HumanActivityRecognition

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

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

How data was recorded

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

> prefix 't' in those metrics denotes time.

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

Feature names

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

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

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

   - tBodyAcc-XYZ

- tGravityAcc-XYZ
- tBodyAccJerk-XYZ
- tBodyGyro-XYZ
- tBodyGyroJerk-XYZ
- tBodyAccMag
- tGravityAccMag
- tBodyAccJerkMag
- tBodyGyroMag
- tBodyGyroJerkMag
- fBodyAcc-XYZ
- fBodyAccJerk-XYZ
- fBodyGyro-XYZ
- fBodyAccMag
- fBodyAccJerkMag
- fBodyGyroMag
- fBodyGyroJerkMag

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

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

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

    - gravityMean
    - tBodyAccMean
    - tBodyAccJerkMean
    - tBodyGyroMean
    - tBodyGyroJerkMean

Y_Labels(Encoded)

- In the dataset, Y_labels are represented as numbers from 1 to 6 as their identifiers.

- WALKING as **1**
- WALKING_UPSTAIRS as **2**
- WALKING_DOWNSTAIRS as **3**
- SITTING as **4**
- STANDING as **5**
- LAYING as **6**

Train and test data were saperated

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

Data

- Data set can be found here [https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones (https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones)](https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones)

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]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly (https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly)

Enter your authorization code:
··········
Mounted at /content/drive
```

In [2]:
```
import numpy as np
import pandas as pd

# get the features from the file features.txt
features = list()
with open('/content/drive/My Drive/HAR/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
```

In [7]:
```
print(features)
```

```
yro-entropy()-X', 'tBodyGyro-entropy()-Y', 'tBodyGyro-entropy()-Z', 'tBodyGyr
o-arCoeff()-X,1', 'tBodyGyro-arCoeff()-X,2', 'tBodyGyro-arCoeff()-X,3', 'tBod
yGyro-arCoeff()-X,4', 'tBodyGyro-arCoeff()-Y,1', 'tBodyGyro-arCoeff()-Y,2',
'tBodyGyro-arCoeff()-Y,3', 'tBodyGyro-arCoeff()-Y,4', 'tBodyGyro-arCoeff()-Z,
1', 'tBodyGyro-arCoeff()-Z,2', 'tBodyGyro-arCoeff()-Z,3', 'tBodyGyro-arCoeff
()-Z,4', 'tBodyGyro-correlation()-X,Y', 'tBodyGyro-correlation()-X,Z', 'tBody
Gyro-correlation()-Y,Z', 'tBodyGyroJerk-mean()-X', 'tBodyGyroJerk-mean()-Y',
'tBodyGyroJerk-mean()-Z', 'tBodyGyroJerk-std()-X', 'tBodyGyroJerk-std()-Y',
'tBodyGyroJerk-std()-Z', 'tBodyGyroJerk-mad()-X', 'tBodyGyroJerk-mad()-Y', 't
BodyGyroJerk-mad()-Z', 'tBodyGyroJerk-max()-X', 'tBodyGyroJerk-max()-Y', 'tBo
dyGyroJerk-max()-Z', 'tBodyGyroJerk-min()-X', 'tBodyGyroJerk-min()-Y', 'tBody
GyroJerk-min()-Z', 'tBodyGyroJerk-sma()', 'tBodyGyroJerk-energy()-X', 'tBodyG
yroJerk-energy()-Y', 'tBodyGyroJerk-energy()-Z', 'tBodyGyroJerk-iqr()-X', 'tB
odyGyroJerk-iqr()-Y', 'tBodyGyroJerk-iqr()-Z', 'tBodyGyroJerk-entropy()-X',
'tBodyGyroJerk-entropy()-Y', 'tBodyGyroJerk-entropy()-Z', 'tBodyGyroJerk-arCo
eff()-X,1', 'tBodyGyroJerk-arCoeff()-X,2', 'tBodyGyroJerk-arCoeff()-X,3', 'tB
odyGyroJerk-arCoeff()-X,4', 'tBodyGyroJerk-arCoeff()-Y,1', 'tBodyGyroJerk-arC
oeff()-Y,2', 'tBodyGyroJerk-arCoeff()-Y,3', 'tBodyGyroJerk-arCoeff()-Y,4', 't
BodyGyroJerk-arCoeff()-Z,1', 'tBodyGyroJerk-arCoeff()-Z,2', 'tBodyGyroJerk-ar
Coeff()-Z,3', 'tBodyGyroJerk-arCoeff()-Z,4', 'tBodyGyroJerk-correlation()-X,
```

Obtain the train data

In [0]:
```
# get the data from txt files to pandas dataffame
X_train = pd.read_csv("/content/drive/My Drive/HAR/UCI_HAR_dataset/train/X_train.
```

In [0]:
```
# add subject column to the dataframe
X_train['subject'] = pd.read_csv('/content/drive/My Drive/HAR/UCI_HAR_dataset/tra

y_train = pd.read_csv('/content/drive/My Drive/HAR/UCI_HAR_dataset/train/y_train.
y_train_labels = y_train.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNS
                       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()
```

In [0]:
```
train.shape
```

Out[3]:
```
(7352, 564)
```

Obtain the test data

In [0]:
```
# get the data from txt files to pandas dataffame
X_test = pd.read_csv('UCI_HAR_dataset/test/X_test.txt', delim_whitespace=True, he

# add subject column to the dataframe
X_test['subject'] = pd.read_csv('UCI_HAR_dataset/test/subject_test.txt', header=N

# get y labels from the txt file
y_test = pd.read_csv('UCI_HAR_dataset/test/y_test.txt', names=['Activity'], squee
y_test_labels = y_test.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTA
                       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: UserWarning:
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	tBodyAcc-mad()-X	tBodyAcc-mad()-Y
2261	0.279196	-0.018261	-0.103376	-0.996955	-0.982959	-0.988239	-0.9972	-0.982509

1 rows × 564 columns

In [0]:
```
test.shape
```

Out[5]:
```
(2947, 564)
```

Data Cleaning

1. Check for Duplicates

In [0]:
```
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

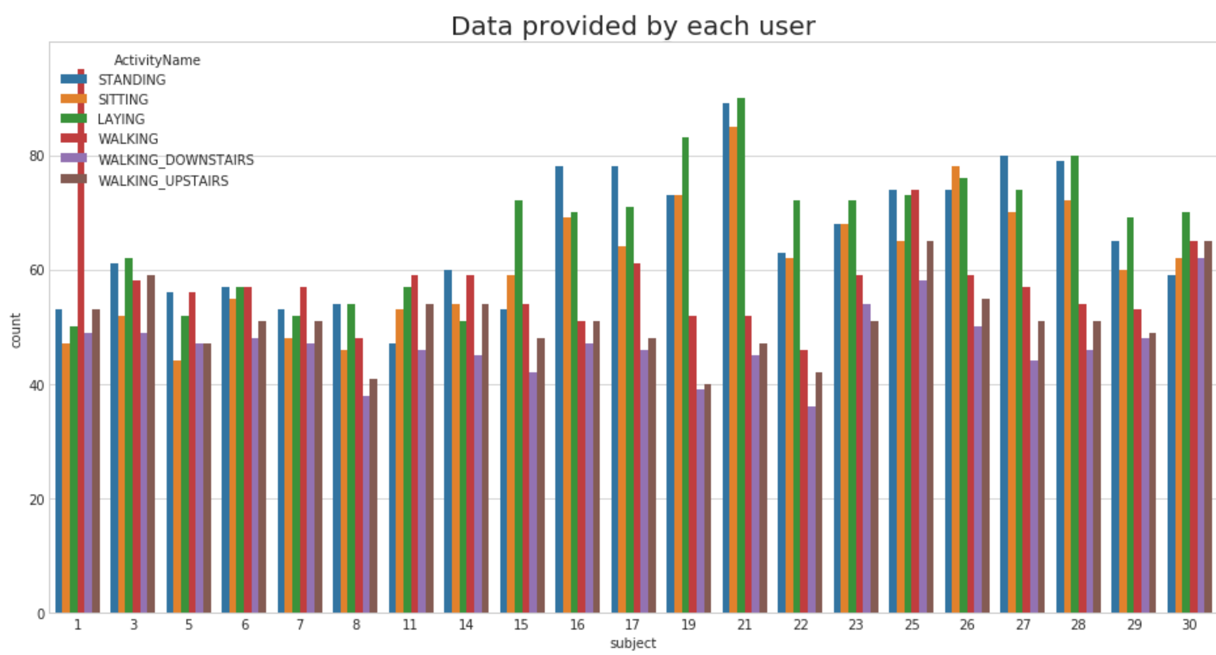
In [0]:
```
print('We have {} NaN/Null values in train'.format(train.isnull().values.sum()))
print('We have {} NaN/Null values in test'.format(test.isnull().values.sum()))
```

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

3. Check for data imbalance

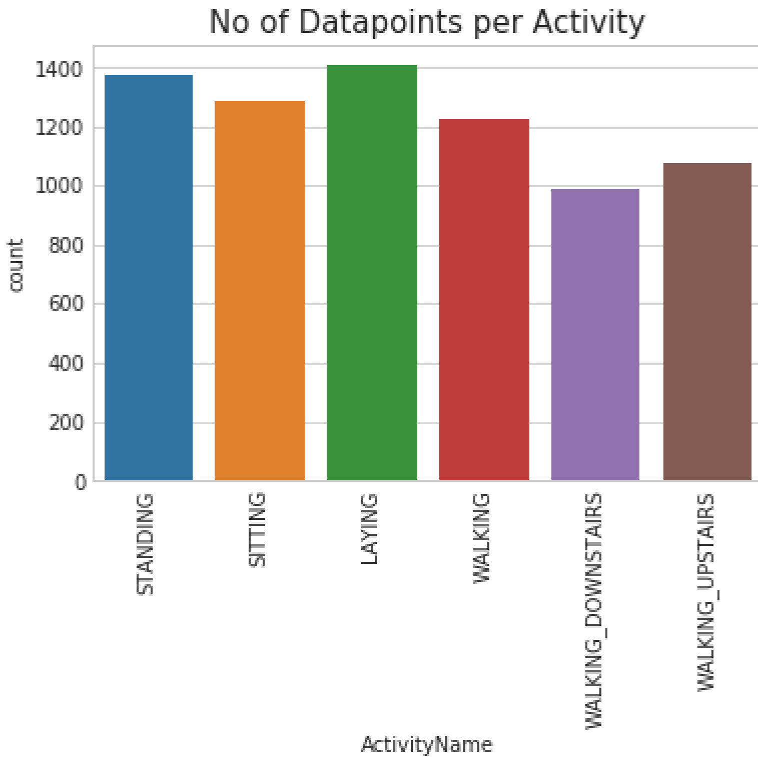
In [0]:
```
import matplotlib.pyplot as plt
import seaborn as sns

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

In [0]:
```
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 [0]:
```
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 [0]:
```
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', 'angletBodyGyroJerkMeangravityMean',
       'angleXgravityMean', 'angleYgravityMean', 'angleZgravityMean',
       'subject', 'Activity', 'ActivityName'],
      dtype='object', length=564)
```

5. Save this dataframe in a csv files

In [0]:
```
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

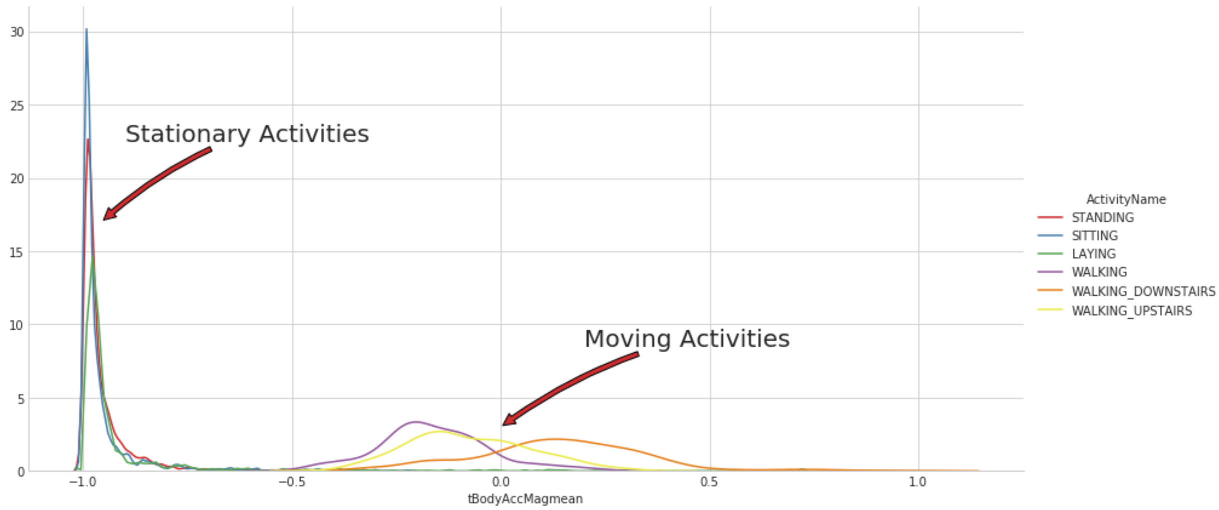
"***Without domain knowledge EDA has no meaning, without EDA a problem has no soul.***"

1. Featuring Engineering from Domain Knowledge

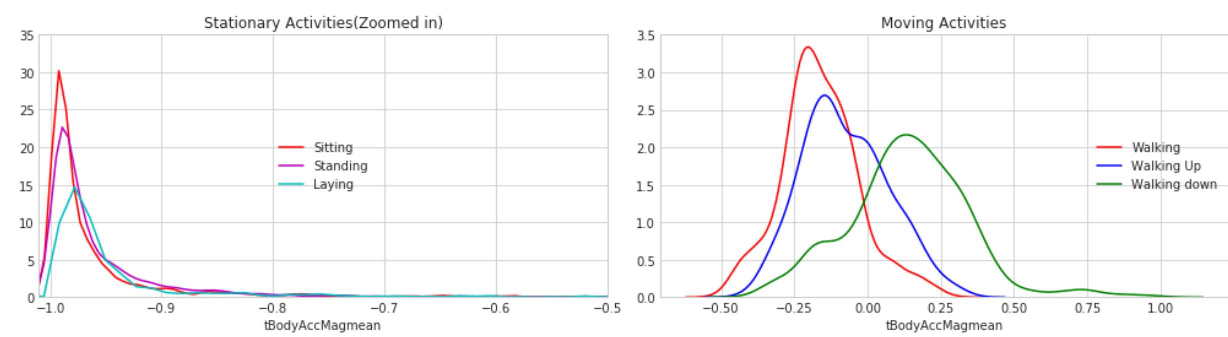
- **Static and Dynamic Activities**
  - In static activities (sit, stand, lie down) motion information will not be very useful.
  - In the dynamic activities (Walking, WalkingUpstairs,WalkingDownstairs) motion info will be significant.

2. Stationary and Moving activities are completely different

In [0]:
```
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()
```

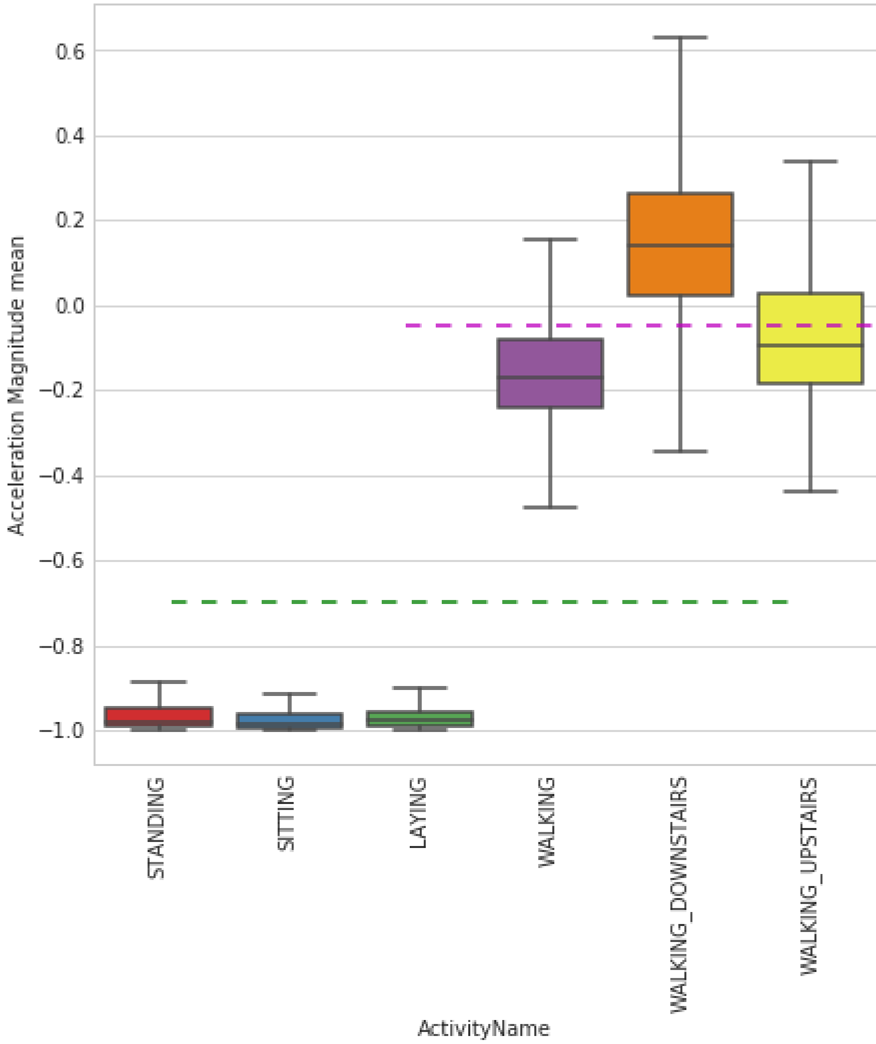


In [0]:
```
# for plotting purposes taking datapoints of each activity to a different datafra
df1 = train[train['Activity']==1]
df2 = train[train['Activity']==2]
df3 = train[train['Activity']==3]
df4 = train[train['Activity']==4]
df5 = train[train['Activity']==5]
df6 = train[train['Activity']==6]

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
sns.distplot(df3['tBodyAccMagmean'],color = 'green',hist = False, label = 'Walkin
plt.legend(loc='center right')


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



3. Magnitude of an acceleration can saperate it well

In [0]:
```
plt.figure(figsize=(7,7))
sns.boxplot(x='ActivityName', y='tBodyAccMagmean',data=train, showfliers=False, s
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()
```

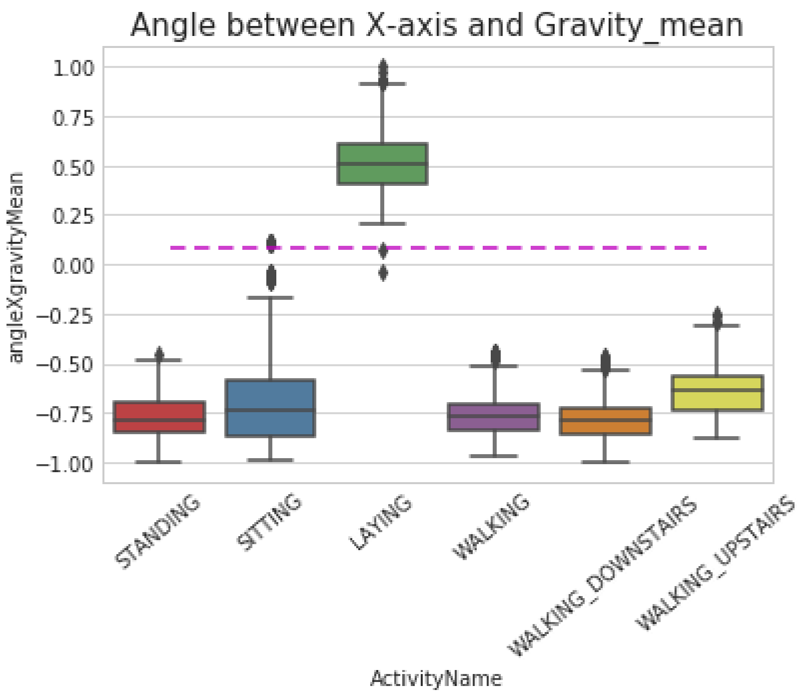


__ Observations__:

- 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

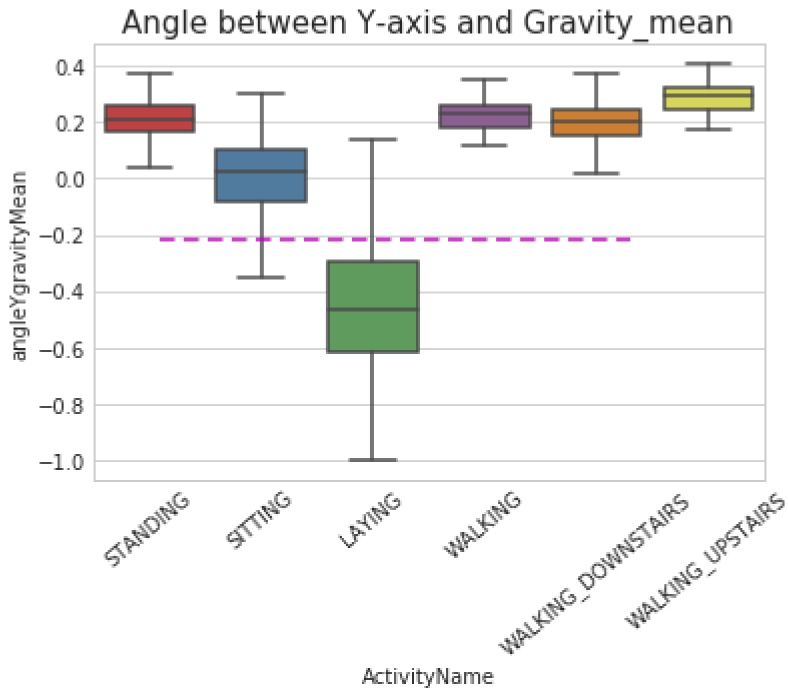
In [0]:
```
sns.boxplot(x='ActivityName', y='angleXgravityMean', data=train)
plt.axhline(y=0.08, xmin=0.1, xmax=0.9,c='m',dashes=(5,3))
plt.title('Angle between X-axis and Gravity_mean', fontsize=15)
plt.xticks(rotation = 40)
plt.show()
```



__ 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 [0]:
```
sns.boxplot(x='ActivityName', y='angleYgravityMean', data = train, showfliers=Fal
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

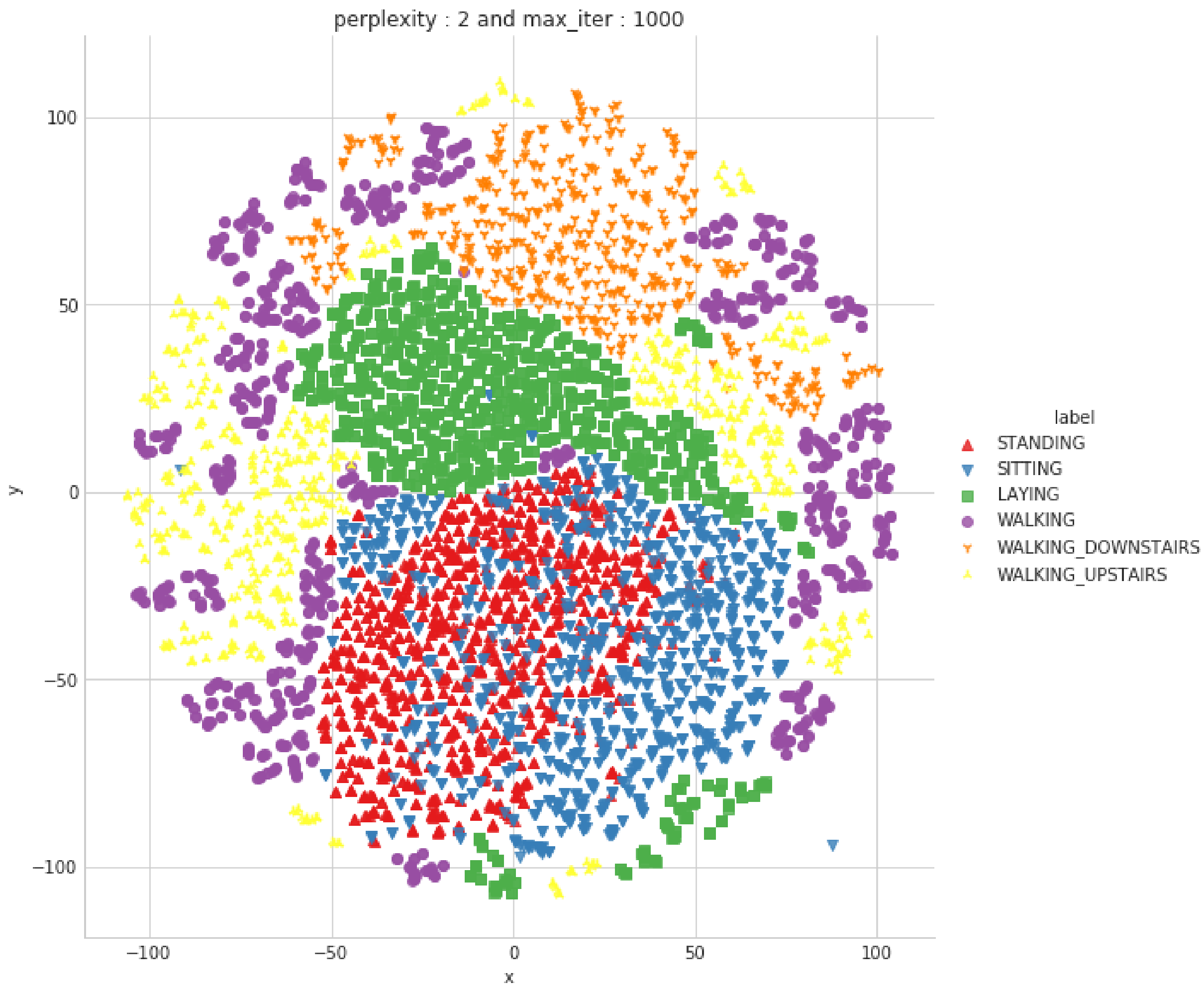
In [0]:
```
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns
```

In [0]:
```
# 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-sn

    for index,perplexity in enumerate(perplexities):
        # perform t-sne
        print('\nperforming tsne with perplexity {} and with {} iterations at max
        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_
        print('saving this plot as image in present working directory...')
        plt.savefig(img_name)
        plt.show()
        print('Done')
```

In [0]:
```
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])
```

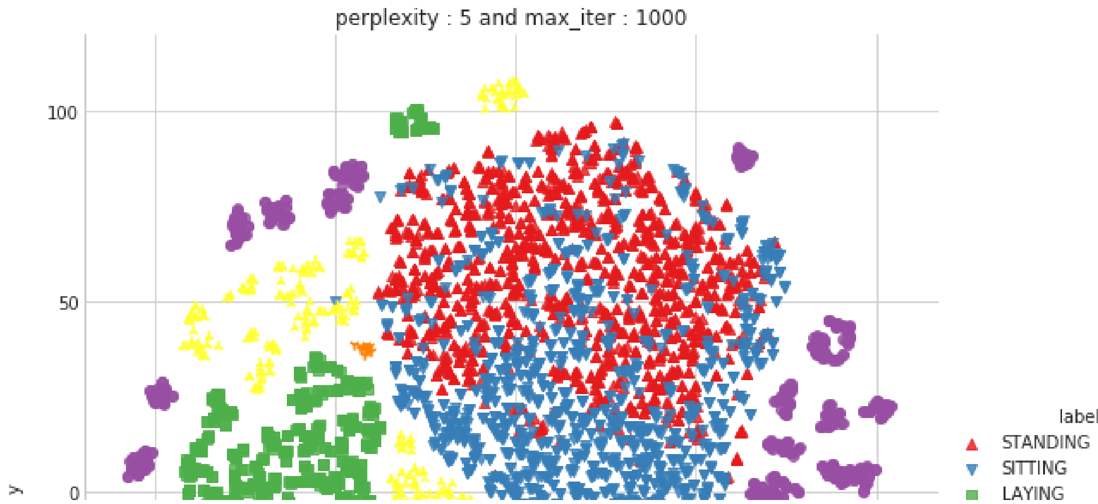
```
performing tsne with perplexity 2 and with 1000 iterations at max
[t-SNE] Computing 7 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.426s...
[t-SNE] Computed neighbors for 7352 samples in 72.001s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional 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 iterat
ions in 16.625s)
[t-SNE] Iteration 100: error = 107.2019501, gradient norm = 0.0284782 (50 itera
tions in 9.735s)
[t-SNE] Iteration 150: error = 100.9872894, gradient norm = 0.0185151 (50 itera
tions in 5.346s)
[t-SNE] Iteration 200: error = 97.6054382, gradient norm = 0.0142084 (50 iterat
ions in 7.013s)
[t-SNE] Iteration 250: error = 95.3084183, gradient norm = 0.0132592 (50 iterat
ions in 5.703s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 95.308418
[t-SNE] Iteration 300: error = 4.1209540, gradient norm = 0.0015668 (50 iterati
ons in 7.156s)
[t-SNE] Iteration 350: error = 3.2113254, gradient norm = 0.0009953 (50 iterati
ons in 8.022s)
[t-SNE] Iteration 400: error = 2.7819963, gradient norm = 0.0007203 (50 iterati
ons in 9.419s)
[t-SNE] Iteration 450: error = 2.5178111, gradient norm = 0.0005655 (50 iterati
ons in 9.370s)
[t-SNE] Iteration 500: error = 2.3341548, gradient norm = 0.0004804 (50 iterati
ons in 7.681s)
[t-SNE] Iteration 550: error = 2.1961622, gradient norm = 0.0004183 (50 iterati
ons in 7.097s)
[t-SNE] Iteration 600: error = 2.0867445, gradient norm = 0.0003664 (50 iterati
ons in 9.274s)
[t-SNE] Iteration 650: error = 1.9967778, gradient norm = 0.0003279 (50 iterati
ons in 7.697s)
[t-SNE] Iteration 700: error = 1.9210005, gradient norm = 0.0002984 (50 iterati
ons in 8.174s)
[t-SNE] Iteration 750: error = 1.8558111, gradient norm = 0.0002776 (50 iterati
ons in 9.747s)
[t-SNE] Iteration 800: error = 1.7989457, gradient norm = 0.0002569 (50 iterati
ons in 8.687s)
[t-SNE] Iteration 850: error = 1.7490212, gradient norm = 0.0002394 (50 iterati
ons in 8.407s)
[t-SNE] Iteration 900: error = 1.7043383, gradient norm = 0.0002224 (50 iterati
ons in 8.351s)
[t-SNE] Iteration 950: error = 1.6641431, gradient norm = 0.0002098 (50 iterati
```

```
ons in 7.841s)
[t-SNE] Iteration 1000: error = 1.6279151, gradient norm = 0.0001989 (50 iterat
ions 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...
```



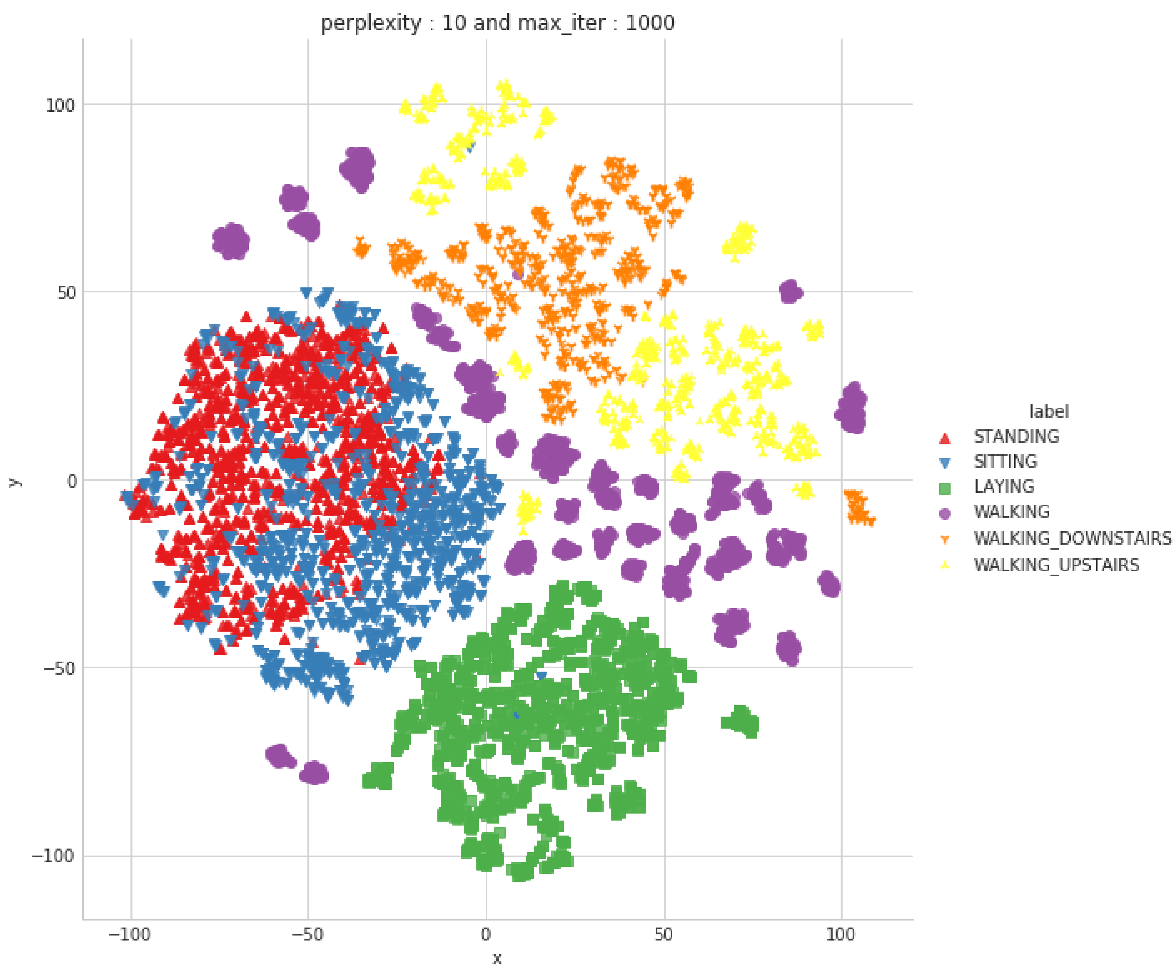

```
Done

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 iterat
ions in 55.655s)
[t-SNE] Iteration 100: error = 97.6535568, gradient norm = 0.0174309 (50 iterat
```

```
ions in 12.580s)
[t-SNE] Iteration 150: error = 93.1900101, gradient norm = 0.0101048 (50 iterat
ions in 9.180s)
[t-SNE] Iteration 200: error = 91.2315445, gradient norm = 0.0074560 (50 iterat
ions in 10.340s)
[t-SNE] Iteration 250: error = 90.0714417, gradient norm = 0.0057667 (50 iterat
ions in 9.458s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 90.071442
[t-SNE] Iteration 300: error = 3.5796804, gradient norm = 0.0014691 (50 iterati
ons in 8.718s)
[t-SNE] Iteration 350: error = 2.8173938, gradient norm = 0.0007508 (50 iterati
ons in 10.180s)
[t-SNE] Iteration 400: error = 2.4344938, gradient norm = 0.0005251 (50 iterati
ons in 10.506s)
[t-SNE] Iteration 450: error = 2.2156141, gradient norm = 0.0004069 (50 iterati
ons in 10.072s)
[t-SNE] Iteration 500: error = 2.0703306, gradient norm = 0.0003340 (50 iterati
ons in 10.511s)
[t-SNE] Iteration 550: error = 1.9646366, gradient norm = 0.0002816 (50 iterati
ons in 9.792s)
[t-SNE] Iteration 600: error = 1.8835558, gradient norm = 0.0002471 (50 iterati
ons in 9.098s)
[t-SNE] Iteration 650: error = 1.8184001, gradient norm = 0.0002184 (50 iterati
ons in 8.656s)
[t-SNE] Iteration 700: error = 1.7647167, gradient norm = 0.0001961 (50 iterati
ons in 9.063s)
[t-SNE] Iteration 750: error = 1.7193680, gradient norm = 0.0001796 (50 iterati
ons in 9.754s)
[t-SNE] Iteration 800: error = 1.6803776, gradient norm = 0.0001655 (50 iterati
ons in 9.540s)
[t-SNE] Iteration 850: error = 1.6465144, gradient norm = 0.0001538 (50 iterati
ons in 9.953s)
[t-SNE] Iteration 900: error = 1.6166563, gradient norm = 0.0001421 (50 iterati
ons in 10.270s)
[t-SNE] Iteration 950: error = 1.5901035, gradient norm = 0.0001335 (50 iterati
ons in 6.609s)
[t-SNE] Iteration 1000: error = 1.5664237, gradient norm = 0.0001257 (50 iterat
ions 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...
```



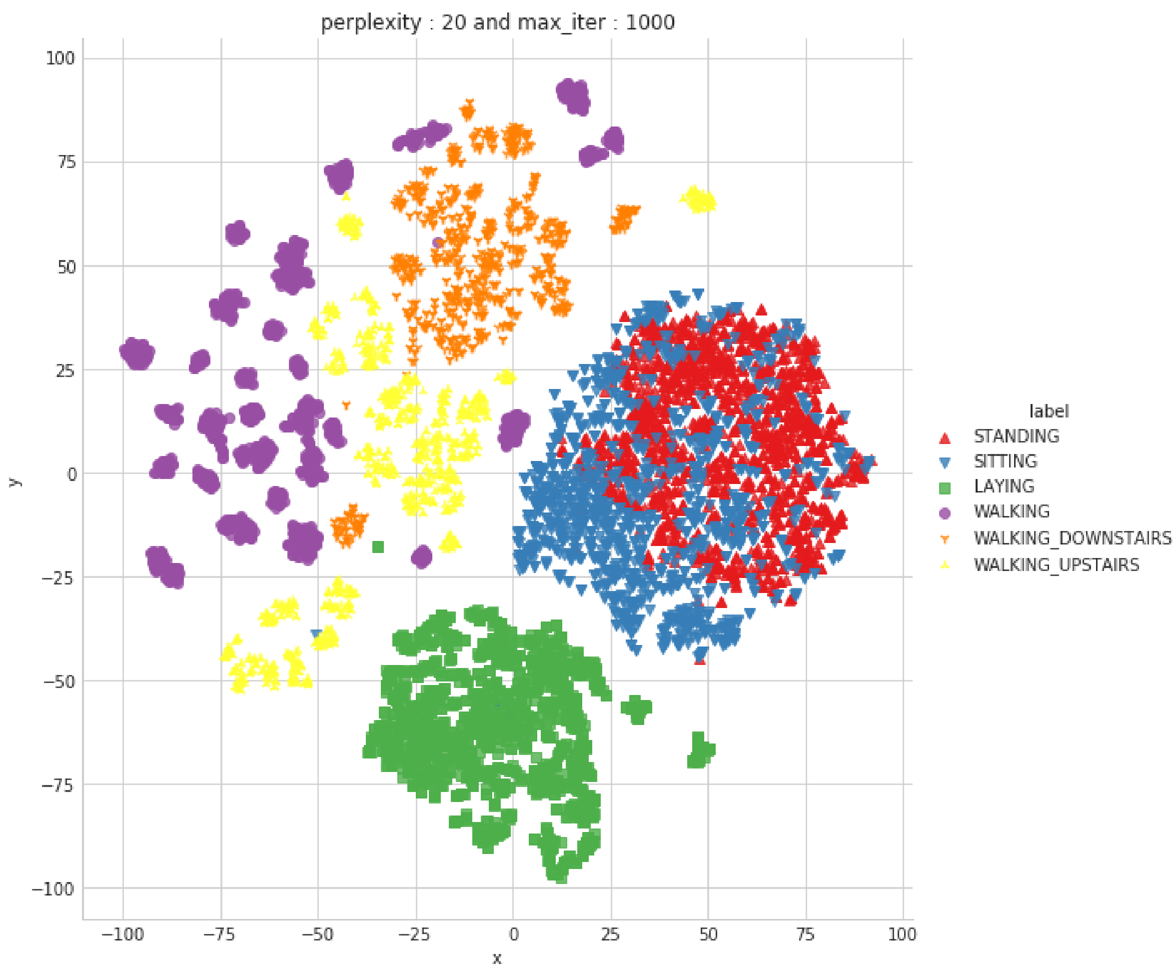

```
Done

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 iterat
ions in 24.550s)
[t-SNE] Iteration 100: error = 90.3036194, gradient norm = 0.0097653 (50 iterat
ions in 11.936s)
[t-SNE] Iteration 150: error = 87.3132935, gradient norm = 0.0053059 (50 iterat
ions in 11.246s)
[t-SNE] Iteration 200: error = 86.1169128, gradient norm = 0.0035844 (50 iterat
ions in 11.864s)
[t-SNE] Iteration 250: error = 85.4133606, gradient norm = 0.0029100 (50 iterat
ions in 11.944s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.413361
[t-SNE] Iteration 300: error = 3.1394315, gradient norm = 0.0013976 (50 iterati
ons in 11.742s)
[t-SNE] Iteration 350: error = 2.4929206, gradient norm = 0.0006466 (50 iterati
ons in 11.627s)
[t-SNE] Iteration 400: error = 2.1733041, gradient norm = 0.0004230 (50 iterati
ons in 11.846s)
[t-SNE] Iteration 450: error = 1.9884514, gradient norm = 0.0003124 (50 iterati
ons in 11.405s)
[t-SNE] Iteration 500: error = 1.8702440, gradient norm = 0.0002514 (50 iterati
ons in 11.320s)
[t-SNE] Iteration 550: error = 1.7870129, gradient norm = 0.0002107 (50 iterati
ons in 12.009s)
[t-SNE] Iteration 600: error = 1.7246909, gradient norm = 0.0001824 (50 iterati
ons in 10.632s)
[t-SNE] Iteration 650: error = 1.6758548, gradient norm = 0.0001590 (50 iterati
```

```
ons in 11.270s)
[t-SNE] Iteration 700: error = 1.6361949, gradient norm = 0.0001451 (50 iterati
ons in 12.072s)
[t-SNE] Iteration 750: error = 1.6034756, gradient norm = 0.0001305 (50 iterati
ons in 11.607s)
[t-SNE] Iteration 800: error = 1.5761518, gradient norm = 0.0001188 (50 iterati
ons in 9.409s)
[t-SNE] Iteration 850: error = 1.5527289, gradient norm = 0.0001113 (50 iterati
ons in 8.309s)
[t-SNE] Iteration 900: error = 1.5328671, gradient norm = 0.0001021 (50 iterati
ons in 9.433s)
[t-SNE] Iteration 950: error = 1.5152045, gradient norm = 0.0000974 (50 iterati
ons in 11.488s)
[t-SNE] Iteration 1000: error = 1.4999681, gradient norm = 0.0000933 (50 iterat
ions 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...
```



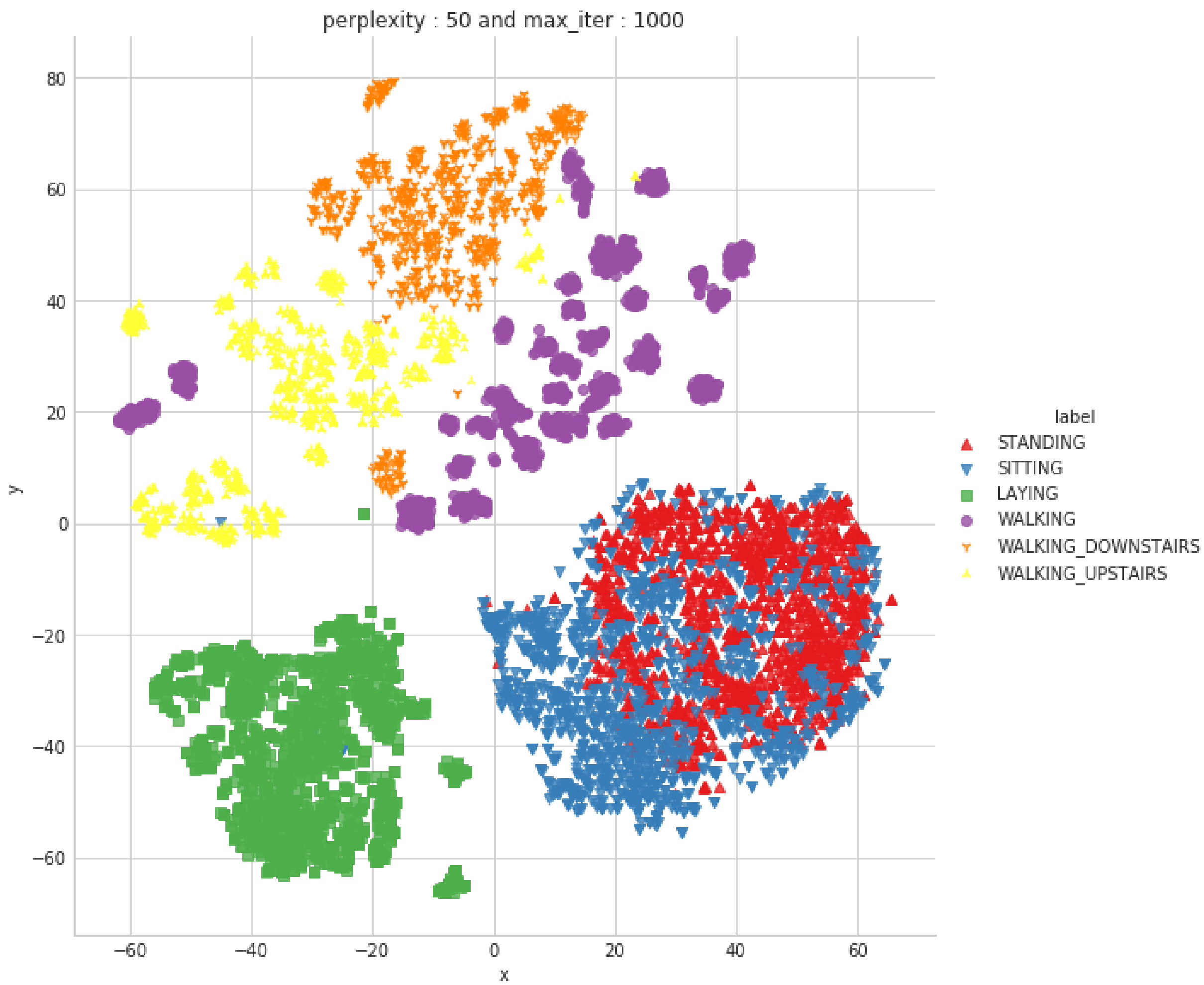
```
Done

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 itera
tions in 21.168s)
[t-SNE] Iteration 100: error = 83.9500732, gradient norm = 0.0059110 (50 iter
ations in 17.306s)
[t-SNE] Iteration 150: error = 81.8804779, gradient norm = 0.0035797 (50 iter
ations in 14.258s)
[t-SNE] Iteration 200: error = 81.1615143, gradient norm = 0.0022536 (50 iter
ations in 14.130s)
[t-SNE] Iteration 250: error = 80.7704086, gradient norm = 0.0018108 (50 iter
ations in 15.340s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 80.770409
[t-SNE] Iteration 300: error = 2.6957574, gradient norm = 0.0012993 (50 itera
tions in 13.605s)
[t-SNE] Iteration 350: error = 2.1637220, gradient norm = 0.0005765 (50 itera
tions in 13.248s)
[t-SNE] Iteration 400: error = 1.9143614, gradient norm = 0.0003474 (50 itera
tions in 14.774s)
[t-SNE] Iteration 450: error = 1.7684202, gradient norm = 0.0002458 (50 itera
tions in 15.502s)
[t-SNE] Iteration 500: error = 1.6744757, gradient norm = 0.0001923 (50 itera
tions in 14.808s)
[t-SNE] Iteration 550: error = 1.6101606, gradient norm = 0.0001575 (50 itera
tions in 14.043s)
[t-SNE] Iteration 600: error = 1.5641028, gradient norm = 0.0001344 (50 itera
tions in 15.769s)
[t-SNE] Iteration 650: error = 1.5291905, gradient norm = 0.0001182 (50 itera
tions in 15.834s)
[t-SNE] Iteration 700: error = 1.5024391, gradient norm = 0.0001055 (50 itera
tions in 15.398s)
[t-SNE] Iteration 750: error = 1.4809053, gradient norm = 0.0000965 (50 itera
tions in 14.594s)
[t-SNE] Iteration 800: error = 1.4631859, gradient norm = 0.0000884 (50 itera
tions in 15.025s)
[t-SNE] Iteration 850: error = 1.4486470, gradient norm = 0.0000832 (50 itera
tions in 14.060s)
```

```
[t-SNE] Iteration 900: error = 1.4367288, gradient norm = 0.0000804 (50 itera
tions in 12.389s)
[t-SNE] Iteration 950: error = 1.4270191, gradient norm = 0.0000761 (50 itera
tions in 10.392s)
[t-SNE] Iteration 1000: error = 1.4189968, gradient norm = 0.0000787 (50 iter
ations 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 itera
tions in 36.249s)
[t-SNE] Iteration 100: error = 75.9874649, gradient norm = 0.0061005 (50 iter
ations in 30.453s)
[t-SNE] Iteration 150: error = 74.7072296, gradient norm = 0.0024708 (50 iter
ations in 28.461s)
[t-SNE] Iteration 200: error = 74.2736282, gradient norm = 0.0018644 (50 iter
ations in 27.735s)
[t-SNE] Iteration 250: error = 74.0722427, gradient norm = 0.0014078 (50 iter
ations in 26.835s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 74.072243
[t-SNE] Iteration 300: error = 2.1539080, gradient norm = 0.0011796 (50 itera
tions in 25.445s)
[t-SNE] Iteration 350: error = 1.7567128, gradient norm = 0.0004845 (50 itera
tions in 21.282s)
[t-SNE] Iteration 400: error = 1.5888531, gradient norm = 0.0002798 (50 itera
tions in 21.015s)
[t-SNE] Iteration 450: error = 1.4956820, gradient norm = 0.0001894 (50 itera
tions in 23.332s)
[t-SNE] Iteration 500: error = 1.4359720, gradient norm = 0.0001420 (50 itera
tions in 23.083s)
[t-SNE] Iteration 550: error = 1.3947564, gradient norm = 0.0001117 (50 itera
tions in 19.626s)
[t-SNE] Iteration 600: error = 1.3653858, gradient norm = 0.0000949 (50 itera
tions in 22.752s)
[t-SNE] Iteration 650: error = 1.3441534, gradient norm = 0.0000814 (50 itera
tions in 23.972s)
[t-SNE] Iteration 700: error = 1.3284039, gradient norm = 0.0000742 (50 itera
tions in 20.636s)
[t-SNE] Iteration 750: error = 1.3171139, gradient norm = 0.0000700 (50 itera
tions in 20.407s)
[t-SNE] Iteration 800: error = 1.3085558, gradient norm = 0.0000657 (50 itera
```

```
tions in 24.951s)
[t-SNE] Iteration 850: error = 1.3017821, gradient norm = 0.0000603 (50 itera
tions in 24.719s)
[t-SNE] Iteration 900: error = 1.2962619, gradient norm = 0.0000586 (50 itera
tions in 24.500s)
[t-SNE] Iteration 950: error = 1.2914882, gradient norm = 0.0000573 (50 itera
tions in 24.132s)
[t-SNE] Iteration 1000: error = 1.2874244, gradient norm = 0.0000546 (50 iter
ations 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
```

In [0]:
```python
import numpy as np
import pandas as pd
```

Obtain the train and test data

In [0]:
```
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 [0]:
```
train.head(3)
```

Out[3]:

	tBodyAccmeanX	tBodyAccmeanY	tBodyAccmeanZ	tBodyAccstdX	tBodyAccstdY	tBodyAccstdZ
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944

3 rows × 564 columns

In [0]:
```
# get X_train and y_train from csv files
X_train = train.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_train = train.ActivityName
```

In [0]:
```
# get X_test and y_test from test csv file
X_test = test.drop(['subject', 'Activity', 'ActivityName'], axis=1)
y_test = test.ActivityName
```

In [0]:
```
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 [0]:
```
labels=['LAYING', 'SITTING','STANDING','WALKING','WALKING_DOWNSTAIRS','WALKING_UP
```

Function to plot the confusion matrix

In [0]:
```
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 [0]:
```
from datetime import datetime
def perform_model(model, X_train, y_train, X_test, y_test, class_labels, cm_norma
                  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='Normal
    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 [0]:
```
def print_grid_search_attributes(model):
    # Estimator that gave highest score among all the estimators formed in GridSe
    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'.forr
```

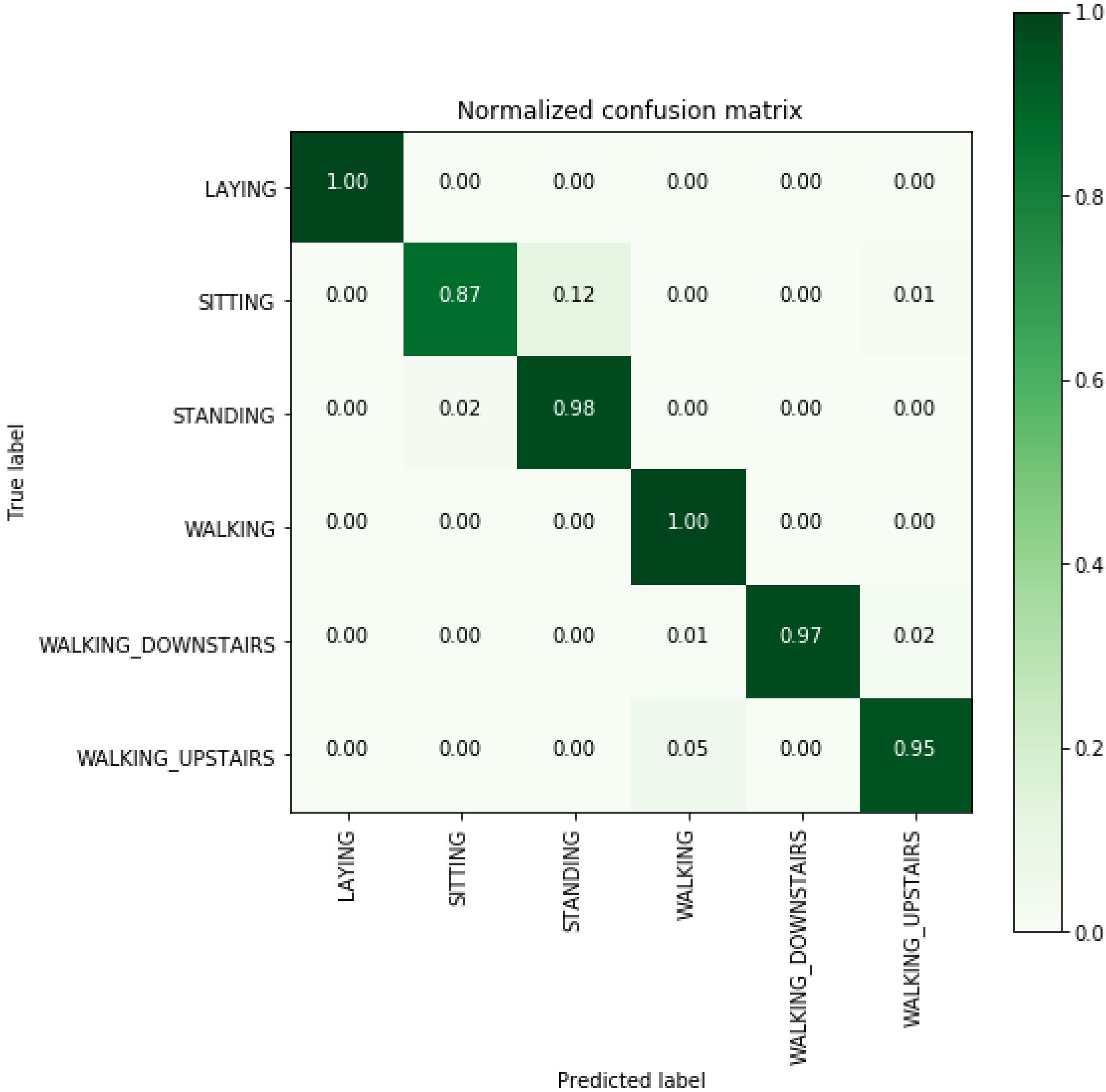
1. Logistic Regression with Grid Search

In [0]:
```
from sklearn import linear_model
from sklearn import metrics

from sklearn.model_selection import GridSearchCV
```

In [0]:

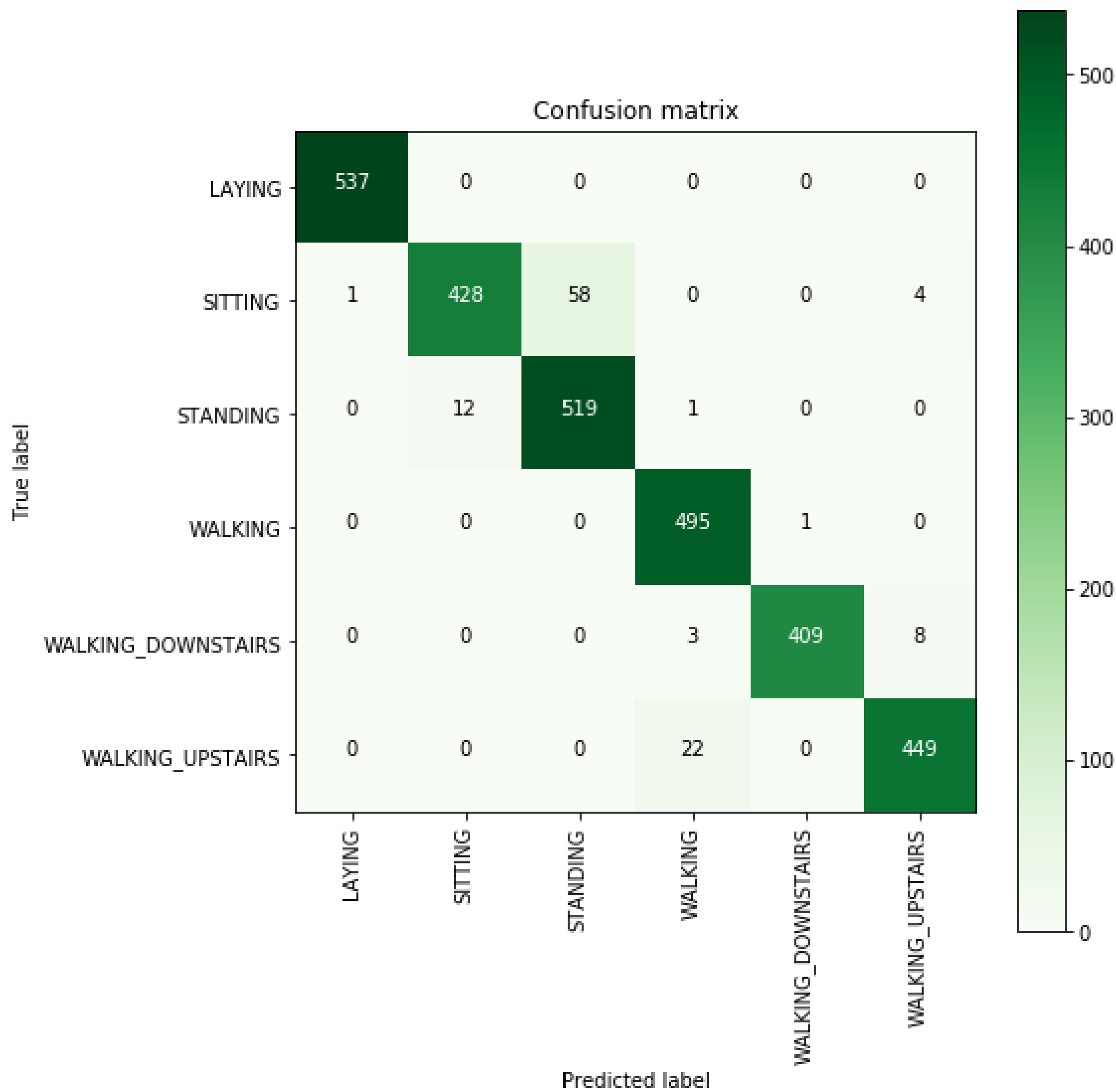
```
# start Grid search
parameters = {'C':[0.01, 0.1, 1, 10, 20, 30], 'penalty':['l2','l1']}
log_reg = linear_model.LogisticRegression()
log_reg_grid = GridSearchCV(log_reg, param_grid=parameters, cv=3, verbose=1, n_jo
log_reg_grid_results =  perform_model(log_reg_grid, X_train, y_train, X_test, y_t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

```
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 [0]:
```
plt.figure(figsize=(8,8))
plt.grid(b=False)
plot_confusion_matrix(log_reg_grid_results['confusion_matrix'], classes=labels, c
plt.show()
```



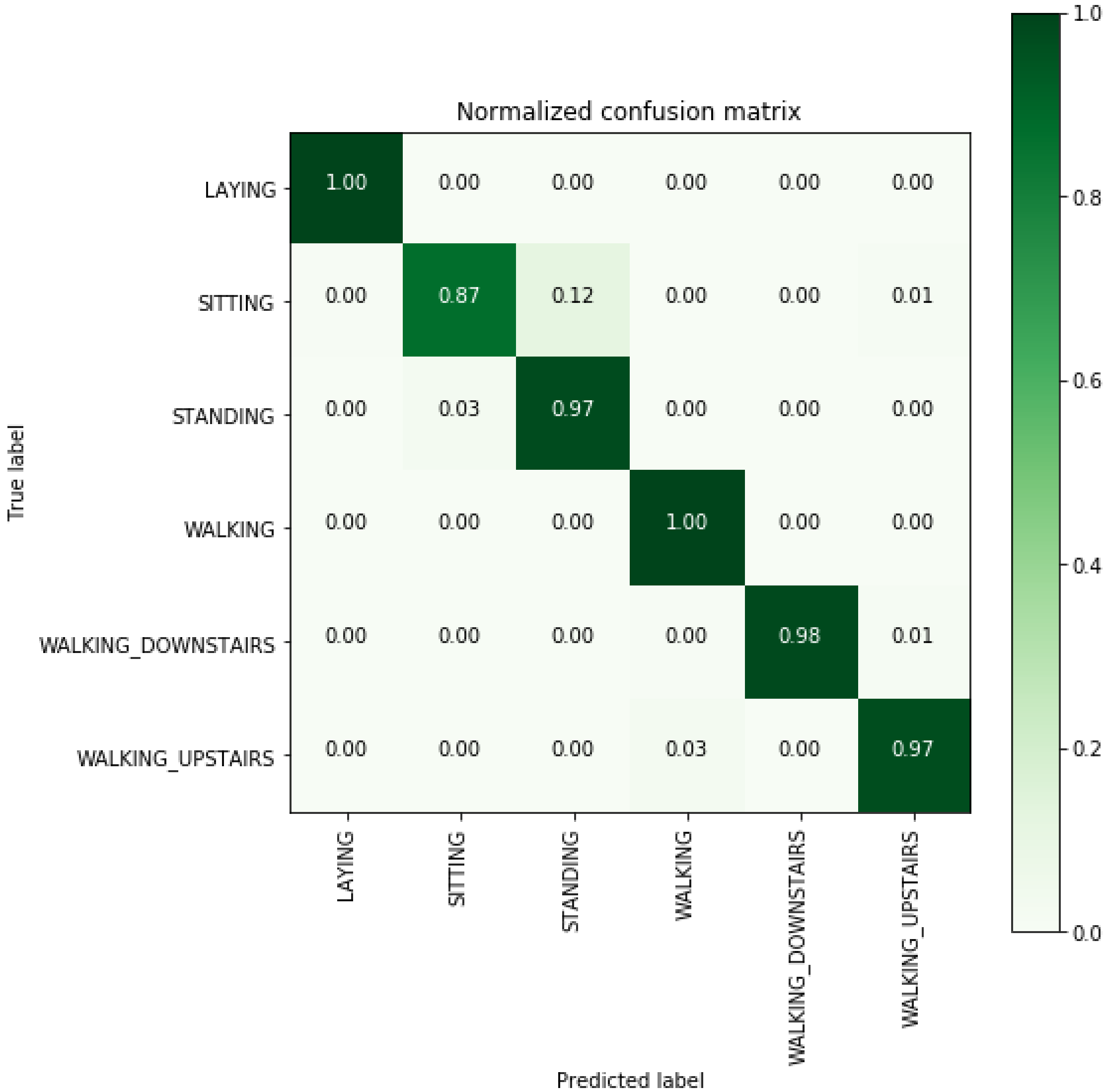
In [0]:
```
# observe the attributes of the model
print_grid_search_attributes(log_reg_grid_results['model'])
```

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

        LogisticRegression(C=30, class_weight=None, dual=False, fit_intercept=T
rue,
          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 [0]:
```
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
```

```
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]]
```




```
-------------------------
| 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
```

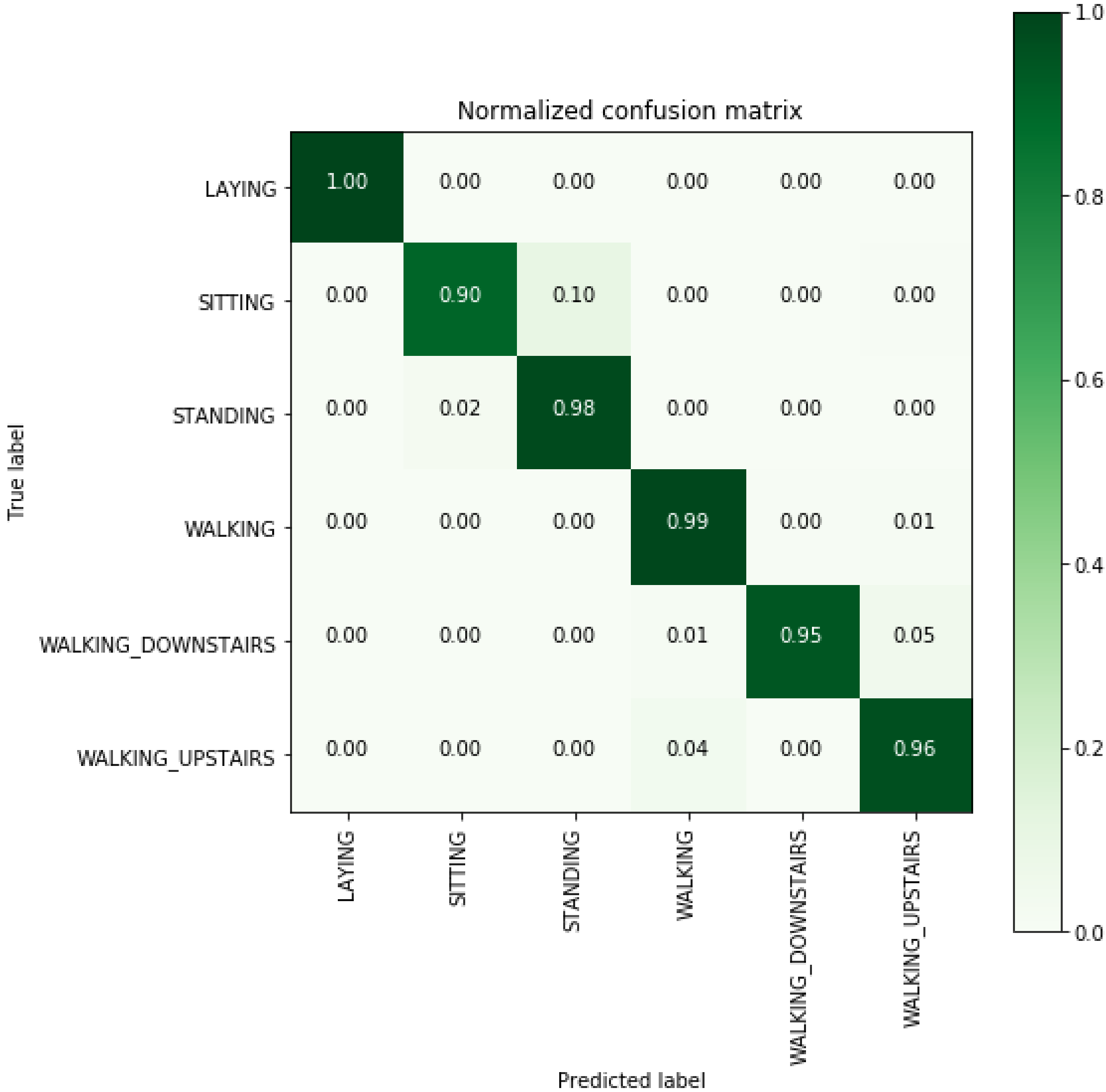
In [0]:
```
print_grid_search_attributes(lr_svc_grid_results['model'])
```

```
--------------------------
|      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 [0]:
```
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_te
```

```
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]]
```




```
-------------------------
| 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
```

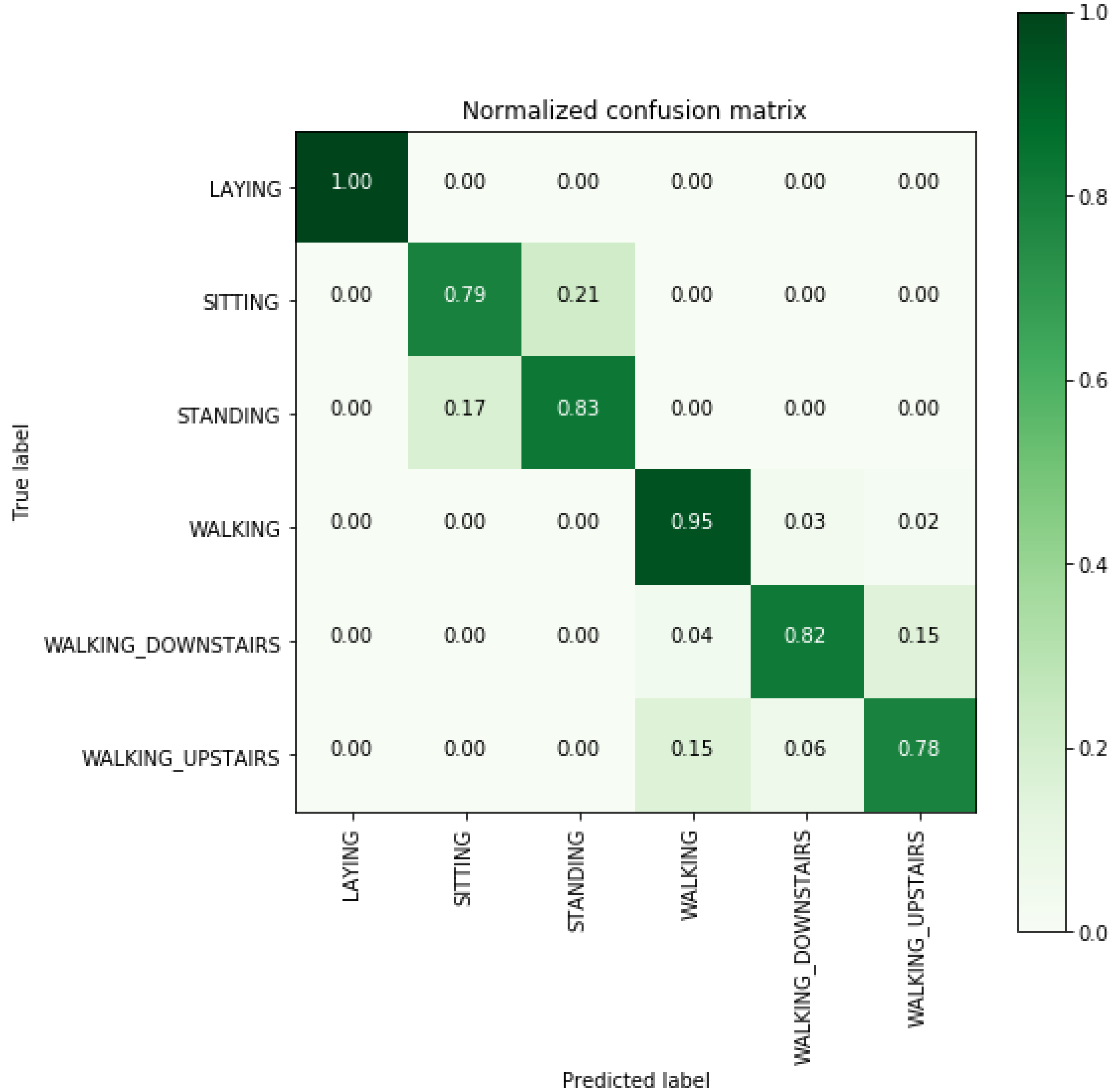
In [0]:
```
print_grid_search_attributes(rbf_svm_grid_results['model'])
```

```
--------------------------
|      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 [0]:
```
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_
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]]
```



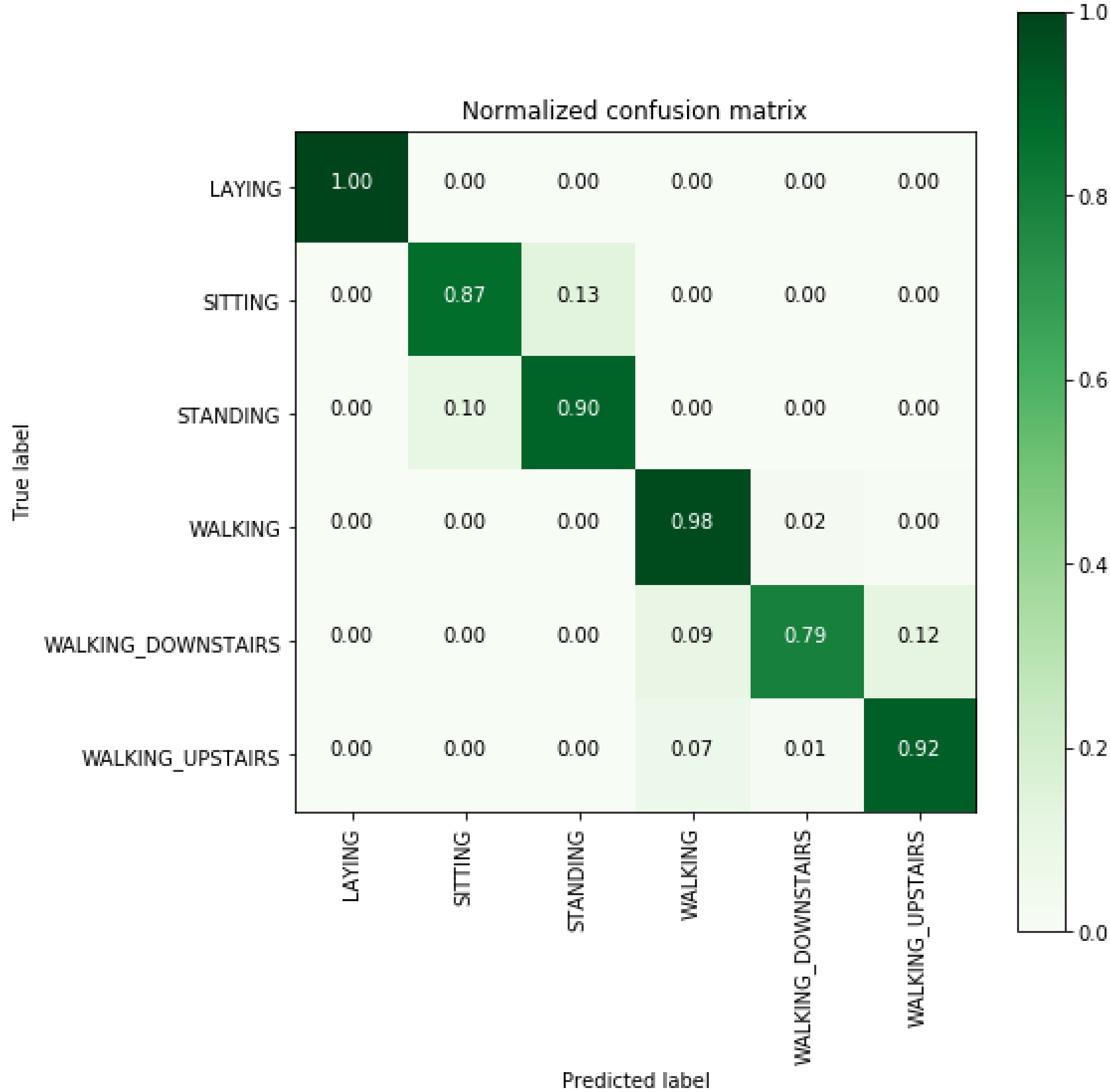

```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            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_depth=
7,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
```

```
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')

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

        {'max_depth': 7}

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.8369151251360174
```

5. Random Forest Classifier with GridSearch

In [0]:
```
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, clas
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]]
```



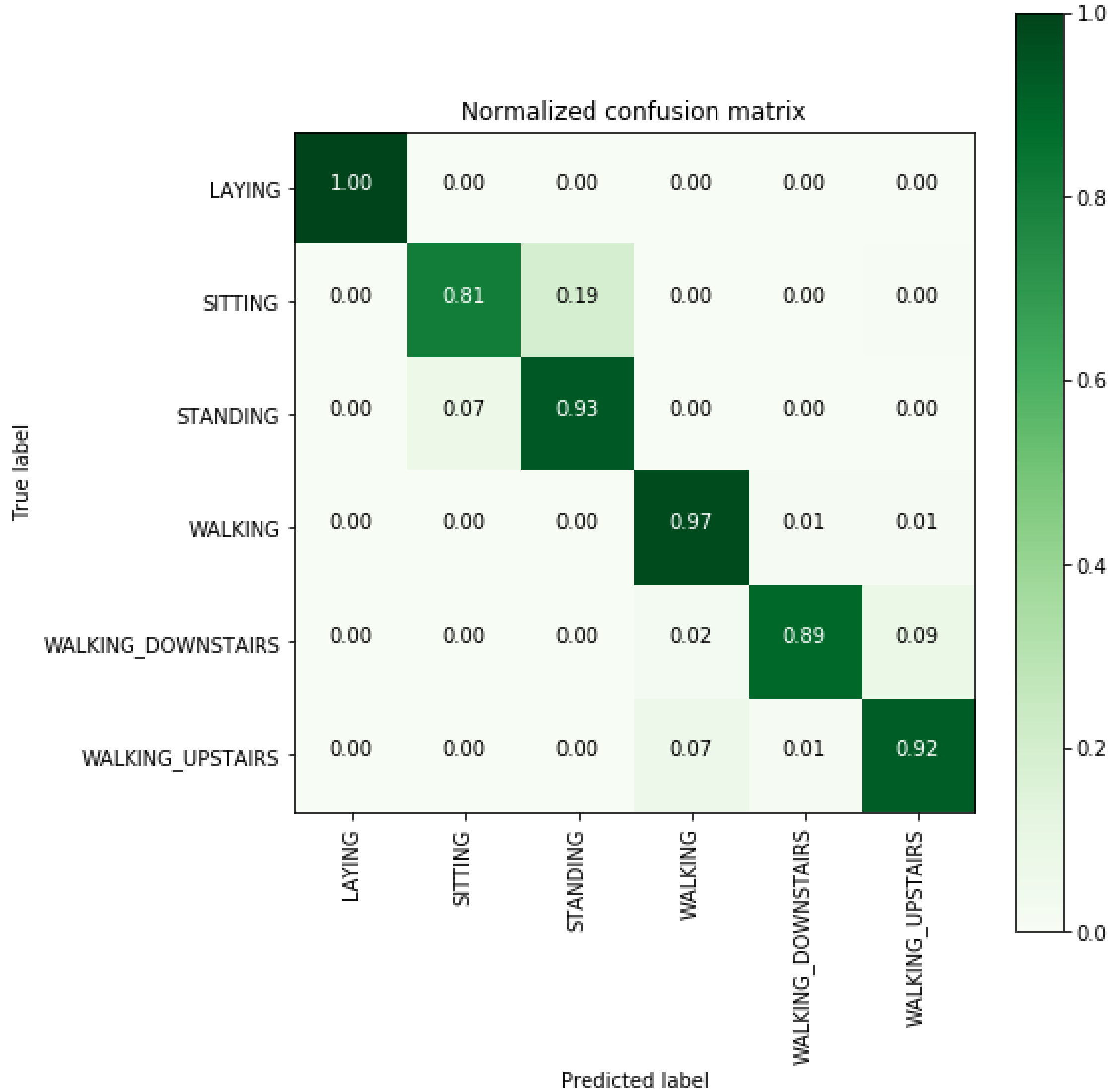

```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            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, criterion='gi
ni',
            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 [0]:
```
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, cl
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]]
```




```
--------------------------
| 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 [0]:
```
print('\n                    Accuracy     Error')
print('                   ----------   --------')
print('Logistic Regression : {:.04}%      {:.04}%'.format(log_reg_grid_results['
                                                 100-(log_reg_grid_results['accu

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

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

print('DecisionTree        : {:.04}%     {:.04}% '.format(dt_grid_results['accur
                                                       100-(dt_grid_results['acc

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

```
                     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***, but Linear SVC is better