequential model
model2 = Sequential()

model2.add(LSTM(n_hidden1,return_sequences=True,input_shape=(timesteps, input_dim)))
model2.add(BatchNormalization())
model2.add(Dropout(d1))

model2.add(LSTM(n_hidden2))
model2.add(BatchNormalization())
model2.add(Dropout(d2))

model2.add(Dense(n_classes, activation='sigmoid'))
model2.summary()
```

```
WARNING:tensorflow:Large dropout rate: 0.6 (>0.5). In TensorFlow 2.x, drop
out() uses dropout rate instead of keep_prob. Please ensure that this is i
ntended.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_1 (LSTM)                (None, 128, 64)           18944     
_________________________________________________________________
batch_normalization_1 (Batch (None, 128, 64)           256       
_________________________________________________________________
dropout_1 (Dropout)          (None, 128, 64)           0         
_________________________________________________________________
lstm_2 (LSTM)                (None, 128)               98816     
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512       
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 774       
=================================================================
Total params: 119,302
Trainable params: 118,918
Non-trainable params: 384
_________________________________________________________________
```

4.2 Compiling

In [0]:

```
# Compiling the model
model2.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

4.3 Checkpointing the model and creating the callback list

In [0]:

```
from keras.callbacks import ModelCheckpoint
from keras.callbacks import CSVLogger
import matplotlib.pyplot as plt
from keras.callbacks import TensorBoard
import tensorflow as tf
import datetime
import keras


filepath='model-ep{epoch:03d}-val_acc{val_acc:.3f}.h5'
checkpoints = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_on
ly=True, mode='max')
train_results = CSVLogger('train_results_model2.log') #storing the training results in
 a pandas dataframe
callbacks_list = [checkpoints, train_results]
```

4.4 Fitting the model in batches

In [129]:

```
# Fitting the model
history1= model2.fit(X_train,Y_train,batch_size=batch_size,validation_data=(X_test, Y_t
est),epochs=epochs)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/50
7352/7352 [==============================] - 88s 12ms/step - loss: 1.0211
- acc: 0.6208 - val_loss: 0.8160 - val_acc: 0.6953
Epoch 2/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.7524
- acc: 0.6862 - val_loss: 0.7849 - val_acc: 0.6661
Epoch 3/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.7182
- acc: 0.6865 - val_loss: 0.7363 - val_acc: 0.7316
Epoch 4/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.6304
- acc: 0.7300 - val_loss: 0.9483 - val_acc: 0.6956
Epoch 5/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.5353
- acc: 0.8070 - val_loss: 0.5818 - val_acc: 0.8368
Epoch 6/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.3677
- acc: 0.8641 - val_loss: 0.4695 - val_acc: 0.8341
Epoch 7/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.2595
- acc: 0.8483 - val_loss: 0.4434 - val_acc: 0.7842
Epoch 8/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.2833
- acc: 0.8303 - val_loss: 0.3670 - val_acc: 0.7920
Epoch 9/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.2319
- acc: 0.8347 - val_loss: 0.4086 - val_acc: 0.7764
Epoch 10/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.2277
- acc: 0.8312 - val_loss: 0.3435 - val_acc: 0.8035
Epoch 11/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.2197
- acc: 0.8402 - val_loss: 0.3575 - val_acc: 0.7978
Epoch 12/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.2174
- acc: 0.8860 - val_loss: 0.3930 - val_acc: 0.9114
Epoch 13/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1803
- acc: 0.9338 - val_loss: 0.4490 - val_acc: 0.8894
Epoch 14/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1561
- acc: 0.9410 - val_loss: 0.4746 - val_acc: 0.8548
Epoch 15/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1666
- acc: 0.9391 - val_loss: 0.2934 - val_acc: 0.9104
Epoch 16/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1702
- acc: 0.9355 - val_loss: 0.3931 - val_acc: 0.8873
Epoch 17/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1906
- acc: 0.9290 - val_loss: 0.3184 - val_acc: 0.9077
Epoch 18/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1631
- acc: 0.9361 - val_loss: 0.2773 - val_acc: 0.9135
Epoch 19/50
7352/7352 [==============================] - 86s 12ms/step - loss: 0.1359
- acc: 0.9407 - val_loss: 0.3139 - val_acc: 0.9060
Epoch 20/50
7352/7352 [==============================] - 86s 12ms/step - loss: 0.1375
- acc: 0.9479 - val_loss: 0.3403 - val_acc: 0.9097
```

```
Epoch 21/50
7352/7352 [==============================] - 86s 12ms/step - loss: 0.1440
- acc: 0.9430 - val_loss: 0.3217 - val_acc: 0.9148
Epoch 22/50
7352/7352 [==============================] - 85s 12ms/step - loss: 0.1313
- acc: 0.9484 - val_loss: 0.3400 - val_acc: 0.9097
Epoch 23/50
7352/7352 [==============================] - 85s 12ms/step - loss: 0.1913
- acc: 0.9340 - val_loss: 0.2570 - val_acc: 0.9186
Epoch 24/50
7352/7352 [==============================] - 86s 12ms/step - loss: 0.1379
- acc: 0.9412 - val_loss: 0.2645 - val_acc: 0.9281
Epoch 25/50
7352/7352 [==============================] - 85s 12ms/step - loss: 0.1649
- acc: 0.9415 - val_loss: 0.2581 - val_acc: 0.9046
Epoch 26/50
7352/7352 [==============================] - 85s 12ms/step - loss: 0.1326
- acc: 0.9478 - val_loss: 0.2355 - val_acc: 0.9355
Epoch 27/50
7352/7352 [==============================] - 86s 12ms/step - loss: 0.1320
- acc: 0.9490 - val_loss: 0.2499 - val_acc: 0.9253
Epoch 28/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1220
- acc: 0.9489 - val_loss: 0.2754 - val_acc: 0.9257
Epoch 29/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1227
- acc: 0.9486 - val_loss: 0.2694 - val_acc: 0.9209
Epoch 30/50
7352/7352 [==============================] - 88s 12ms/step - loss: 0.1287
- acc: 0.9463 - val_loss: 0.2407 - val_acc: 0.9281
Epoch 31/50
7352/7352 [==============================] - 88s 12ms/step - loss: 0.1671
- acc: 0.9306 - val_loss: 0.2330 - val_acc: 0.9175
Epoch 32/50
7352/7352 [==============================] - 88s 12ms/step - loss: 0.1439
- acc: 0.9369 - val_loss: 0.3069 - val_acc: 0.9074
Epoch 33/50
7352/7352 [==============================] - 88s 12ms/step - loss: 0.1429
- acc: 0.9392 - val_loss: 0.3173 - val_acc: 0.9104
Epoch 34/50
7352/7352 [==============================] - 88s 12ms/step - loss: 0.1222
- acc: 0.9506 - val_loss: 0.2809 - val_acc: 0.9318
Epoch 35/50
7352/7352 [==============================] - 88s 12ms/step - loss: 0.1227
- acc: 0.9521 - val_loss: 0.2797 - val_acc: 0.9233
Epoch 36/50
7352/7352 [==============================] - 88s 12ms/step - loss: 0.1186
- acc: 0.9504 - val_loss: 0.3137 - val_acc: 0.9226
Epoch 37/50
7352/7352 [==============================] - 88s 12ms/step - loss: 0.1596
- acc: 0.9354 - val_loss: 0.3006 - val_acc: 0.9128
Epoch 38/50
7352/7352 [==============================] - 88s 12ms/step - loss: 0.1533
- acc: 0.9351 - val_loss: 0.3289 - val_acc: 0.8965
Epoch 39/50
7352/7352 [==============================] - 88s 12ms/step - loss: 0.1540
- acc: 0.9370 - val_loss: 0.2790 - val_acc: 0.9243
Epoch 40/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1287
- acc: 0.9464 - val_loss: 0.2605 - val_acc: 0.9284
Epoch 41/50
```

```
7352/7352 [==============================] - 88s 12ms/step - loss: 0.1227
- acc: 0.9479 - val_loss: 0.2856 - val_acc: 0.9260
Epoch 42/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1214
- acc: 0.9467 - val_loss: 0.3178 - val_acc: 0.9274
Epoch 43/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1218
- acc: 0.9493 - val_loss: 0.3100 - val_acc: 0.9270
Epoch 44/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1222
- acc: 0.9497 - val_loss: 0.3382 - val_acc: 0.9182
Epoch 45/50
7352/7352 [==============================] - 89s 12ms/step - loss: 0.1255
- acc: 0.9509 - val_loss: 0.3199 - val_acc: 0.9230
Epoch 46/50
7352/7352 [==============================] - 89s 12ms/step - loss: 0.1120
- acc: 0.9532 - val_loss: 0.3275 - val_acc: 0.9213
Epoch 47/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1225
- acc: 0.9487 - val_loss: 0.3052 - val_acc: 0.9247
Epoch 48/50
7352/7352 [==============================] - 88s 12ms/step - loss: 0.1304
- acc: 0.9421 - val_loss: 0.3078 - val_acc: 0.9165
Epoch 49/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1237
- acc: 0.9484 - val_loss: 0.3364 - val_acc: 0.9186
Epoch 50/50
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1196
- acc: 0.9524 - val_loss: 0.3126 - val_acc: 0.9308
```

4.5 Confusion matrix

In [132]:

```
cm1= confusion_matrix(Y_test, model2.predict(X_test))
cm1
```

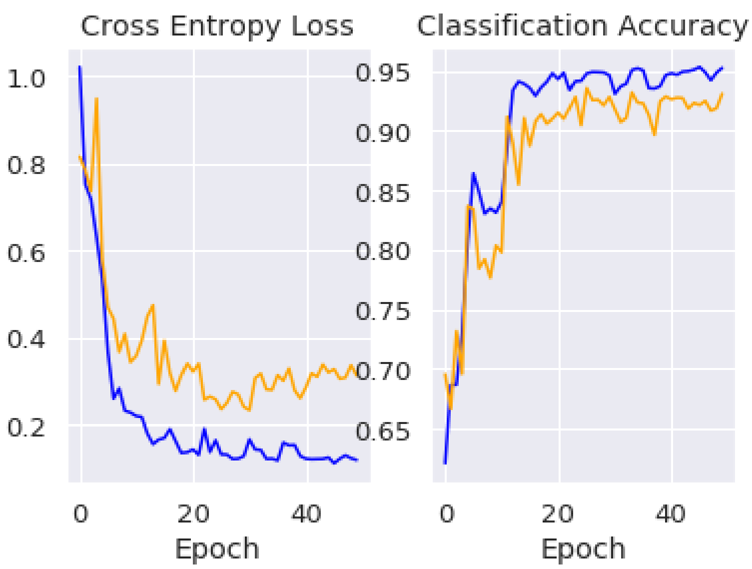
Out[132]:

Pred	LAYING	SITTING	STANDING	WALKING	WALKING_DOWN
True					
LAYING	537	0	0	0	0
SITTING	0	370	118	0	0
STANDING	0	50	482	0	0
WALKING	0	0	0	466	27
WALKING_DOWNSTAIRS	0	0	0	1	418
WALKING_UPSTAIRS	0	0	0	1	0

4.6 Plots on training results

In [134]:

```
plot(history1)
```



4.7 Model Testing

In [135]:

```
score = model2.evaluate(X_test, Y_test, verbose=1)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
2947/2947 [==============================] - 19s 6ms/step
Test loss: 0.31255650963575987
Test accuracy: 0.9307770614183916
```

5.0 Summary

In [3]:

```
#Ref: http://zetcode.com/python/prettytable/

from prettytable import PrettyTable
x=PrettyTable()
x.field_names=["Model","Test loss","Test accuracy"]

x.add_row(["1 layered LSTM without hyp tuning","0.3088","90.97%"])
x.add_row(["1 layered LSTM with hyp tuning","0.2620","91.30%"])
x.add_row(["Deep 2 layered LSTM","0.3126","93.08%"])

print(x)
```

```
+-----------------------------------+-----------+---------------+
|               Model               | Test loss | Test accuracy |
+-----------------------------------+-----------+---------------+
| 1 layered LSTM without hyp tuning |   0.3088  |     90.97%    |
|   1 layered LSTM with hyp tuning  |   0.2620  |     91.30%    |
|        Deep 2 layered LSTM        |   0.3126  |     93.08%    |
+-----------------------------------+-----------+---------------+
```