Assignment 21 - Human Activity Detection

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
  - fBodyGyroJerkMa 8.

1. 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.

- 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 [6]:
```
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.googleuser
content.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_t
ype=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.t
est%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fw
ww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.go
ogleapis.com%2fauth%2fpeopleapi.readonly

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

In [7]:
```
!pip install hyperas
```

```
Requirement already satisfied: hyperas in /usr/local/lib/python3.6/dist
-packages (0.4.1)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.6/di
st-packages (from hyperas) (5.6.1)
Requirement already satisfied: jupyter in /usr/local/lib/python3.6/dist
-packages (from hyperas) (1.0.0)
Requirement already satisfied: entrypoints in /usr/local/lib/python3.6/
dist-packages (from hyperas) (0.3)
Requirement already satisfied: keras in /usr/local/lib/python3.6/dist-p
ackages (from hyperas) (2.2.5)
Requirement already satisfied: nbformat in /usr/local/lib/python3.6/dis
t-packages (from hyperas) (4.4.0)
Requirement already satisfied: hyperopt in /usr/local/lib/python3.6/dis
t-packages (from hyperas) (0.1.2)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.6/d
ist-packages (from nbconvert->hyperas) (0.6.0)
Requirement already satisfied: pygments in /usr/local/lib/python3.6/dis
t-packages (from nbconvert->hyperas) (2.1.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.6/dist-
packages (from nbconvert->hyperas) (3.1.0)
Requirement already satisfied: testpath in /usr/local/lib/python3.6/dis
t-packages (from nbconvert->hyperas) (0.4.4)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python
3.6/dist-packages (from nbconvert->hyperas) (4.3.3)
Requirement already satisfied: jupyter-core in /usr/local/lib/python3.
```

```
6/dist-packages (from nbconvert->hyperas) (4.6.1)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/p
ython3.6/dist-packages (from nbconvert->hyperas) (1.4.2)
Requirement already satisfied: jinja2>=2.4 in /usr/local/lib/python3.6/
dist-packages (from nbconvert->hyperas) (2.10.3)
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/pyth
on3.6/dist-packages (from nbconvert->hyperas) (0.8.4)
Requirement already satisfied: jupyter-console in /usr/local/lib/python
3.6/dist-packages (from jupyter->hyperas) (5.2.0)
Requirement already satisfied: qtconsole in /usr/local/lib/python3.6/di
st-packages (from jupyter->hyperas) (4.6.0)
Requirement already satisfied: ipywidgets in /usr/local/lib/python3.6/d
ist-packages (from jupyter->hyperas) (7.5.1)
Requirement already satisfied: ipykernel in /usr/local/lib/python3.6/di
st-packages (from jupyter->hyperas) (4.6.1)
Requirement already satisfied: notebook in /usr/local/lib/python3.6/dis
t-packages (from jupyter->hyperas) (5.2.2)
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-pa
ckages (from keras->hyperas) (2.8.0)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.6/d
ist-packages (from keras->hyperas) (1.12.0)
Requirement already satisfied: scipy>=0.14 in /usr/local/lib/python3.6/
dist-packages (from keras->hyperas) (1.3.3)
Requirement already satisfied: keras-applications>=1.0.8 in /usr/local/
lib/python3.6/dist-packages (from keras->hyperas) (1.0.8)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.6/dist-
packages (from keras->hyperas) (3.13)
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.
6/dist-packages (from keras->hyperas) (1.17.4)
Requirement already satisfied: keras-preprocessing>=1.1.0 in /usr/loca
l/lib/python3.6/dist-packages (from keras->hyperas) (1.1.0)
Requirement already satisfied: ipython-genutils in /usr/local/lib/pytho
n3.6/dist-packages (from nbformat->hyperas) (0.2.0)
Requirement already satisfied: jsonschema!=2.5.0,>=2.4 in /usr/local/li
b/python3.6/dist-packages (from nbformat->hyperas) (2.6.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.6/dis
t-packages (from hyperopt->hyperas) (2.4)
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-
packages (from hyperopt->hyperas) (0.16.0)
```

```
Requirement already satisfied: pymongo in /usr/local/lib/python3.6/dist
-packages (from hyperopt->hyperas) (3.10.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-pa
ckages (from hyperopt->hyperas) (4.28.1)
Requirement already satisfied: webencodings in /usr/local/lib/python3.
6/dist-packages (from bleach->nbconvert->hyperas) (0.5.1)
Requirement already satisfied: decorator in /usr/local/lib/python3.6/di
st-packages (from traitlets>=4.2->nbconvert->hyperas) (4.4.1)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/pytho
n3.6/dist-packages (from jinja2>=2.4->nbconvert->hyperas) (1.1.1)
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.0 in /usr/loc
al/lib/python3.6/dist-packages (from jupyter-console->jupyter->hyperas)
(1.0.18)
Requirement already satisfied: ipython in /usr/local/lib/python3.6/dist
-packages (from jupyter-console->jupyter->hyperas) (5.5.0)
Requirement already satisfied: jupyter-client in /usr/local/lib/python
3.6/dist-packages (from jupyter-console->jupyter->hyperas) (5.3.4)
Requirement already satisfied: widgetsnbextension~=3.5.0 in /usr/local/
lib/python3.6/dist-packages (from ipywidgets->jupyter->hyperas) (3.5.1)
Requirement already satisfied: tornado>=4.0 in /usr/local/lib/python3.
6/dist-packages (from ipykernel->jupyter->hyperas) (4.5.3)
Requirement already satisfied: terminado>=0.3.3; sys_platform != "win3
2" in /usr/local/lib/python3.6/dist-packages (from notebook->jupyter->h
yperas) (0.8.3)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.6/dist
-packages (from prompt-toolkit<2.0.0,>=1.0.0->jupyter-console->jupyter-
>hyperas) (0.1.7)
Requirement already satisfied: simplegeneric>0.8 in /usr/local/lib/pyth
on3.6/dist-packages (from ipython->jupyter-console->jupyter->hyperas)
(0.8.1)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.6/
dist-packages (from ipython->jupyter-console->jupyter->hyperas) (0.7.5)
Requirement already satisfied: pexpect; sys_platform != "win32" in /us
r/local/lib/python3.6/dist-packages (from ipython->jupyter-console->jup
yter->hyperas) (4.7.0)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/pytho
n3.6/dist-packages (from ipython->jupyter-console->jupyter->hyperas) (4
2.0.2)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/p
```

```
ython3.6/dist-packages (from jupyter-client->jupyter-console->jupyter->
hyperas) (2.6.1)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.6/di
st-packages (from jupyter-client->jupyter-console->jupyter->hyperas) (1
7.0.0)
Requirement already satisfied: ptyprocess; os_name != "nt" in /usr/loca
l/lib/python3.6/dist-packages (from terminado>=0.3.3; sys_platform !=
"win32"->notebook->jupyter->hyperas) (0.6.0)
```

In [0]:

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
import warnings
from datetime import datetime
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout
from hyperopt import Trials, STATUS_OK, tpe
from hyperas import optim
from hyperas.distributions import choice, uniform
warnings.simplefilter("ignore")
```

1.1 Extracting the features

In [9]:

```python
# get the features from the file features.txt
features = list()
```

```
with open('/content/drive/My Drive/HumanActivityRecognition.zip (Unzipped Files)/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
```

1.2 Obtain fron train data

In [10]:

```
# get the data from txt files to pandas dataffame
X_train = pd.read_csv('/content/drive/My Drive/HumanActivityRecognition.zip (Unzipped Files)/HAR/UCI_HAR_Dataset/train/X_train.txt', delim_whitespace=True, header=None)

X_train.columns = features
# add subject column to the dataframe
X_train['subject'] = pd.read_csv('/content/drive/My Drive/HumanActivityRecognition.zip (Unzipped Files)/HAR/UCI_HAR_Dataset/train/subject_train.txt', header=None, squeeze=True)

y_train = pd.read_csv('/content/drive/My Drive/HumanActivityRecognition.zip (Unzipped Files)/HAR/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()
```

Out[10]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	t

6367	0.292701	-0.01742	-0.109541	-0.984132	-0.978384	-0.973413	-0.986724	-

1 rows × 564 columns

In [11]:
```
train.head()
```

Out[11]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBo m
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.98
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.97
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.96
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.98
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.97

5 rows × 564 columns

In [12]:
```
train.shape
```

Out[12]: (7352, 564)

Obtain the test data

In [13]:
```
# get the data from txt files to pandas dataffame
X_test = pd.read_csv('/content/drive/My Drive/HumanActivityRecognition.
zip (Unzipped Files)/HAR/UCI_HAR_Dataset/test/X_test.txt', delim_whites
pace=True, header=None)

X_test.columns = features
```

```python
# add subject column to the dataframe
X_test['subject'] = pd.read_csv('/content/drive/My Drive/HumanActivityRecognition.zip (Unzipped Files)/HAR/UCI_HAR_Dataset/test/subject_test.txt', header=None, squeeze=True)

# get y labels from the txt file
y_test = pd.read_csv('/content/drive/My Drive/HumanActivityRecognition.zip (Unzipped Files)/HAR/UCI_HAR_Dataset/test/y_test.txt', names=['Activity'], squeeze=True)
y_test_labels = y_test.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS',\
                       4:'SITTING', 5:'STANDING',6:'LAYING'})


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

Out[13]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	t
1882	0.2762	-0.020466	-0.115694	-0.997035	-0.941785	-0.973809	-0.997528	-

1 rows × 564 columns

In [14]:

```python
test.head()
```

Out[14]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBo m
0	0.257178	-0.023285	-0.014654	-0.938404	-0.920091	-0.667683	-0.952501	-0.92

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBo m
1	0.286027	-0.013163	-0.119083	-0.975415	-0.967458	-0.944958	-0.986799	-0.96
2	0.275485	-0.026050	-0.118152	-0.993819	-0.969926	-0.962748	-0.994403	-0.97
3	0.270298	-0.032614	-0.117520	-0.994743	-0.973268	-0.967091	-0.995274	-0.97
4	0.274833	-0.027848	-0.129527	-0.993852	-0.967445	-0.978295	-0.994111	-0.96

5 rows × 564 columns

In [15]:
```
test.shape
```

Out[15]:
```
(2947, 564)
```

Data Cleaning

1. Check for Duplicates

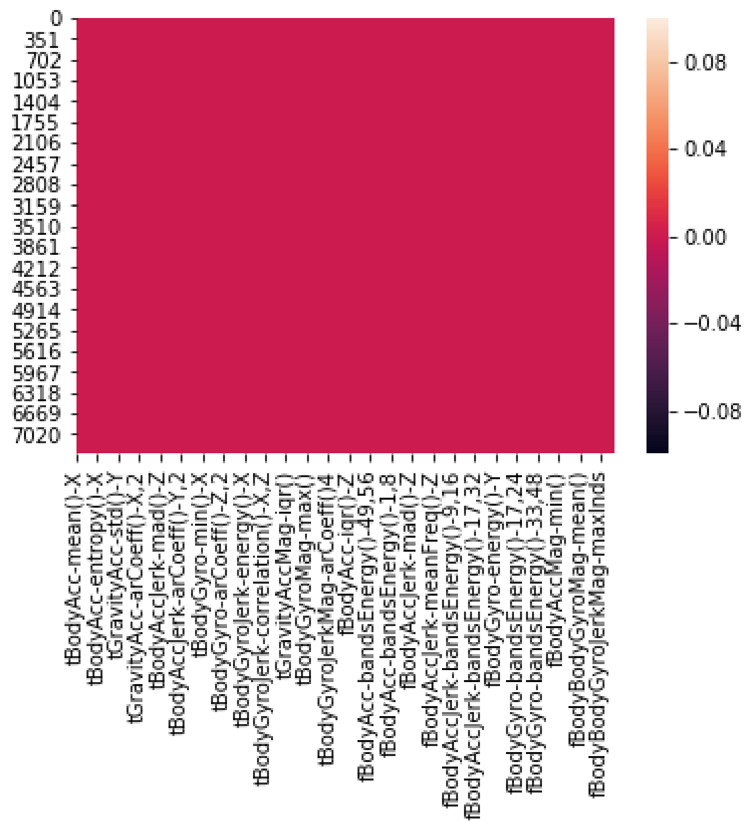
In [16]:
```
print('No of duplicates in train: {}'.format(sum(train.duplicated())))
print('No of duplicates in test : {}'.format(sum(test.duplicated())))
```

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

In [17]:
```
print("\nGraphical representation of null values for train data\n")

sns.heatmap(train.isnull())
```

```
Graphical representation of null values for train data
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6681119588>



2. Checking for NaN/null values

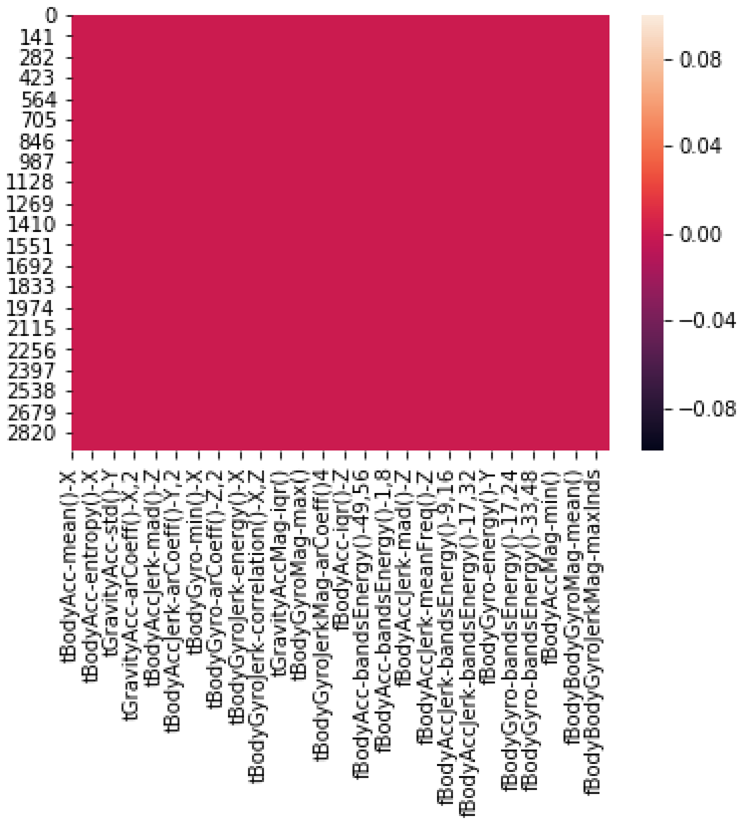
In [18]:
```
print('We have {} NaN/Null values in train'.format(train.isnull().value
s.sum()))
print('We have {} NaN/Null values in test'.format(test.isnull().values.
sum()))
```

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

In [19]:
```python
print("\nGraphical representation of null values for test data\n")

sns.heatmap(test.isnull())
```

```
Graphical representation of null values for test data
```

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7f667df70438>

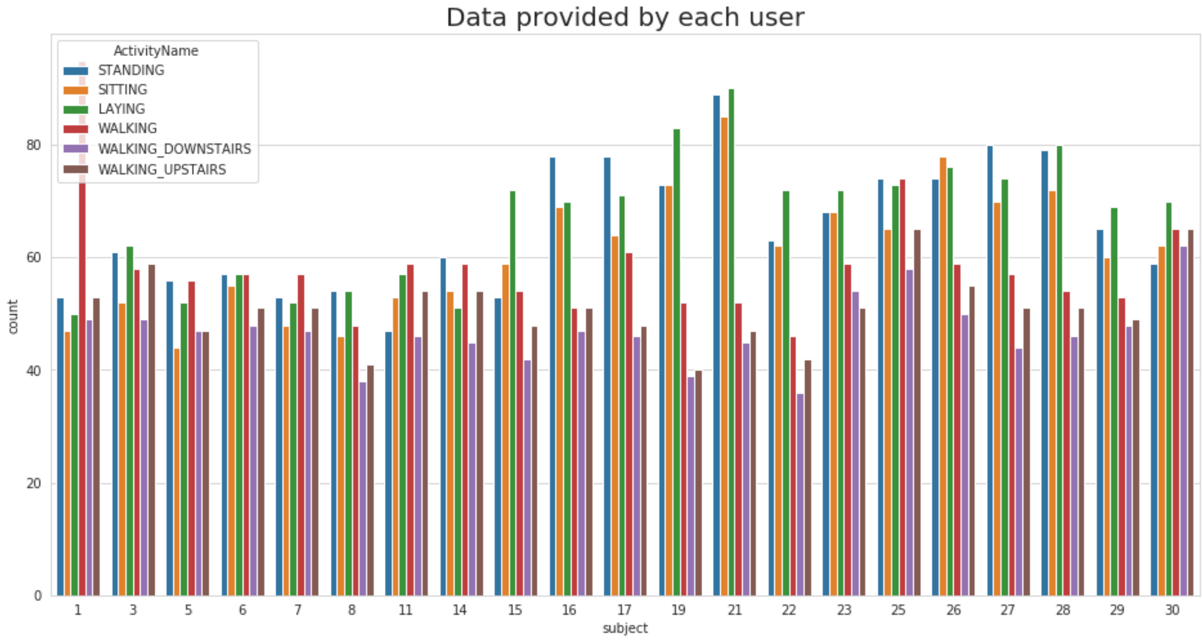


3. Check for data imbalance

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

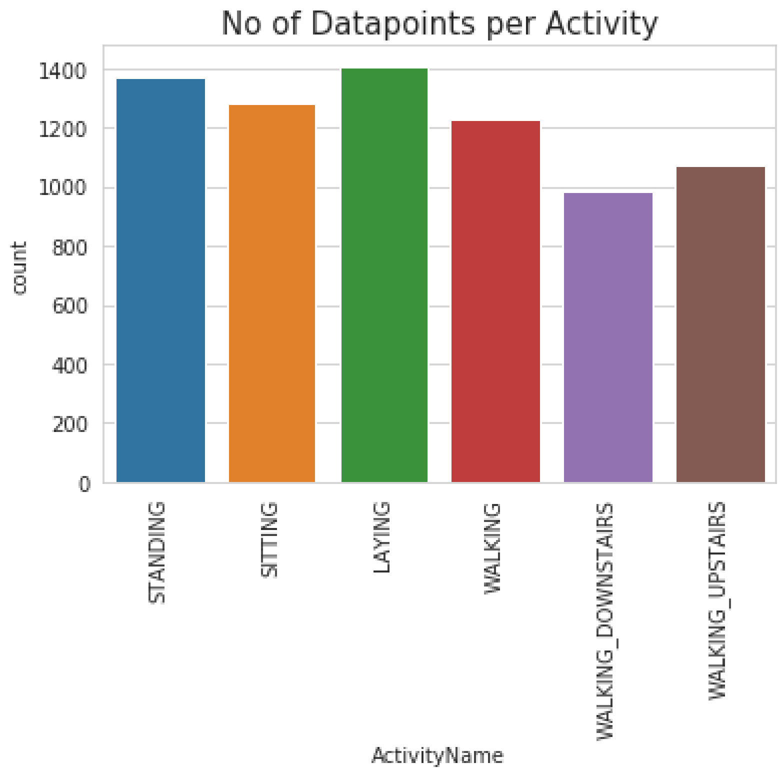
```
sns.set_style('whitegrid')
plt.rcParams['font.family'] = 'Dejavu Sans'
```

In [21]:
```
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 [22]:
```
plt.title('No of Datapoints per Activity', fontsize=15)
sns.countplot(train.ActivityName)
plt.xticks(rotation=90)
plt.show()
```

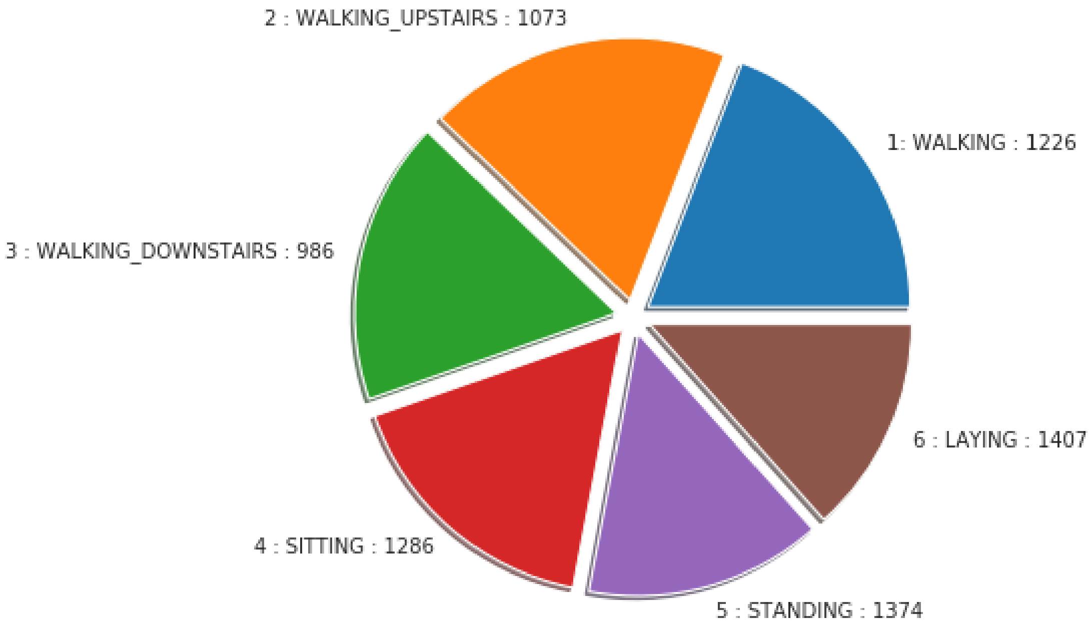


observation:

From above barplot we can say that our data is almost equally balanced

In [23]:
```
labels = ['1: WALKING : 1226', '2 : WALKING_UPSTAIRS : 1073', '3 : WALKING_DOWNSTAIRS : 986',
          '4 : SITTING : 1286', '5 : STANDING : 1374', '6 : LAYING : 1407']

explode = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1]

plt.pie(train.Activity.value_counts(), explode = explode, labels = labels, radius = 1.5, shadow = True)
```

Out[23]:
```
([<matplotlib.patches.Wedge at 0x7f667d747470>,
  <matplotlib.patches.Wedge at 0x7f667d747ba8>,
  <matplotlib.patches.Wedge at 0x7f667d7562e8>,
  <matplotlib.patches.Wedge at 0x7f667d7569e8>,
  <matplotlib.patches.Wedge at 0x7f667d762128>,
  <matplotlib.patches.Wedge at 0x7f667d762828>],
 [Text(1.4431238278887235, 0.9898957608656574, '1: WALKING : 1226'),
  Text(-0.3798237460574488, 1.7082839113949668, '2 : WALKING_UPSTAIRS :
1073'),
  Text(-1.7095723197702357, 0.37398192934340496, '3 : WALKING_DOWNSTAIR
S : 986'),
  Text(-1.1443614362337686, -1.323985235288138, '4 : SITTING : 1286'),
  Text(0.46616160314671445, -1.6867700968868538, '5 : STANDING : 137
4'),
  Text(1.5969558711437852, -0.7157038113768814, '6 : LAYING : 1407')])
```

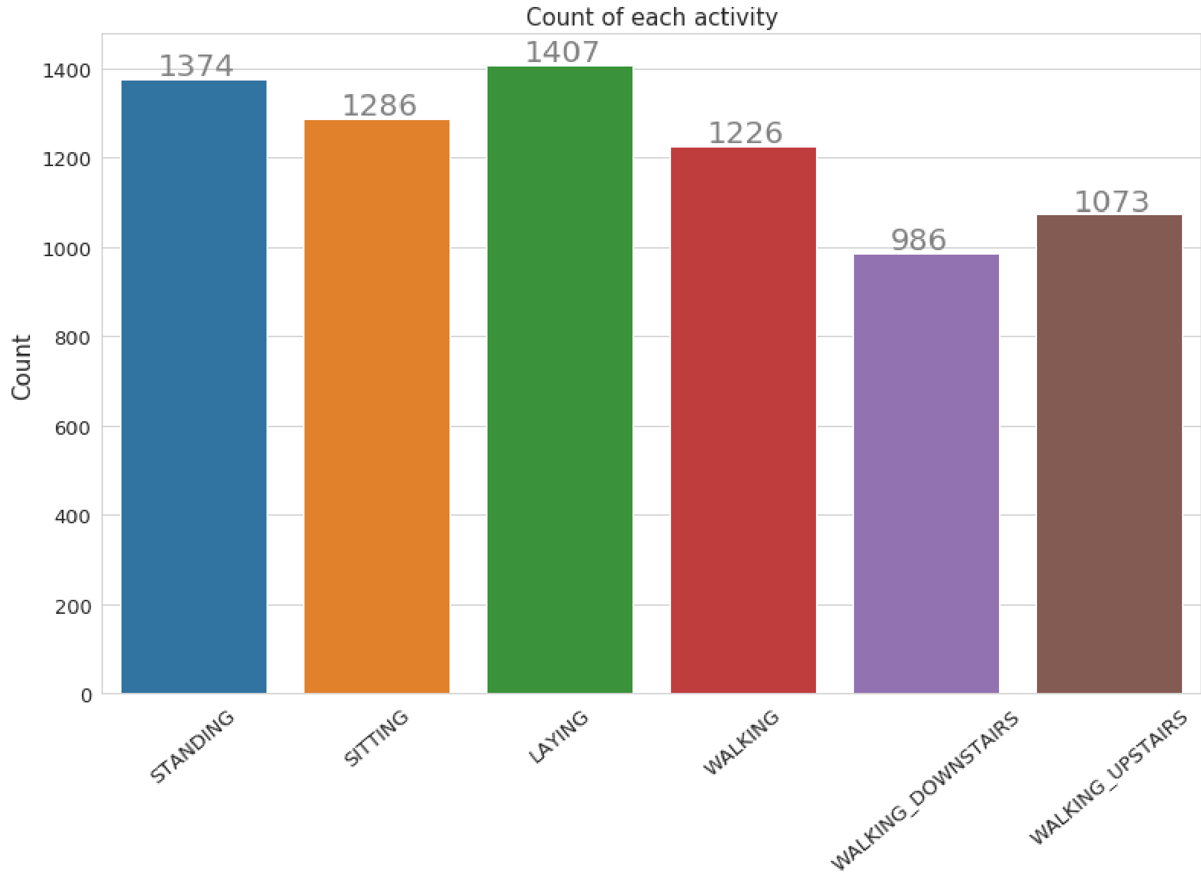


Observation -

1) For Laying activity, maximum count is 1407.

2) For walking downstairs, minimum count is 986.

In [24]:
```
fig = plt.figure(figsize = (10, 6))
ax = fig.add_axes([0,0,1,1])
ax.set_title("Count of each activity", fontsize = 15)
plt.tick_params(labelsize = 10)
sns.countplot(x = "ActivityName", data = train)
for i in ax.patches:
    ax.text(x = i.get_x() + 0.2, y = i.get_height()+10, s = str(i.get_h
eight()), fontsize = 20, color = "grey")
plt.xlabel("")
plt.ylabel("Count", fontsize = 15)
plt.tick_params(labelsize = 13)
plt.xticks(rotation = 40)
plt.show()
```



4. Changing feature names

In [25]:
```
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[25]:

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

5. Save this dataframe in a csv files

In [0]:

```
train.to_csv('/content/drive/My Drive/HumanActivityRecognition.zip (Unz
ipped Files)/HAR/UCI_HAR_Dataset/csv_files/train.csv', index=False)
test.to_csv('/content/drive/My Drive/HumanActivityRecognition.zip (Unzi
pped Files)/HAR/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.***"
AAIC

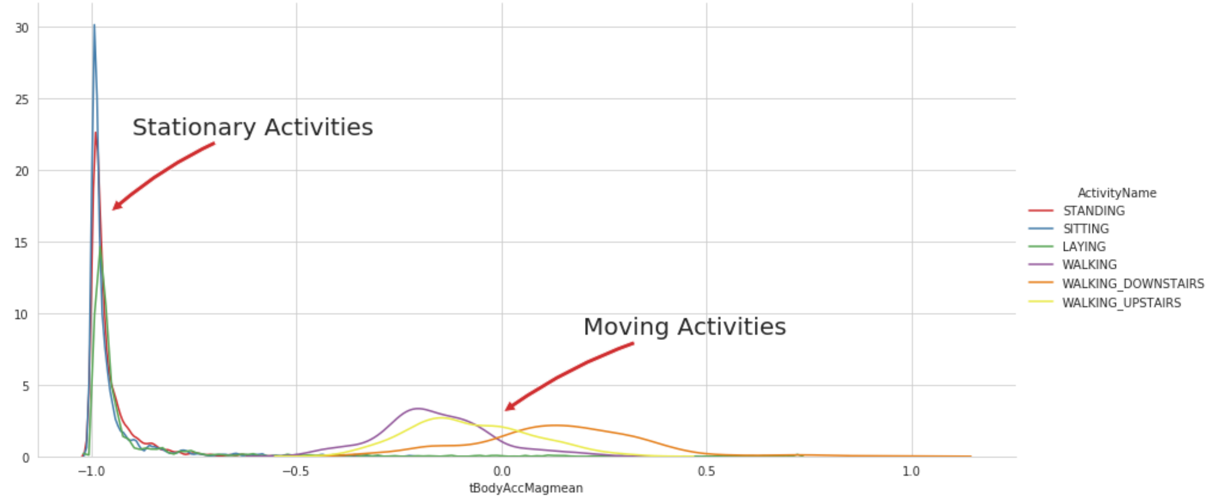
1. Featuring Engineering from Domain Knowledge

- **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.

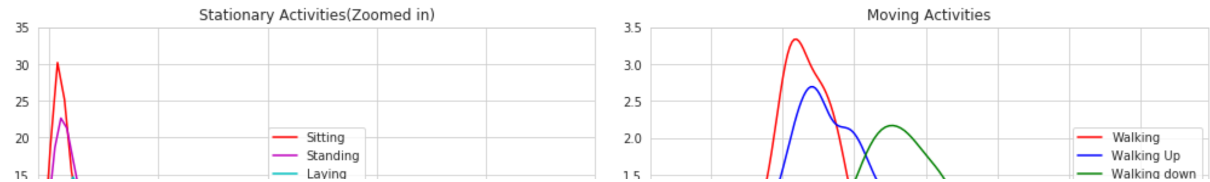
2. Stationary and Moving activities are completely different

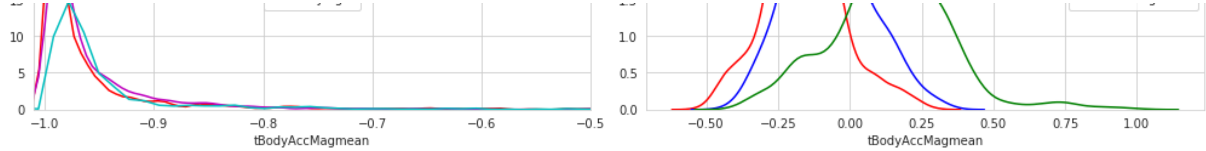
In [27]:
```
sns.set_palette("Set1", desat=0.80)
facetgrid = sns.FacetGrid(train, hue='ActivityName', size=6,aspect=2)
facetgrid.map(sns.distplot,'tBodyAccMagmean', hist=False)\
    .add_legend()
plt.annotate("Stationary Activities", xy=(-0.956,17), xytext=(-0.9, 23
), size=20,\
            va='center', ha='left',\
            arrowprops=dict(arrowstyle="simple",connectionstyle="arc3,r
ad=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,r
ad=0.1"))
plt.show()
```



In [28]:

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

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

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


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

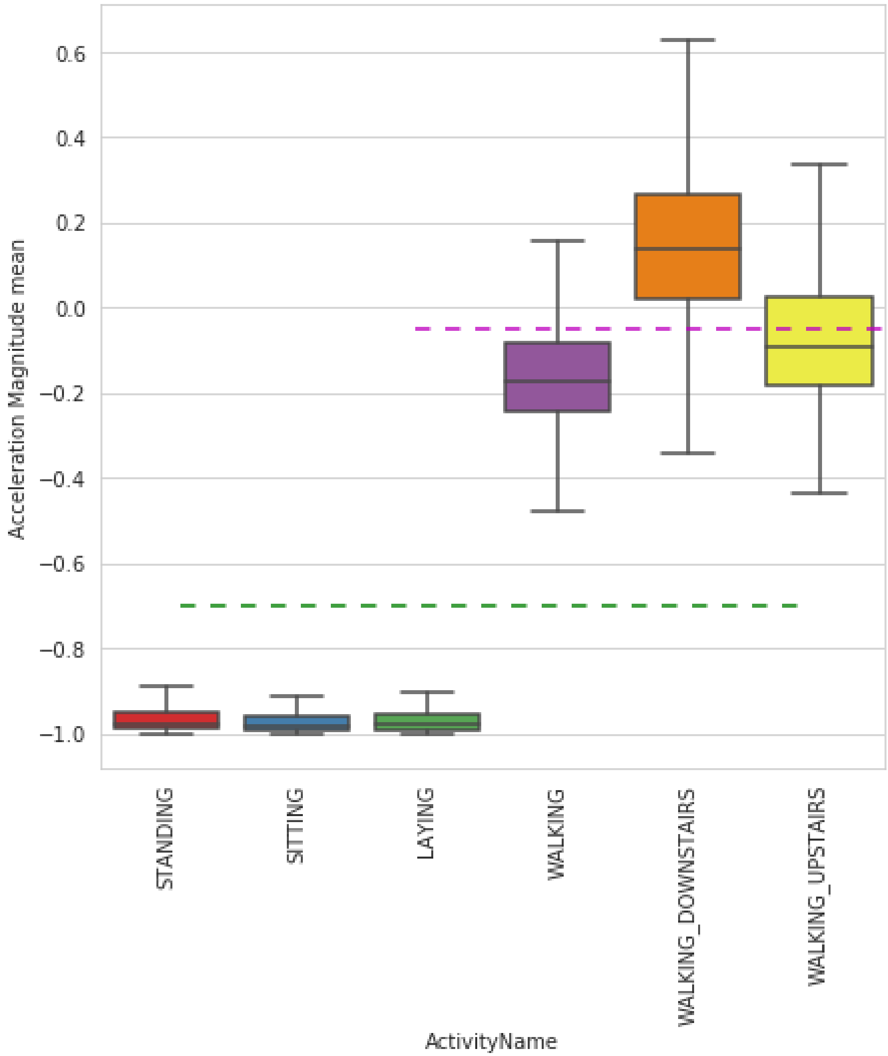




3. Magnitude of an acceleration can saperate it well

In [29]:

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



Observations:

1. If tAccMean is < -0.8 then the Activities are either Standing or Sitting or Laying.

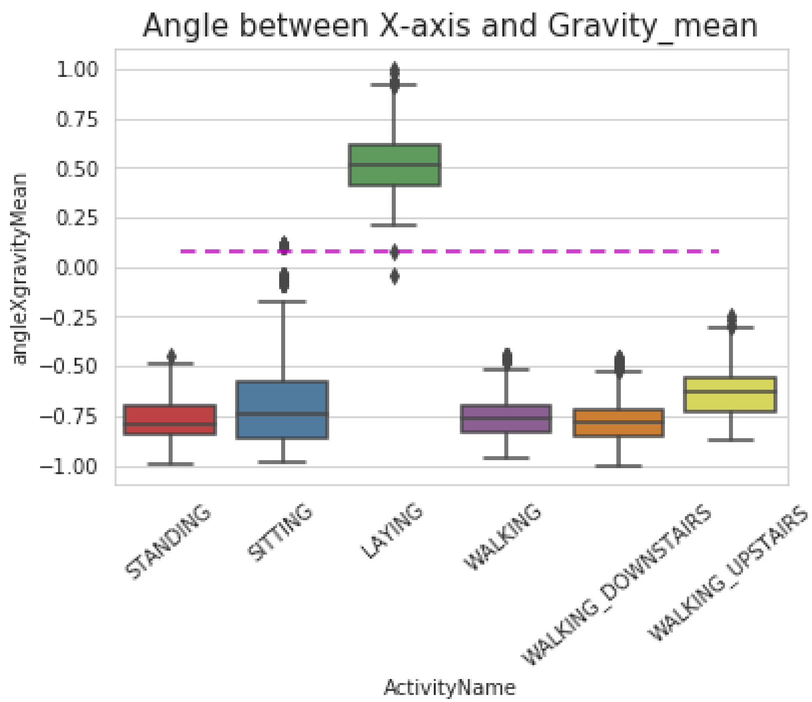
2. If tAccMean is > -0.6 then the Activities are either Walking or WalkingDownstairs or WalkingUpstairs.

3. If tAccMean > 0.0 then the Activity is WalkingDownstairs.

4. We can classify 75% the Acitivity labels with some errors.

4. Position of GravityAccelerationComponants also matters

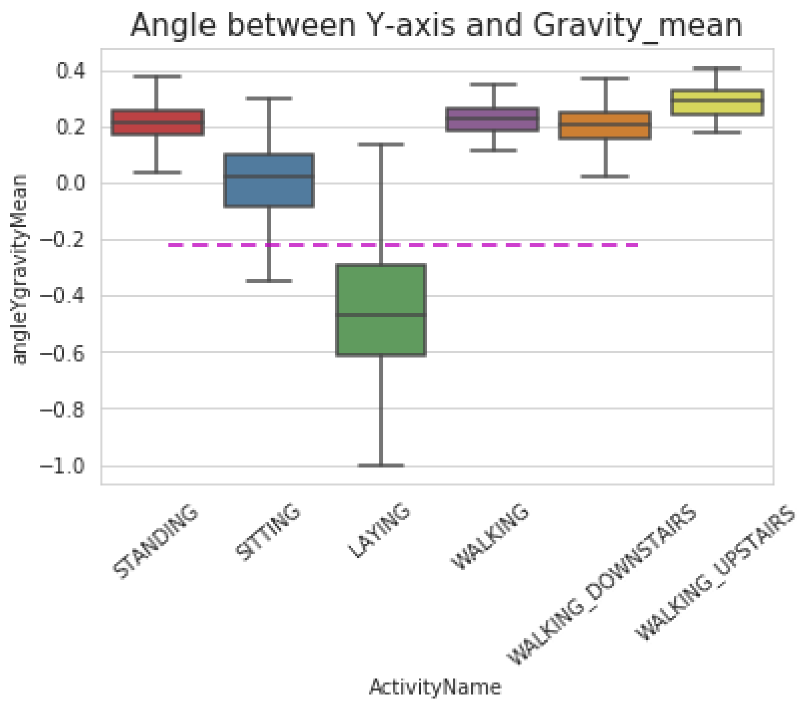
In [30]:
```
sns.boxplot(x='ActivityName', y='angleXgravityMean', data=train)
plt.axhline(y=0.08, xmin=0.1, xmax=0.9,c='m',dashes=(5,3))
plt.title('Angle between X-axis and Gravity_mean', fontsize=15)
plt.xticks(rotation = 40)
plt.show()
```



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 [31]:
```
sns.boxplot(x='ActivityName', y='angleYgravityMean', data = train, show
fliers=False)
plt.title('Angle between Y-axis and Gravity_mean', fontsize=15)
plt.xticks(rotation = 40)
plt.axhline(y=-0.22, xmin=0.1, xmax=0.8, dashes=(5,3), c='m')
plt.show()
```



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
```

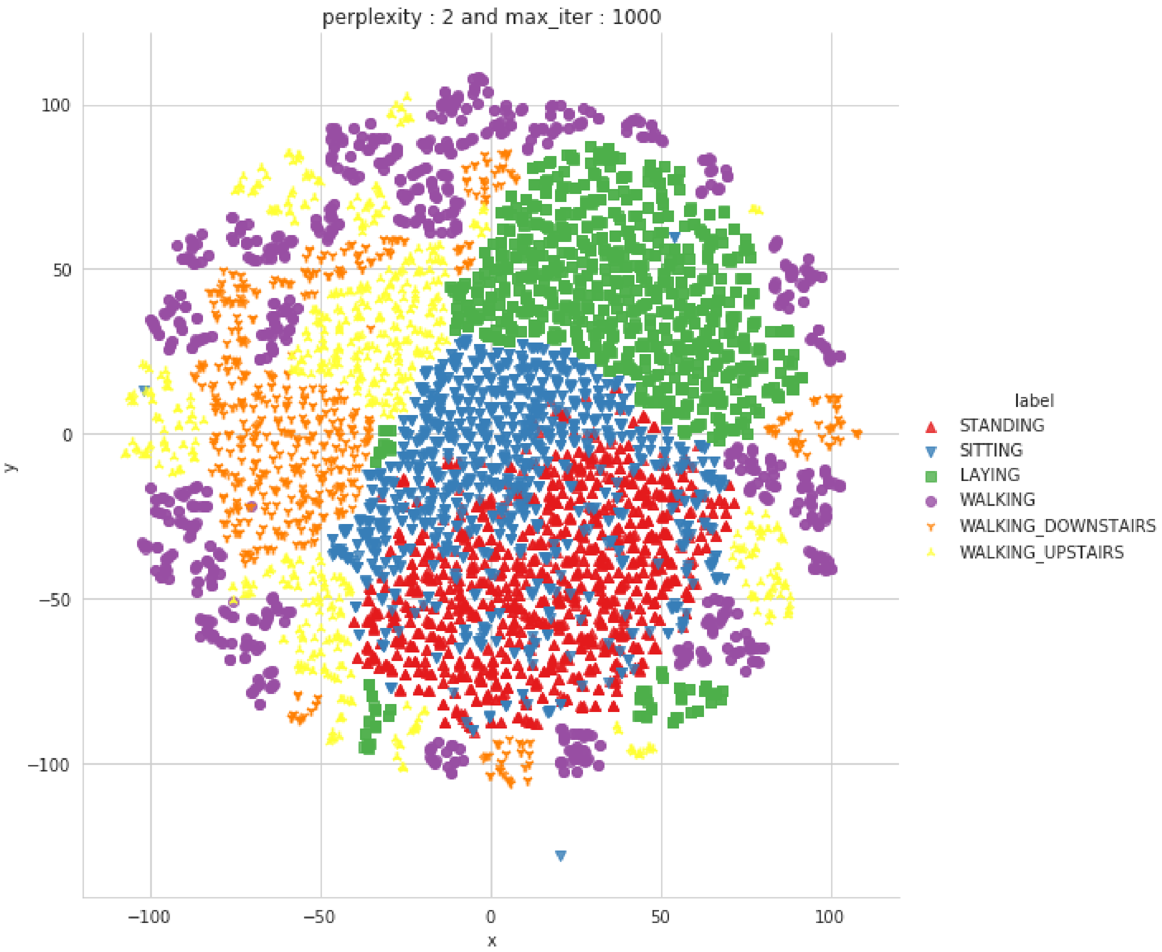
In [0]:

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

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

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

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

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

In [34]:
```python
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 0.139s...
[t-SNE] Computed neighbors for 7352 samples in 33.528s...
[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 0.044s
[t-SNE] Iteration 50: error = 124.8061676, gradient norm = 0.0285766
(50 iterations in 7.834s)
[t-SNE] Iteration 100: error = 106.8774338, gradient norm = 0.0290012
(50 iterations in 3.307s)
[t-SNE] Iteration 150: error = 100.5496140, gradient norm = 0.0217231
(50 iterations in 2.312s)
[t-SNE] Iteration 200: error = 97.1940231, gradient norm = 0.0174928
(50 iterations in 2.201s)
[t-SNE] Iteration 250: error = 94.9549026, gradient norm = 0.0140103
(50 iterations in 2.186s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 9
4.954903
[t-SNE] Iteration 300: error = 4.1132541, gradient norm = 0.0015612
(50 iterations in 1.889s)
[t-SNE] Iteration 350: error = 3.2053852, gradient norm = 0.0010098
(50 iterations in 1.731s)
[t-SNE] Iteration 400: error = 2.7754846, gradient norm = 0.0007136
(50 iterations in 1.797s)
[t-SNE] Iteration 450: error = 2.5115921, gradient norm = 0.0005689
(50 iterations in 1.838s)
[t-SNE] Iteration 500: error = 2.3280444, gradient norm = 0.0004827
```

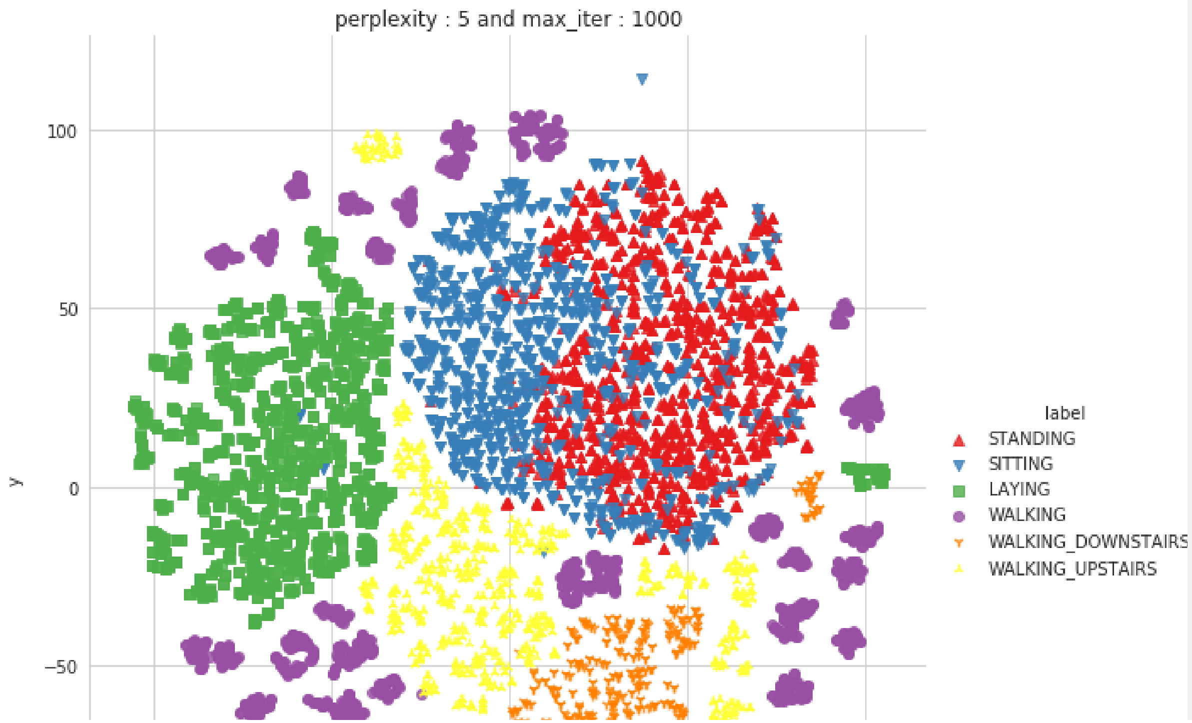
```
(50 iterations in 1.891s)
[t-SNE] Iteration 550: error = 2.1903365, gradient norm = 0.0004195
(50 iterations in 1.817s)
[t-SNE] Iteration 600: error = 2.0808122, gradient norm = 0.0003692
(50 iterations in 1.853s)
[t-SNE] Iteration 650: error = 1.9911609, gradient norm = 0.0003327
(50 iterations in 1.849s)
[t-SNE] Iteration 700: error = 1.9157352, gradient norm = 0.0003002
(50 iterations in 1.864s)
[t-SNE] Iteration 750: error = 1.8507640, gradient norm = 0.0002788
(50 iterations in 1.912s)
[t-SNE] Iteration 800: error = 1.7941960, gradient norm = 0.0002577
(50 iterations in 1.907s)
[t-SNE] Iteration 850: error = 1.7441696, gradient norm = 0.0002393
(50 iterations in 1.884s)
[t-SNE] Iteration 900: error = 1.6995744, gradient norm = 0.0002251
(50 iterations in 1.877s)
[t-SNE] Iteration 950: error = 1.6595628, gradient norm = 0.0002089
(50 iterations in 1.923s)
[t-SNE] Iteration 1000: error = 1.6233702, gradient norm = 0.0001986
(50 iterations in 2.003s)
[t-SNE] KL divergence after 1000 iterations: 1.623370
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```

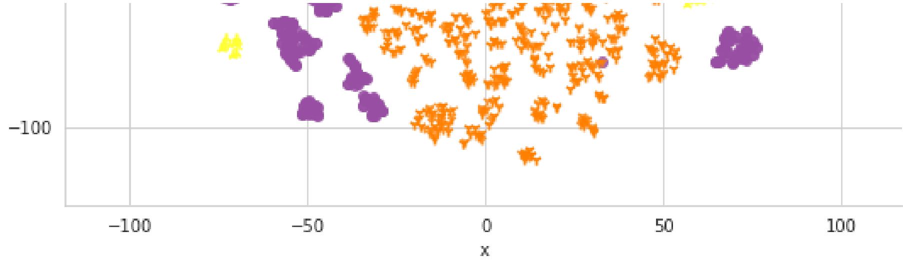



```
Done

performing tsne with perplexity 5 and with 1000 iterations at max
[t-SNE] Computing 16 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.163s...
[t-SNE] Computed neighbors for 7352 samples in 33.669s...
```

```
[t-SNE] Computed neighbors for 7352 samples in 33.009s...
[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 0.057s
[t-SNE] Iteration 50: error = 113.9422455, gradient norm = 0.0243714

(50 iterations in 4.992s)
[t-SNE] Iteration 100: error = 97.2134857, gradient norm = 0.0152058
(50 iterations in 2.243s)
[t-SNE] Iteration 150: error = 93.0750504, gradient norm = 0.0087848
(50 iterations in 1.859s)
[t-SNE] Iteration 200: error = 91.1818390, gradient norm = 0.0070986
(50 iterations in 1.843s)
[t-SNE] Iteration 250: error = 90.0344925, gradient norm = 0.0049017
(50 iterations in 1.773s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 9
0.034492
[t-SNE] Iteration 300: error = 3.5737171, gradient norm = 0.0014639
(50 iterations in 1.708s)
[t-SNE] Iteration 350: error = 2.8145530, gradient norm = 0.0007546
(50 iterations in 1.774s)
[t-SNE] Iteration 400: error = 2.4325902, gradient norm = 0.0005284
(50 iterations in 1.788s)
[t-SNE] Iteration 450: error = 2.2144225, gradient norm = 0.0004087
(50 iterations in 1.738s)
[t-SNE] Iteration 500: error = 2.0696349, gradient norm = 0.0003325
(50 iterations in 1.773s)
[t-SNE] Iteration 550: error = 1.9641596, gradient norm = 0.0002815
(50 iterations in 1.786s)
[t-SNE] Iteration 600: error = 1.8832289, gradient norm = 0.0002480
(50 iterations in 1.798s)
[t-SNE] Iteration 650: error = 1.8182777, gradient norm = 0.0002186
(50 iterations in 1.810s)
[t-SNE] Iteration 700: error = 1.7645472, gradient norm = 0.0001971
```

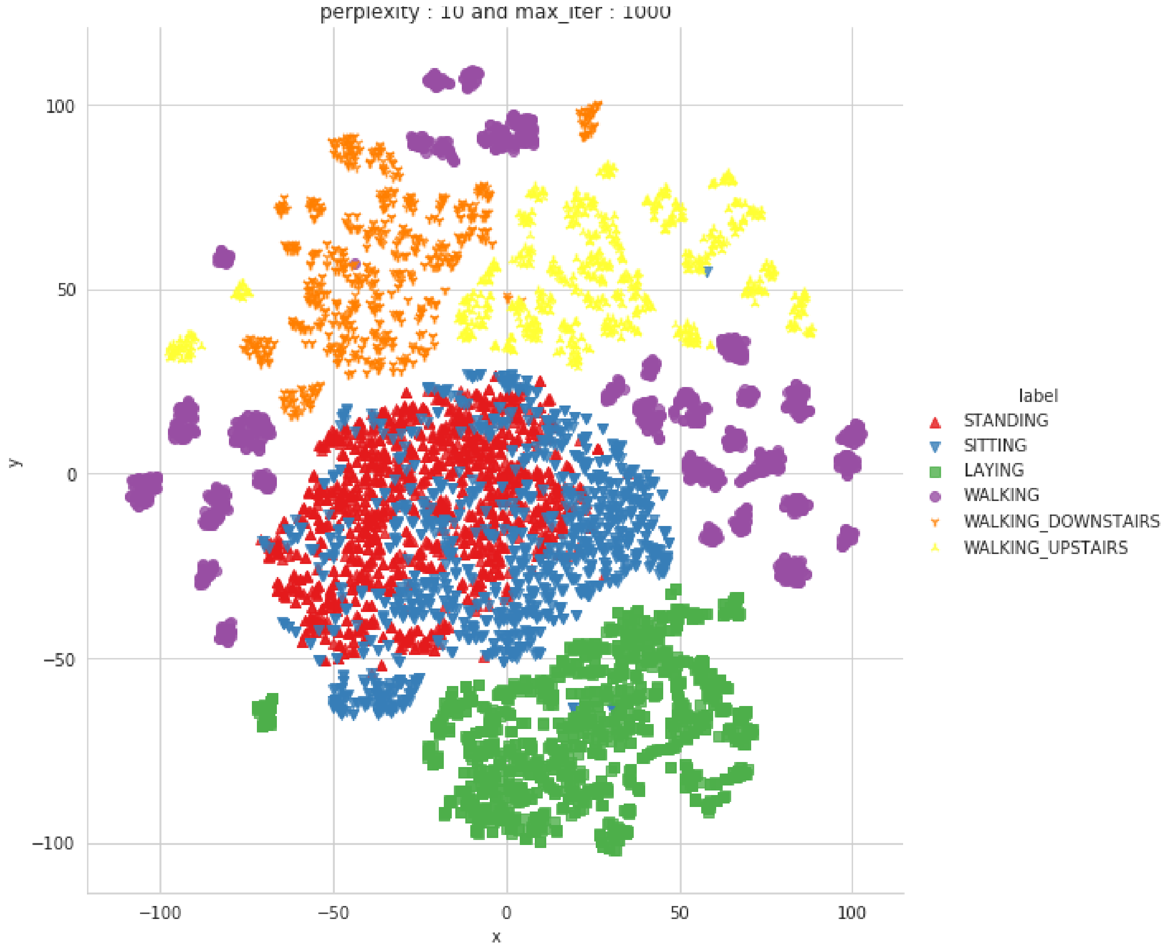
```
(50 iterations in 1.835s)
[t-SNE] Iteration 750: error = 1.7192354, gradient norm = 0.0001800
(50 iterations in 1.845s)
[t-SNE] Iteration 800: error = 1.6803484, gradient norm = 0.0001654
(50 iterations in 1.814s)
[t-SNE] Iteration 850: error = 1.6463732, gradient norm = 0.0001538
(50 iterations in 1.785s)
[t-SNE] Iteration 900: error = 1.6165980, gradient norm = 0.0001425
(50 iterations in 1.782s)
[t-SNE] Iteration 950: error = 1.5903155, gradient norm = 0.0001327
(50 iterations in 1.805s)

[t-SNE] Iteration 1000: error = 1.5667050, gradient norm = 0.0001263
(50 iterations in 1.804s)
[t-SNE] KL divergence after 1000 iterations: 1.566705
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 0.147s...
[t-SNE] Computed neighbors for 7352 samples in 33.984s...
[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 0.107s
[t-SNE] Iteration 50: error = 105.5476303, gradient norm = 0.0211429
(50 iterations in 3.439s)
[t-SNE] Iteration 100: error = 91.2134323, gradient norm = 0.0143048
(50 iterations in 2.369s)
[t-SNE] Iteration 150: error = 87.5614624, gradient norm = 0.0062345
(50 iterations in 2.058s)
[t-SNE] Iteration 200: error = 86.2821045, gradient norm = 0.0042266
(50 iterations in 1.962s)
[t-SNE] Iteration 250: error = 85.5850677, gradient norm = 0.0040011
(50 iterations in 1.916s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 8
5.585068
[t-SNE] Iteration 300: error = 3.1450641, gradient norm = 0.0013913
(50 iterations in 1.946s)
[t-SNE] Iteration 350: error = 2.5038221, gradient norm = 0.0006494
```

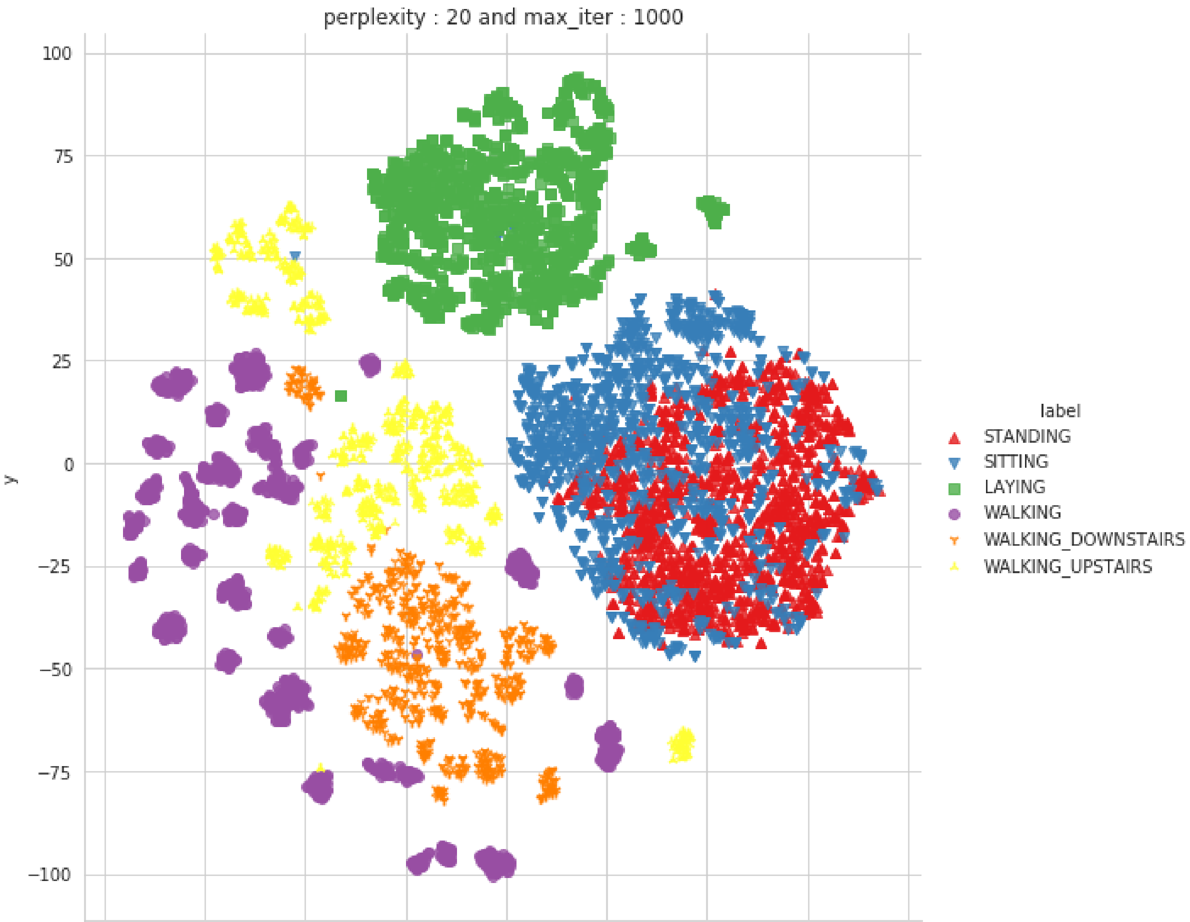
```
(50 iterations in 1.800s)
[t-SNE] Iteration 400: error = 2.1848371, gradient norm = 0.0004264
(50 iterations in 1.859s)
[t-SNE] Iteration 450: error = 2.0004153, gradient norm = 0.0003167
(50 iterations in 1.841s)
[t-SNE] Iteration 500: error = 1.8819729, gradient norm = 0.0002508
(50 iterations in 1.892s)
[t-SNE] Iteration 550: error = 1.7983079, gradient norm = 0.0002120
(50 iterations in 1.803s)
[t-SNE] Iteration 600: error = 1.7355720, gradient norm = 0.0001843
(50 iterations in 1.803s)
[t-SNE] Iteration 650: error = 1.6861305, gradient norm = 0.0001620
(50 iterations in 1.778s)
[t-SNE] Iteration 700: error = 1.6464047, gradient norm = 0.0001443
(50 iterations in 1.785s)
[t-SNE] Iteration 750: error = 1.6135875, gradient norm = 0.0001306
(50 iterations in 1.812s)
[t-SNE] Iteration 800: error = 1.5859232, gradient norm = 0.0001223
(50 iterations in 1.841s)
[t-SNE] Iteration 850: error = 1.5626334, gradient norm = 0.0001125
(50 iterations in 1.806s)
[t-SNE] Iteration 900: error = 1.5424889, gradient norm = 0.0001052
(50 iterations in 1.802s)
[t-SNE] Iteration 950: error = 1.5251316, gradient norm = 0.0000979
(50 iterations in 1.848s)
[t-SNE] Iteration 1000: error = 1.5098749, gradient norm = 0.0000951
(50 iterations in 1.860s)
[t-SNE] KL divergence after 1000 iterations: 1.509875
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 0.137s...
[t-SNE] Computed neighbors for 7352 samples in 34.769s...
[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
```

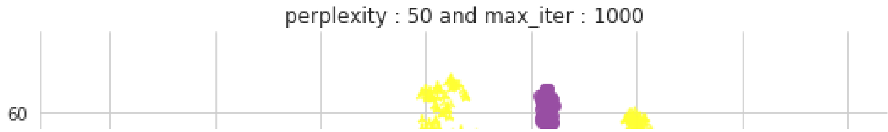
```
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352

[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.274335
[t-SNE] Computed conditional probabilities in 0.207s
[t-SNE] Iteration 50: error = 95.4594650, gradient norm = 0.0352948
(50 iterations in 4.525s)
[t-SNE] Iteration 100: error = 83.6445236, gradient norm = 0.0072099
(50 iterations in 2.427s)
[t-SNE] Iteration 150: error = 81.7852173, gradient norm = 0.0032198
(50 iterations in 2.098s)
[t-SNE] Iteration 200: error = 81.1039734, gradient norm = 0.0024356
(50 iterations in 2.052s)
[t-SNE] Iteration 250: error = 80.7446747, gradient norm = 0.0017798
(50 iterations in 2.005s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 8
0.744675
[t-SNE] Iteration 300: error = 2.6980817, gradient norm = 0.0013012
(50 iterations in 1.973s)
[t-SNE] Iteration 350: error = 2.1642921, gradient norm = 0.0005756
(50 iterations in 1.906s)
[t-SNE] Iteration 400: error = 1.9149469, gradient norm = 0.0003462
(50 iterations in 1.949s)
[t-SNE] Iteration 450: error = 1.7685212, gradient norm = 0.0002476
(50 iterations in 1.947s)
[t-SNE] Iteration 500: error = 1.6743351, gradient norm = 0.0001927
(50 iterations in 2.031s)
[t-SNE] Iteration 550: error = 1.6099870, gradient norm = 0.0001572
(50 iterations in 2.057s)
[t-SNE] Iteration 600: error = 1.5632104, gradient norm = 0.0001330
(50 iterations in 1.987s)
[t-SNE] Iteration 650: error = 1.5276917, gradient norm = 0.0001169
(50 iterations in 2.031s)
[t-SNE] Iteration 700: error = 1.5003418, gradient norm = 0.0001045
(50 iterations in 2.063s)
[t-SNE] Iteration 750: error = 1.4784725, gradient norm = 0.0000954
(50 iterations in 2.053s)
[t-SNE] Iteration 800: error = 1.4606897, gradient norm = 0.0000890
(50 iterations in 1.981s)
[t-SNE] Iteration 850: error = 1.4464505, gradient norm = 0.0000810
```

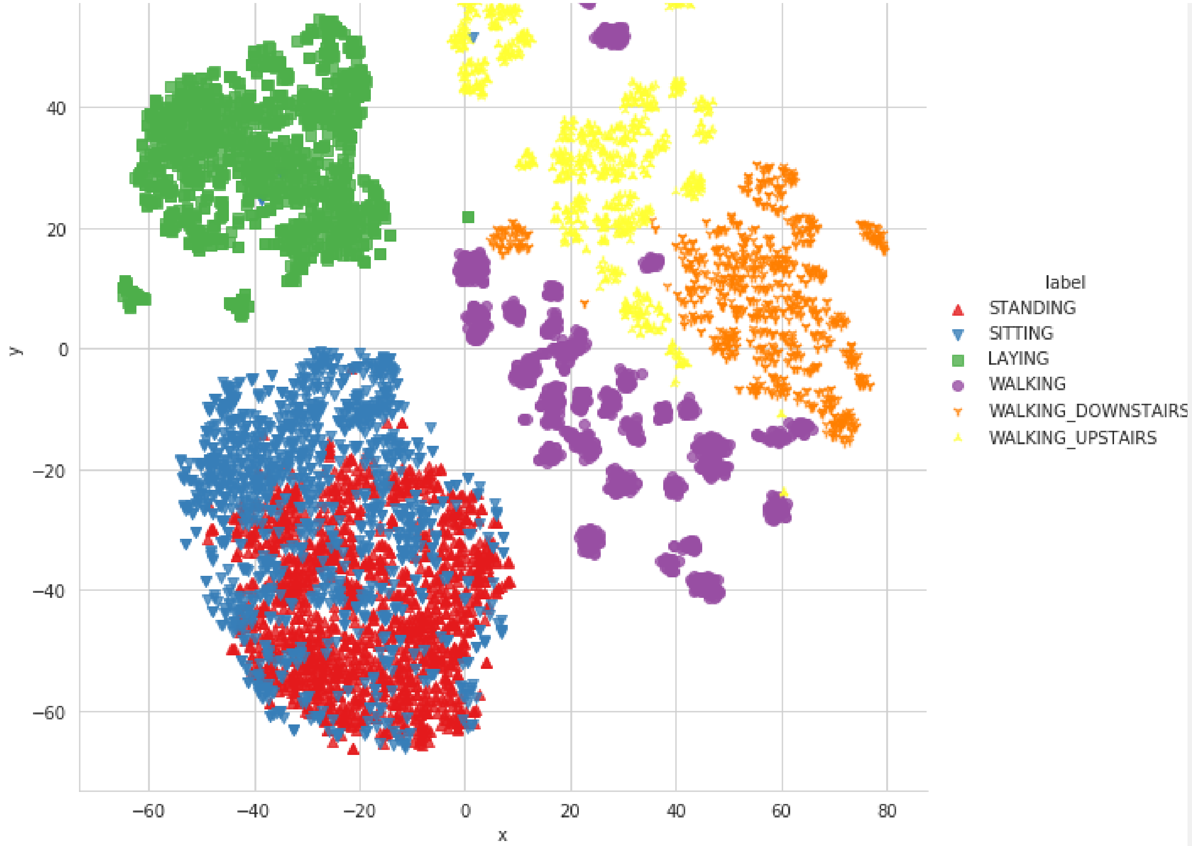
```
(50 iterations in 1.967s)

[t-SNE] Iteration 900: error = 1.4340171, gradient norm = 0.0000776
(50 iterations in 1.983s)
[t-SNE] Iteration 950: error = 1.4234513, gradient norm = 0.0000718
(50 iterations in 1.969s)
[t-SNE] Iteration 1000: error = 1.4145061, gradient norm = 0.0000711
(50 iterations in 1.990s)
[t-SNE] KL divergence after 1000 iterations: 1.414506
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 0.139s...
[t-SNE] Computed neighbors for 7352 samples in 35.823s...
[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 0.494s
[t-SNE] Iteration 50: error = 85.3524475, gradient norm = 0.0300863
(50 iterations in 3.709s)
[t-SNE] Iteration 100: error = 75.6486969, gradient norm = 0.0047866
(50 iterations in 3.235s)
[t-SNE] Iteration 150: error = 74.5928955, gradient norm = 0.0022354
(50 iterations in 2.898s)
[t-SNE] Iteration 200: error = 74.2268372, gradient norm = 0.0017724
(50 iterations in 2.926s)
[t-SNE] Iteration 250: error = 74.0473022, gradient norm = 0.0011132
(50 iterations in 2.996s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 7
4.047302
[t-SNE] Iteration 300: error = 2.1568205, gradient norm = 0.0011777
(50 iterations in 2.751s)
[t-SNE] Iteration 350: error = 1.7580163, gradient norm = 0.0004870
(50 iterations in 2.448s)
[t-SNE] Iteration 400: error = 1.5896004, gradient norm = 0.0002816
(50 iterations in 2.424s)
[t-SNE] Iteration 450: error = 1.4954200, gradient norm = 0.0001909
(50 iterations in 2.470s)
[t-SNE] Iteration 500: error = 1.4351623, gradient norm = 0.0001405
```

```
(50 iterations in 2.475s)
[t-SNE] Iteration 550: error = 1.3938395, gradient norm = 0.0001137
(50 iterations in 2.431s)
[t-SNE] Iteration 600: error = 1.3646932, gradient norm = 0.0000954
(50 iterations in 2.465s)
[t-SNE] Iteration 650: error = 1.3433210, gradient norm = 0.0000826
(50 iterations in 2.538s)
[t-SNE] Iteration 700: error = 1.3278475, gradient norm = 0.0000753
(50 iterations in 2.493s)
[t-SNE] Iteration 750: error = 1.3164045, gradient norm = 0.0000695
(50 iterations in 2.516s)
[t-SNE] Iteration 800: error = 1.3075224, gradient norm = 0.0000636
(50 iterations in 2.483s)
[t-SNE] Iteration 850: error = 1.3003546, gradient norm = 0.0000599
(50 iterations in 2.482s)
[t-SNE] Iteration 900: error = 1.2945056, gradient norm = 0.0000596
(50 iterations in 2.492s)
[t-SNE] Iteration 950: error = 1.2897059, gradient norm = 0.0000542
(50 iterations in 2.506s)
[t-SNE] Iteration 1000: error = 1.2854290, gradient norm = 0.0000558
(50 iterations in 2.498s)
[t-SNE] KL divergence after 1000 iterations: 1.285429
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 [36]:
```python
train = pd.read_csv('/content/drive/My Drive/HumanActivityRecognition.zip (Unzipped Files)/HAR/UCI_HAR_Dataset/csv_files/train.csv')
test = pd.read_csv('/content/drive/My Drive/HumanActivityRecognition.zi
```

```
p (Unzipped Files)/HAR/UCI_HAR_Dataset/csv_files/test.csv')
print(train.shape, test.shape)
```

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

In [37]:

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

Out[37]:

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

3 rows × 564 columns

In [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 [40]:

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

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

Let's model with our data

Labels that are useful in plotting confusion matrix

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

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_normalize=True, \
                 print_cm=True, cm_cmap=plt.cm.Greens):


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

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


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


    # calculate overall accuracty of the model
```

```
    accuracy = metrics.accuracy_score(y_true=y_test, y_pred=y_pred)
    # store accuracy in results
    results['accuracy'] = accuracy
    print('---------------------')
    print('|      Accuracy      |')
    print('---------------------')
    print('\n    {}\n\n'.format(accuracy))


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

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

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

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

    return results
```

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 GridSearch
    print('--------------------------')
    print('|      Best Estimator     |')
    print('--------------------------')
    print('\n\t{}\n'.format(model.best_estimator_))


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


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


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

Logistic Regression with Grid Search

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

from sklearn.model_selection import GridSearchCV
```

In [46]:
```
# start Grid search
parameters = {'C':[0.01, 0.1, 1, 10, 20, 30], 'penalty':['l2','l1']}
log_reg = linear_model.LogisticRegression()
log_reg_grid = GridSearchCV(log_reg, param_grid=parameters, cv=3, verbo
se=1, n_jobs=-1)
log_reg_grid_results =  perform_model(log_reg_grid, X_train, y_train, X
_test, y_test, class_labels=labels)
```

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

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  36 out of  36 | elapsed:  1.9min finished
```

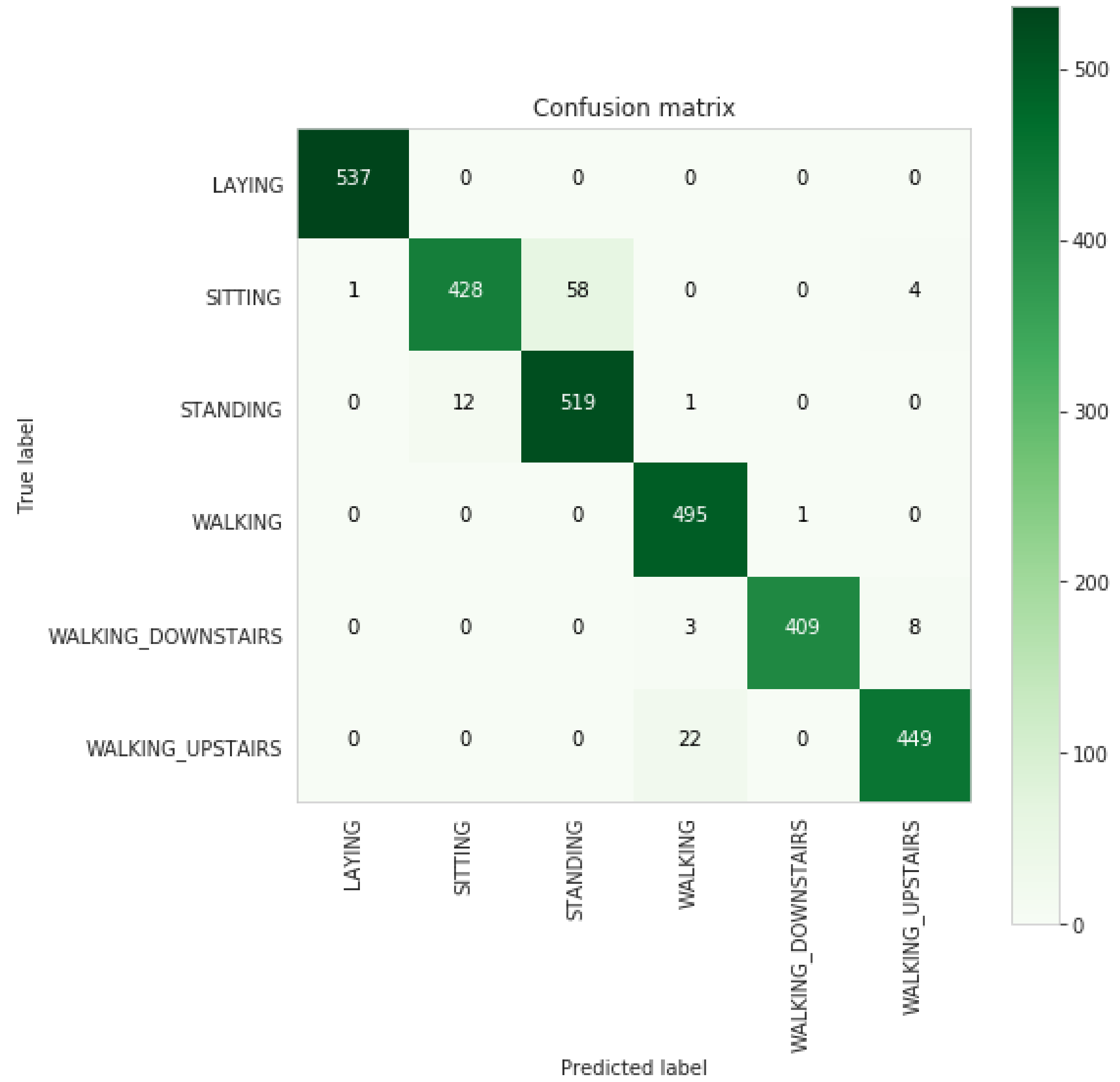
```
Done


training_time(HH:MM:SS.ms) - 0:02:02.197169


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.008467
```

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

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

In [47]:
```
plt.figure(figsize=(8,8))
plt.grid(b=False)
plot_confusion_matrix(log_reg_grid_results['confusion_matrix'], classes
=labels, cmap=plt.cm.Greens, )
plt.show()
```



In [48]:

```
# 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_int
ercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbos
e=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]:
```
from sklearn.svm import LinearSVC
```

In [50]:
```
parameters = {'C':[0.125, 0.5, 1, 2, 8, 16]}
```

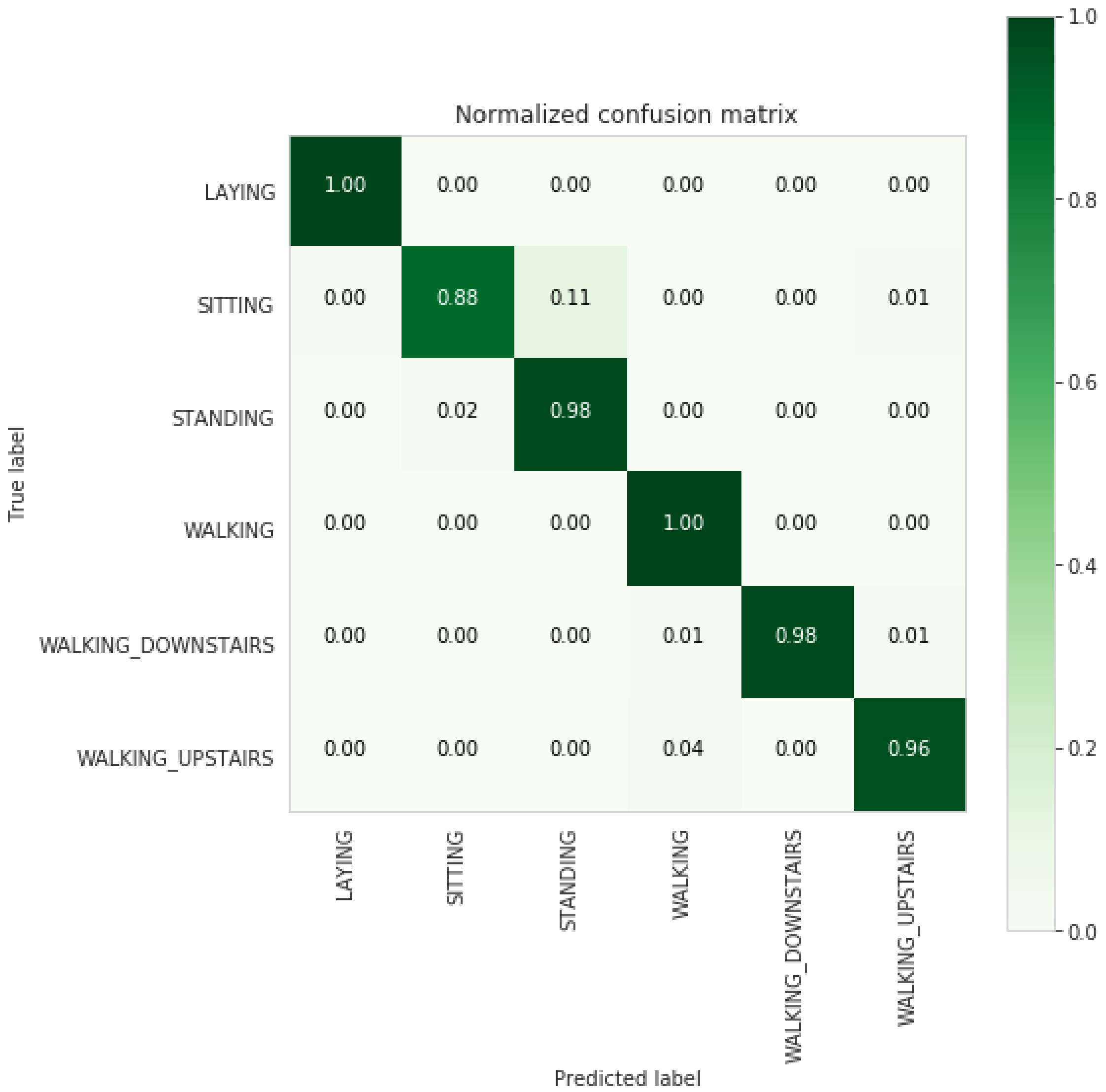
```
lr_svc = LinearSVC(tol=0.00005)
lr_svc_grid = GridSearchCV(lr_svc, param_grid=parameters, n_jobs=-1, ve
rbose=1)
lr_svc_grid_results = perform_model(lr_svc_grid, X_train, y_train, X_te
st, y_test, class_labels=labels)
```

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

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  18 out of  18 | elapsed:   33.4s finished
```

```
Done


training_time(HH:MM:SS.ms) - 0:00:37.538439


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.006808


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

    0.9674244994910078


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

 [[537   0   0   0   0   0]
 [  2 433  53   0   0   3]
 [  0  12 519   1   0   0]
 [  0   0   0 496   0   0]
```

```
[  0   0   0   3 412   5]
[  0   0   0  17   0 454]]
```




```
-------------------------
| Classifiction Report |
```

```
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.97      0.88      0.93       491
          STANDING       0.91      0.98      0.94       532
           WALKING       0.96      1.00      0.98       496
WALKING_DOWNSTAIRS       1.00      0.98      0.99       420
  WALKING_UPSTAIRS       0.98      0.96      0.97       471

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

In [51]:
```
print_grid_search_attributes(lr_svc_grid_results['model'])
```

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

        LinearSVC(C=0.5, class_weight=None, dual=True, fit_intercept=Tr
ue,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=5e-0
5,
          verbose=0)

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

        {'C': 0.5}

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

```
        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.9457290533188248
```

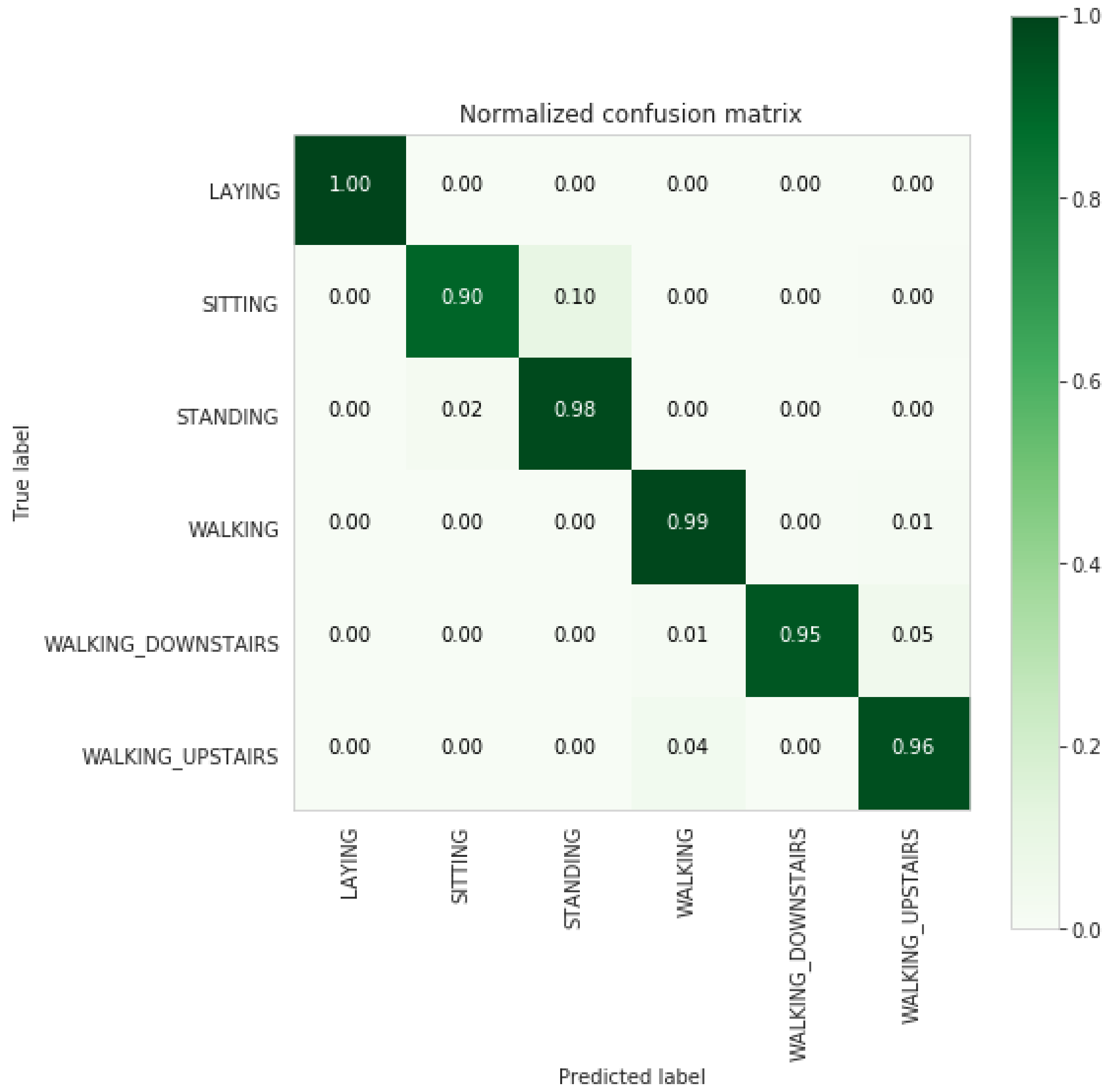
3. Kernel SVM with GridSearch

In [52]:

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

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:05:38.115511


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:02.566120


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




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

            LAYING       1.00      1.00      1.00       537
```

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

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

In [53]:
```python
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='r
bf',
    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       |
--------------------------
```

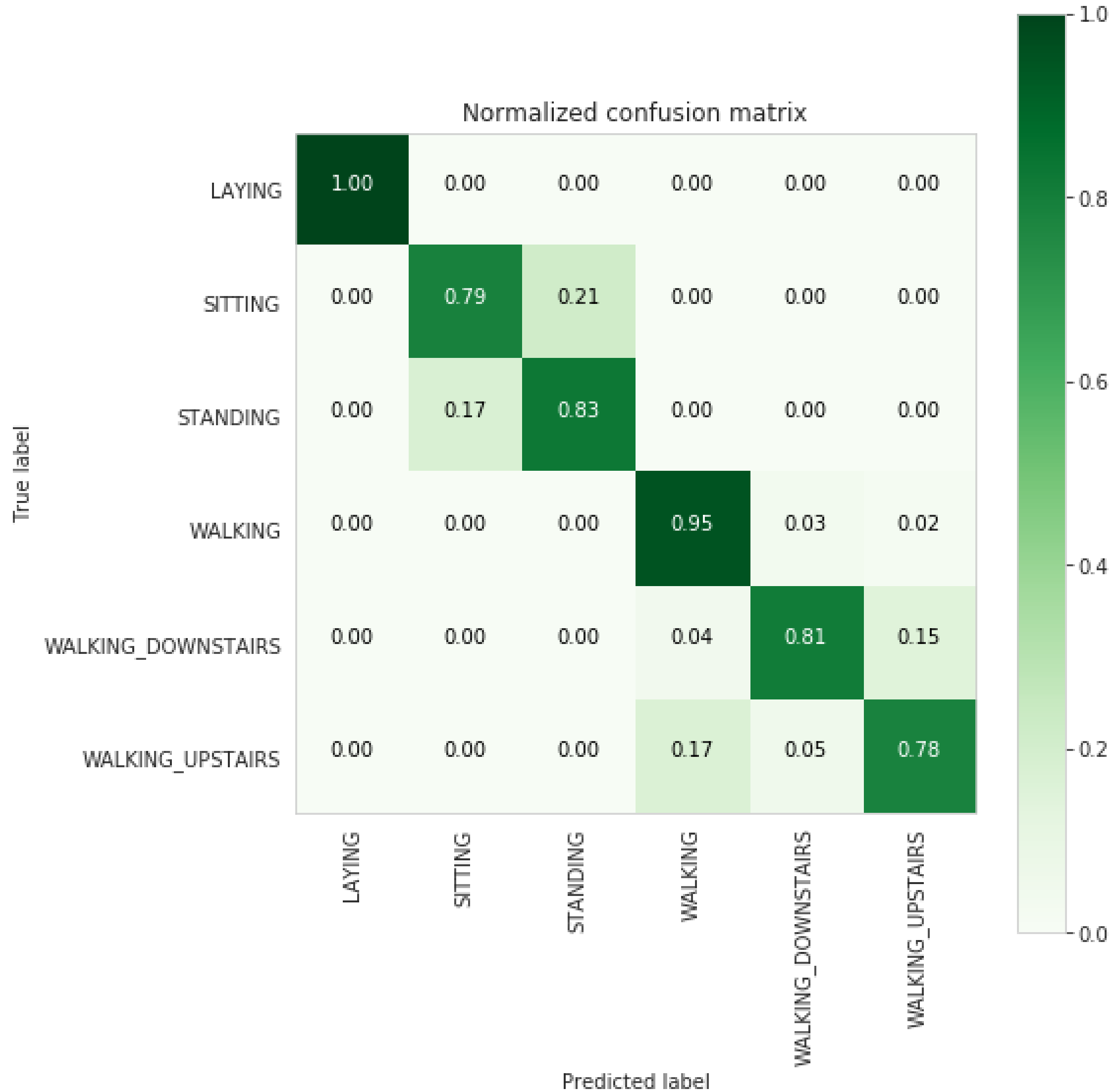
```
Average Cross Validate scores of best estimator :

0.9440968443960827
```

4. Decision Trees with GridSearchCV

In [54]:
```
%%time
from sklearn.tree import DecisionTreeClassifier
parameters = {'max_depth':np.arange(3,10,2)}
dt = DecisionTreeClassifier()
dt_grid = GridSearchCV(dt,param_grid=parameters, n_jobs=-1)
dt_grid_results = perform_model(dt_grid, X_train, y_train, X_test, y_te
st, class_labels=labels)
print_grid_search_attributes(dt_grid_results['model'])
```

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:00:14.994724


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.005478


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

    0.8632507634882932
```

```
--------------------
| 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  18 341  61]
 [  0   0   0  78  24 369]]
```




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

            LAYING       1.00      1.00      1.00       537
```

```
            SITTING       0.81      0.79      0.80       491
           STANDING       0.81      0.83      0.82       532
            WALKING       0.83      0.95      0.89       496
 WALKING_DOWNSTAIRS       0.90      0.81      0.85       420
   WALKING_UPSTAIRS       0.84      0.78      0.81       471

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

--------------------------
|      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=No
ne,
                       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 :

        0.838139281828074

CPU times: user 3.12 s, sys: 140 ms, total: 3.26 s
Wall time: 15.5 s
```

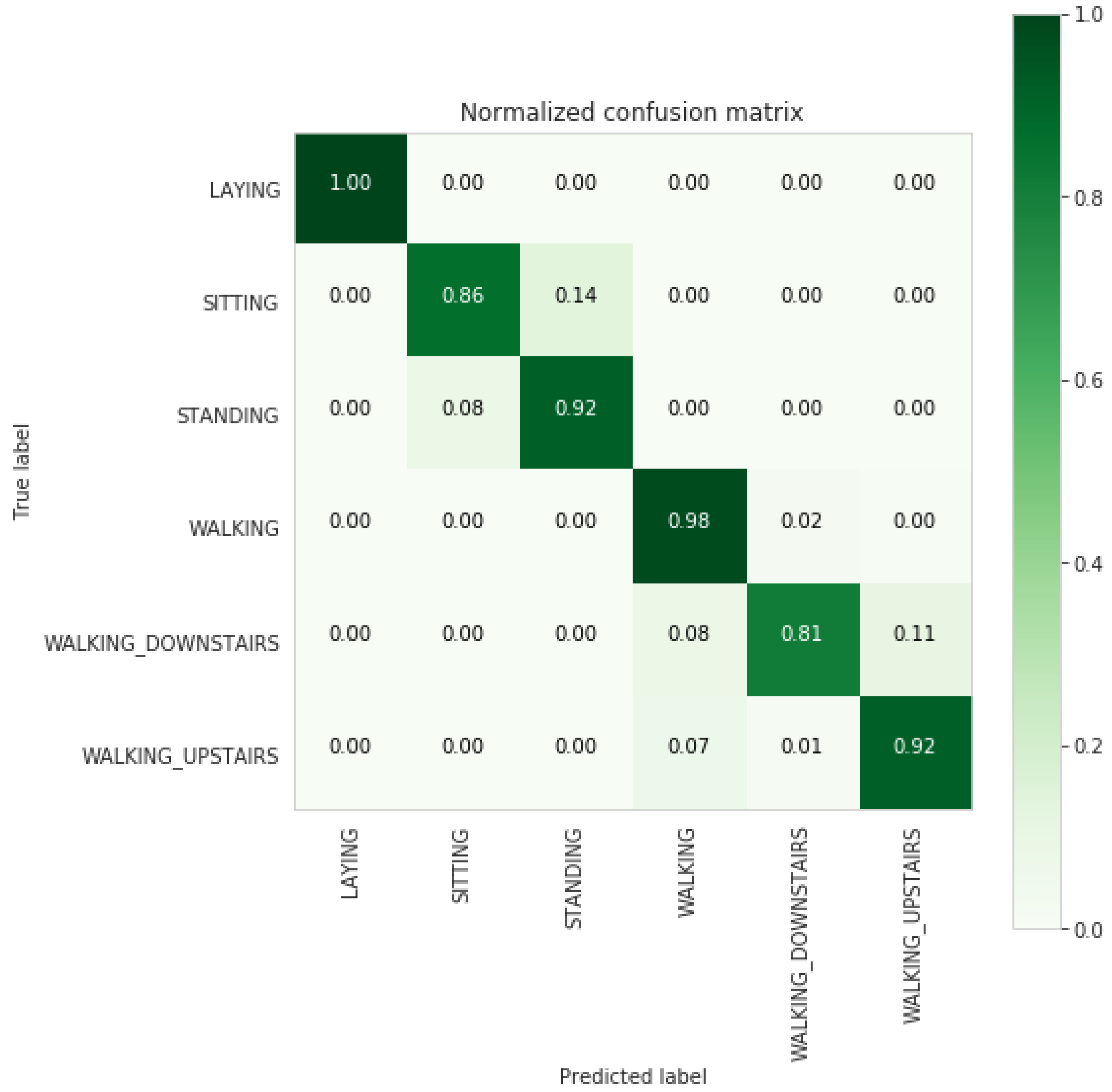
5. Random Forest Classifier with GridSearch

In [55]:

```
from sklearn.ensemble import RandomForestClassifier
params = {'n_estimators': np.arange(10,201,20), 'max_depth':np.arange(3
,15,2)}
rfc = RandomForestClassifier()
rfc_grid = GridSearchCV(rfc, param_grid=params, n_jobs=-1)
rfc_grid_results = perform_model(rfc_grid, X_train, y_train, X_test, y_
test, class_labels=labels)
print_grid_search_attributes(rfc_grid_results['model'])
```

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:10:13.458776


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.037720


---------------------
|      Accuracy      |
---------------------
```

```
0.9175432643366135


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

 [[537   0   0   0   0   0]
 [  0 424  67   0   0   0]
 [  0  45 487   0   0   0]
 [  0   0   0 484  10   2]
 [  0   0   0  32 341  47]
 [  0   0   0  34   6 431]]
```




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

```
              LAYING       1.00      1.00      1.00       537
             SITTING       0.90      0.86      0.88       491
            STANDING       0.88      0.92      0.90       532
             WALKING       0.88      0.98      0.93       496
  WALKING_DOWNSTAIRS       0.96      0.81      0.88       420
    WALKING_UPSTAIRS       0.90      0.92      0.91       471

            accuracy                           0.92      2947
           macro avg       0.92      0.91      0.91      2947
        weighted avg       0.92      0.92      0.92      2947

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

        RandomForestClassifier(bootstrap=True, class_weight=None, crite
rion='gini',
                       max_depth=7, max_features='auto', max_leaf_nodes
=None,
                       min_impurity_decrease=0.0, min_impurity_split=No
ne,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=110,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)

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

        {'max_depth': 7, 'n_estimators': 110}

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

        Total numbre of cross validation sets: 3
```

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

                Average Cross Validate scores of best estimator :

                0.9140369967355821
```

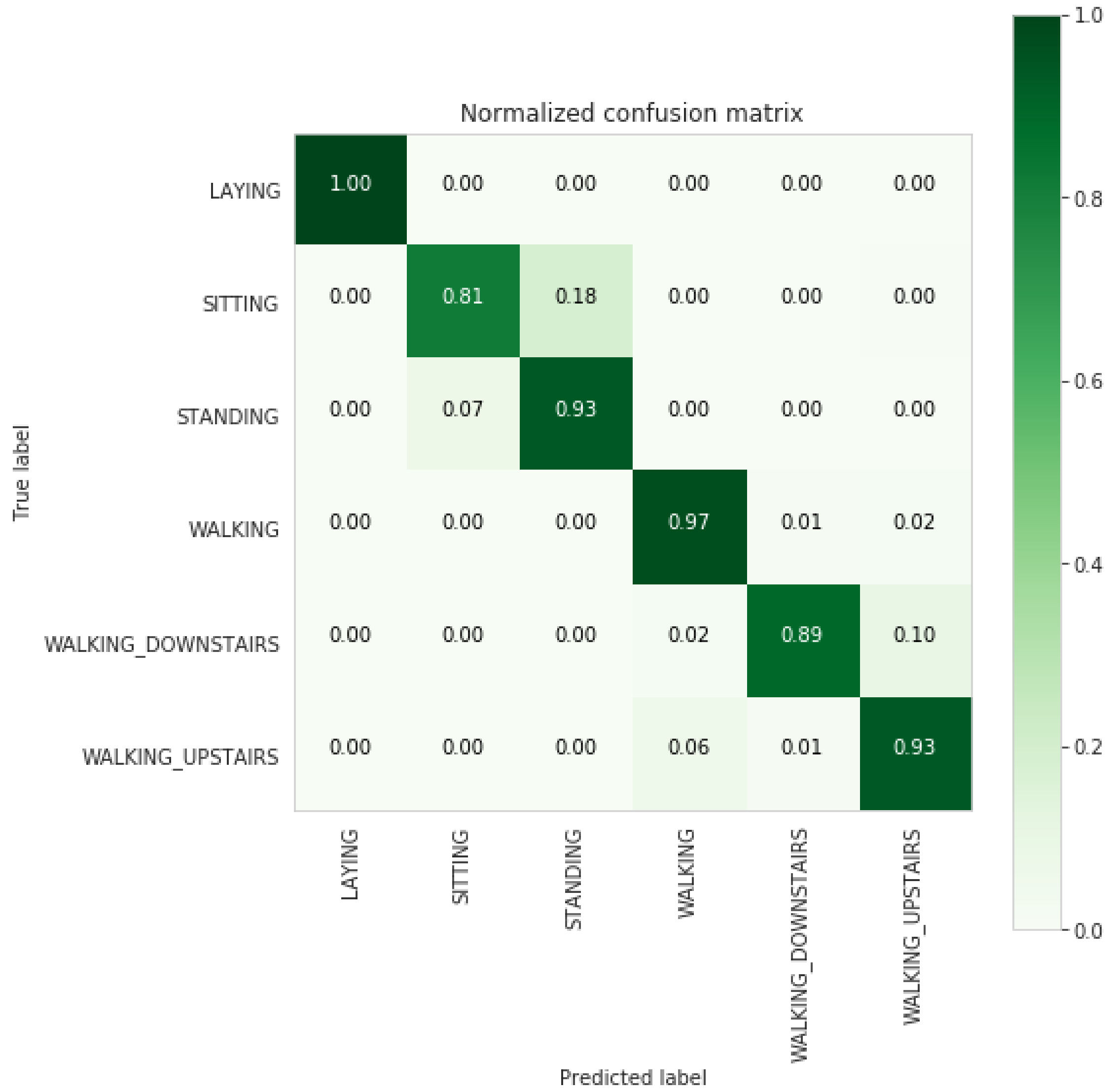
6. Gradient Boosted Decision Trees With GridSearch

In [56]:

```
from sklearn.ensemble import GradientBoostingClassifier
param_grid = {'max_depth': np.arange(5,8,1), 'n_estimators':np.arange(130,170,10)}

gbdt = GradientBoostingClassifier()
gbdt_grid = GridSearchCV(gbdt, param_grid=param_grid, n_jobs=-1)

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

```
training the model..
Done


training_time(HH:MM:SS.ms) - 1:20:42.597410


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.058005
```

```
---------------------
|      Accuracy      |
---------------------

    0.9239904988123515


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

 [[537   0   0   0   0   0]
 [  0 400  89   0   0   2]
 [  0  37 495   0   0   0]
 [  0   0   0 481   7   8]
 [  0   0   0   8 372  40]
 [  0   1   0  28   4 438]]
```




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

            LAYING       1.00      1.00      1.00       537
```

```
           LAYING       1.00      1.00      1.00       537
          SITTING       0.91      0.81      0.86       491
         STANDING       0.85      0.93      0.89       532
          WALKING       0.93      0.97      0.95       496
WALKING_DOWNSTAIRS       0.97      0.89      0.93       420
  WALKING_UPSTAIRS       0.90      0.93      0.91       471

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

In [59]:
```
print_grid_search_attributes(gbdt_grid_results["model"])
```

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

        GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_dept
h=5,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_spli
t=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=1
50,
                           n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0.000
1,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)

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

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

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.9030195865070729
```

7. Comparing all models

In [60]:
```
print('\n                              Accuracy     Error')
print('                            ----------   --------')
print('Logistic Regression : {:.04}%       {:.04}%'.format(log_reg_grid
_results['accuracy'] * 100,\
                                                       100-(log_reg_grid_res
ults['accuracy'] * 100)))

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

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

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

```
                                                                100-(dt_grid_re
sults['accuracy'] * 100)))

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

```
                      Accuracy     Error
                      ----------   --------
Logistic Regression : 96.27%       3.733%
Linear SVC          : 96.74%       3.258%
rbf SVM classifier  : 96.27%      3.733%
DecisionTree        : 86.33%      13.67%
Random Forest       : 91.75%      8.246%
GradientBoosting DT : 91.75%      8.246%
```

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

Conclusion :

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

LSTM

In [0]:

```
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
```

```
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

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

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=['Pr
ed'])
```

Data

In [0]:
```
# Data directory
DATADIR = '/content/drive/My Drive/HumanActivityRecognition.zip (Unzipp
```

```
ed Files)/HAR/UCI_HAR_Dataset'
```

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'/content/drive/My Drive/HumanActivityRecognition.z
ip (Unzipped Files)/HAR/UCI_HAR_Dataset/{subset}/Inertial Signals/{sign
al}_{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'/content/drive/My Drive/HumanActivityRecognition.zip (Unzipped Files)/HAR/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.set_random_seed(42)
```

In [0]:
```
# Configuring a session
session_conf = tf.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]:
```
# Importing libraries
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout
```

In [0]:
```
# Defining 'plt_la' function

# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error

def plt_la(x, vy, ty, ax, t, colors=['b']):

  if t == 'loss':
    ax.plot(x, vy, 'b', label="Validation Loss")
    ax.plot(x, ty, 'r', label="Train Loss")
    plt.title("Epoch vs Loss")
    plt.legend()
    plt.grid()

  if t == 'acc':
    ax.plot(x, vy, 'b', label="Validation Accuracy")
    ax.plot(x, ty, 'r', label="Train Accuracy")
    plt.title("Epoch vs Accuracy")
    plt.legend()
    plt.grid()
```

In [0]:
```
# Defining a function 'plotting' to visualize epoch vs loss

def plotting(history, t):
```

```
  fig,ax = plt.subplots(1,1)
  ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy')

  # list of epoch numbers
  x = list(range(1,epochs+1))

  # print(history.history.keys())
  # dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
  # history = model_drop.fit(X_train, Y_train, batch_size=batch_size, e
pochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

  # we will get val_loss and val_acc only when you pass the paramter va
lidation_data
  # val_loss : validation loss
  # val_acc : validation accuracy

  # loss : training loss
  # acc : train accuracy
  # for each key in histrory.histrory we will have a list of length equ
al to number of epochs

  if t == 'loss':
    vy = history.history['val_loss']
    ty = history.history['loss']

    plt_la(x, vy, ty, ax, t)

  if t == 'acc':
    vy = history.history['val_acc']
    ty = history.history['acc']

    plt_la(x, vy, ty, ax, t)

  return vy, ty
```

In [0]:

```
# Initializing parameters
epochs = 15
batch_size = 16
```

In [0]:
```python
# Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

Train test split

In [0]:
```python
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

In [81]:
```python
len(X_train[0][0])
```

Out[81]: 9

In [82]:
```python
timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = _count_classes(Y_train)

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

```
128
9
7352
```

Model-1

In [83]:
```python
n_hidden = 64

model_1 = Sequential()

# 1 LSTM layer
model_1.add(LSTM(n_hidden, input_shape = (timesteps, input_dim)))     #
 1 LSTM
```

```
model_1.add(Dropout(0.25))
model_1.add(Dense(n_classes, activation = 'sigmoid'))
model_1.compile(loss = 'binary_crossentropy', optimizer = 'rmsprop', me
trics = ['accuracy'])
print(model_1.summary())
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/ba
ckend/tensorflow_backend.py:541: The name tf.placeholder is deprecated.
Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/ba
ckend/tensorflow_backend.py:4432: The name tf.random_uniform is depreca
ted. Please use tf.random.uniform instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/ba
ckend/tensorflow_backend.py:148: The name tf.placeholder_with_default i
s deprecated. Please use tf.compat.v1.placeholder_with_default instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/ba
ckend/tensorflow_backend.py:3733: calling dropout (from tensorflow.pyth
on.ops.nn_ops) with keep_prob is deprecated and will be removed in a fu
ture version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate =
1 - keep_prob`.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/op
timizers.py:793: The name tf.train.Optimizer is deprecated. Please use
tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/ba
ckend/tensorflow_backend.py:3657: The name tf.log is deprecated. Please
use tf.math.log instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorfl
ow_core/python/ops/nn_impl.py:183: where (from tensorflow.python.ops.ar
ray_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Model: "sequential_1"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_1 (LSTM)                (None, 64)                18944     
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 390       
=================================================================
Total params: 19,334
Trainable params: 19,334
Non-trainable params: 0
_________________________________________________________________
None
```

In [84]:

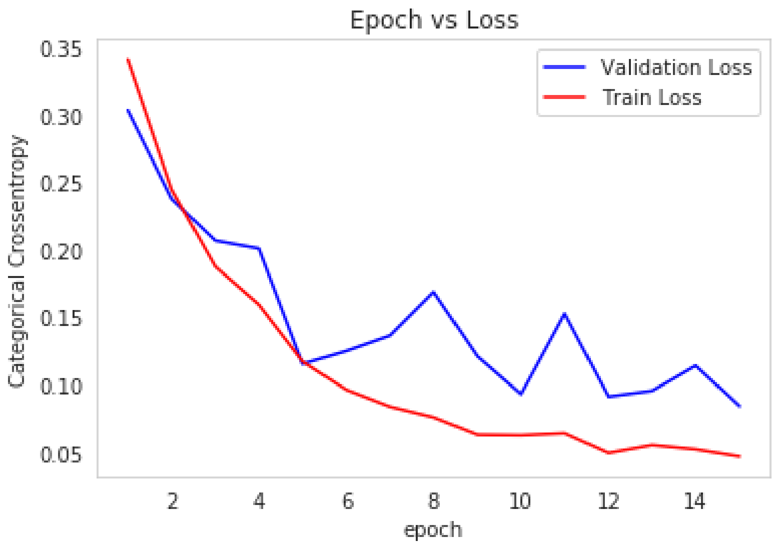
```
history_1 = model_1.fit(X_train, Y_train, epochs = epochs, batch_size =
 batch_size, validation_data = (X_test, Y_test))

# Final evaluation of the model
scores_1 = model_1.evaluate(X_test, Y_test, verbose = 1)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/ba
ckend/tensorflow_backend.py:1033: The name tf.assign_add is deprecated.
Please use tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/ba
ckend/tensorflow_backend.py:1020: The name tf.assign is deprecated. Ple
ase use tf.compat.v1.assign instead.

Train on 7352 samples, validate on 2947 samples
Epoch 1/15
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/ba
ckend/tensorflow_backend.py:190: The name tf.get_default_session is dep
recated. Please use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/ba
ckend/tensorflow_backend.py:207: The name tf.global_variables is deprec
ated. Please use tf.compat.v1.global_variables instead.
```

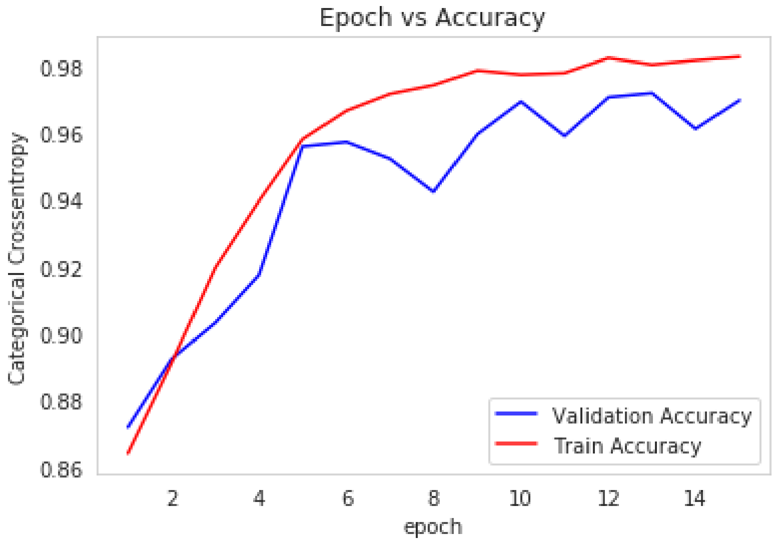
```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/ba
ckend/tensorflow_backend.py:216: The name tf.is_variable_initialized is
deprecated. Please use tf.compat.v1.is_variable_initialized instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/ba
ckend/tensorflow_backend.py:223: The name tf.variables_initializer is d
eprecated. Please use tf.compat.v1.variables_initializer instead.

7352/7352 [==============================] - 90s 12ms/step - loss: 0.34
10 - acc: 0.8642 - val_loss: 0.3033 - val_acc: 0.8721
Epoch 2/15
7352/7352 [==============================] - 82s 11ms/step - loss: 0.24
49 - acc: 0.8912 - val_loss: 0.2378 - val_acc: 0.8924
Epoch 3/15
7352/7352 [==============================] - 81s 11ms/step - loss: 0.18
76 - acc: 0.9198 - val_loss: 0.2067 - val_acc: 0.9033
Epoch 4/15
7352/7352 [==============================] - 81s 11ms/step - loss: 0.15
93 - acc: 0.9397 - val_loss: 0.2008 - val_acc: 0.9175
Epoch 5/15
7352/7352 [==============================] - 81s 11ms/step - loss: 0.11
73 - acc: 0.9582 - val_loss: 0.1154 - val_acc: 0.9559
Epoch 6/15
7352/7352 [==============================] - 82s 11ms/step - loss: 0.09
61 - acc: 0.9667 - val_loss: 0.1250 - val_acc: 0.9573
Epoch 7/15
7352/7352 [==============================] - 81s 11ms/step - loss: 0.08
36 - acc: 0.9717 - val_loss: 0.1364 - val_acc: 0.9523
Epoch 8/15
7352/7352 [==============================] - 81s 11ms/step - loss: 0.07
57 - acc: 0.9743 - val_loss: 0.1687 - val_acc: 0.9424
Epoch 9/15
7352/7352 [==============================] - 81s 11ms/step - loss: 0.06
30 - acc: 0.9787 - val_loss: 0.1210 - val_acc: 0.9597
Epoch 10/15
7352/7352 [==============================] - 82s 11ms/step - loss: 0.06
26 - acc: 0.9774 - val_loss: 0.0929 - val_acc: 0.9694
Epoch 11/15
7352/7352 [==============================] - 82s 11ms/step - loss: 0.06
```

```
40 - acc: 0.9779 - val_loss: 0.1527 - val_acc: 0.9591
Epoch 12/15
7352/7352 [==============================] - 82s 11ms/step - loss: 0.04
94 - acc: 0.9824 - val_loss: 0.0911 - val_acc: 0.9707
Epoch 13/15
7352/7352 [==============================] - 83s 11ms/step - loss: 0.05
52 - acc: 0.9804 - val_loss: 0.0954 - val_acc: 0.9719
Epoch 14/15
7352/7352 [==============================] - 83s 11ms/step - loss: 0.05
21 - acc: 0.9817 - val_loss: 0.1142 - val_acc: 0.9613
Epoch 15/15
7352/7352 [==============================] - 83s 11ms/step - loss: 0.04
69 - acc: 0.9828 - val_loss: 0.0840 - val_acc: 0.9698
2947/2947 [==============================] - 6s 2ms/step
```

In [85]:
```
v_l_1, t_l_1 = plotting(history_1, 'loss')
```



In [86]:
```
v_a_1, t_a_1 = plotting(history_1, 'acc')
```



In [87]:
```
tr_a_1 = np.round(max(t_a_1),3)
va_a_1 = np.round(max(v_a_1),3)

print("Train accuracy:", tr_a_1)
print("Validation accuracy:", va_a_1, '\n')

tr_l_1 = np.round(min(t_l_1),3)
va_l_1 = np.round(min(v_l_1),3)

print("Train loss:", tr_l_1)
print("Validation loss:", va_l_1)
```

```
Train accuracy: 0.983
Validation accuracy: 0.972

Train loss: 0.047
Validation loss: 0.084
```

In [89]:
```
# Confusion Matrix
print(confusion_matrix(Y_test, model_1.predict(X_test)))
```

```
Pred                LAYING  SITTING  ...  WALKING_DOWNSTAIRS  WALKING_U
```

```
PSTAIRS
True                                     ...

LAYING                      536         0  ...                         0
      1
SITTING                       0       372  ...                         1
      2
STANDING                      0        73  ...                         0
      0
WALKING                       0         0  ...                        15
     10
WALKING_DOWNSTAIRS            0         0  ...                       417
      2
WALKING_UPSTAIRS              0         0  ...                        11
    430

[6 rows x 6 columns]
```

Observation:

From the above confusion matrix, laying is very well predicted and sitting showing more errors when we compare to the other activities

Model-2

In [90]:
```
n_hidden_2 = 100

model_2 = Sequential()

# 2 LSTM layer
model_2.add(LSTM(n_hidden_2, input_shape = (timesteps, input_dim), retu
rn_sequences = True))  # 1 LSTM
model_2.add(Dropout(0.50))
model_2.add(LSTM(n_hidden_2))                         # 2 LSTM
```

```
model_2.add(Dropout(0.50))
model_2.add(Dense(n_classes, activation = 'sigmoid'))
model_2.compile(loss = 'binary_crossentropy', optimizer = 'rmsprop', me
trics = ['accuracy'])
print(model_2.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_2 (LSTM)                (None, 128, 100)          44000     
_________________________________________________________________
dropout_2 (Dropout)          (None, 128, 100)          0         
_________________________________________________________________
lstm_3 (LSTM)                (None, 100)               80400     
_________________________________________________________________
dropout_3 (Dropout)          (None, 100)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 606       
=================================================================
Total params: 125,006
Trainable params: 125,006
Non-trainable params: 0
_________________________________________________________________
None
```

In [91]:

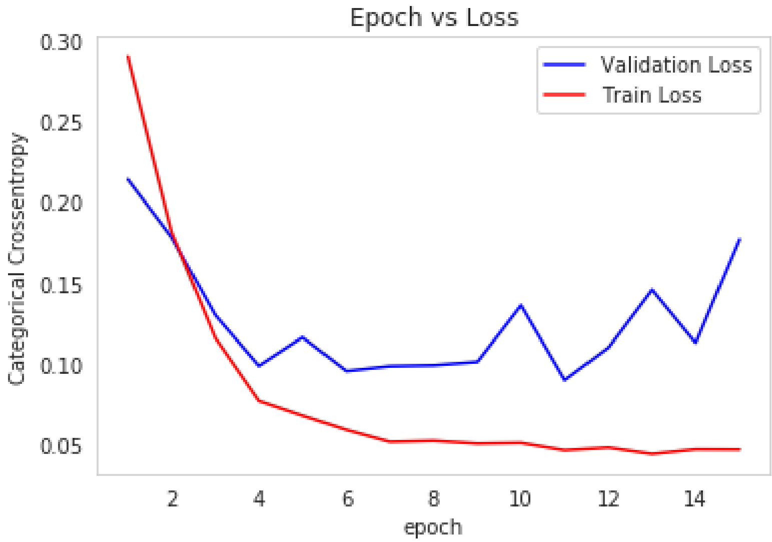
```
history_2 = model_2.fit(X_train, Y_train, epochs = epochs, batch_size =
 batch_size , validation_data = (X_test, Y_test))

# Final evaluation of the model
scores_2 = model_2.evaluate(X_test, Y_test, verbose = 1)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/15
7352/7352 [==============================] - 169s 23ms/step - loss: 0.2
895 - acc: 0.8736 - val_loss: 0.2139 - val_acc: 0.9126
Epoch 2/15
7352/7352 [==============================] - 168s 23ms/step - loss: 0.1
815 - acc: 0.9280 - val_loss: 0.1783 - val_acc: 0.9369

Epoch 3/15
```

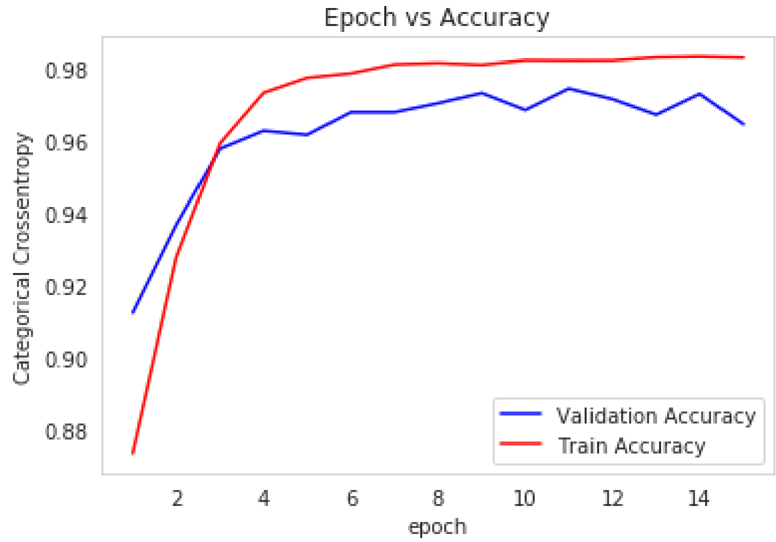
```
Epoch 3/15
7352/7352 [==============================] - 173s 23ms/step - loss: 0.1
162 - acc: 0.9596 - val_loss: 0.1303 - val_acc: 0.9580
Epoch 4/15
7352/7352 [==============================] - 169s 23ms/step - loss: 0.0
774 - acc: 0.9736 - val_loss: 0.0989 - val_acc: 0.9630
Epoch 5/15
7352/7352 [==============================] - 170s 23ms/step - loss: 0.0
683 - acc: 0.9777 - val_loss: 0.1165 - val_acc: 0.9619
Epoch 6/15
7352/7352 [==============================] - 169s 23ms/step - loss: 0.0
595 - acc: 0.9789 - val_loss: 0.0958 - val_acc: 0.9682
Epoch 7/15
7352/7352 [==============================] - 170s 23ms/step - loss: 0.0
522 - acc: 0.9814 - val_loss: 0.0988 - val_acc: 0.9682
Epoch 8/15
7352/7352 [==============================] - 171s 23ms/step - loss: 0.0
528 - acc: 0.9818 - val_loss: 0.0993 - val_acc: 0.9706
Epoch 9/15
7352/7352 [==============================] - 171s 23ms/step - loss: 0.0
511 - acc: 0.9813 - val_loss: 0.1014 - val_acc: 0.9735
Epoch 10/15
7352/7352 [==============================] - 170s 23ms/step - loss: 0.0
515 - acc: 0.9825 - val_loss: 0.1363 - val_acc: 0.9688
Epoch 11/15
7352/7352 [==============================] - 170s 23ms/step - loss: 0.0
469 - acc: 0.9825 - val_loss: 0.0902 - val_acc: 0.9747
Epoch 12/15
7352/7352 [==============================] - 171s 23ms/step - loss: 0.0
486 - acc: 0.9825 - val_loss: 0.1102 - val_acc: 0.9718
Epoch 13/15
7352/7352 [==============================] - 174s 24ms/step - loss: 0.0
447 - acc: 0.9833 - val_loss: 0.1458 - val_acc: 0.9675
Epoch 14/15
7352/7352 [==============================] - 172s 23ms/step - loss: 0.0
475 - acc: 0.9835 - val_loss: 0.1131 - val_acc: 0.9732
Epoch 15/15
7352/7352 [==============================] - 172s 23ms/step - loss: 0.0


474 - acc: 0.9833 - val_loss: 0.1770 - val_acc: 0.9648
```

```
474 - acc: 0.9833 - val_loss: 0.1770 - val_acc: 0.9048
2947/2947 [==============================] - 12s 4ms/step
```

In [92]:
```
v_l_2, t_l_2 = plotting(history_2, 'loss')
```



In [93]:
```
v_a_2, t_a_2 = plotting(history_2, 'acc')
```



In [94]:
```
tr_a_2 = np.round(max(t_a_2),3)
va_a_2 = np.round(max(v_a_2),3)

print("Train accuracy:", tr_a_2)
print("Validation accuracy:", va_a_2, '\n')

tr_l_2 = np.round(min(t_l_2),3)
va_l_2 = np.round(min(v_a_2),3)

print("Train loss:", tr_l_2)
print("Validation loss:", va_l_2)
```

```
Train accuracy: 0.984
Validation accuracy: 0.975

Train loss: 0.045
Validation loss: 0.913
```

In [95]:
```
# Confusion Matrix
print(confusion_matrix(Y_test, model_2.predict(X_test)))
```

```
Pred                    LAYING  SITTING  ...  WALKING_DOWNSTAIRS  WALKING_U
```

```
PSTAIRS
True                                    ...

LAYING                   510        0   ...                      0
     27
SITTING                    4      370   ...                      0
      3
STANDING                   0       58   ...                      0
      0
WALKING                    0        0   ...                     11
     17
WALKING_DOWNSTAIRS         0        0   ...                    372
      8
WALKING_UPSTAIRS           0        0   ...                      9
    444

[6 rows x 6 columns]
```

Model-3

In [96]:

```
n_hidden_3 = 150

model_3 = Sequential()

# 3 LSTM layer
model_3.add(LSTM(n_hidden_3, input_shape = (timesteps, input_dim), retu
rn_sequences = True))  # 1 LSTM
model_3.add(Dropout(0.75))
model_3.add(LSTM(n_hidden_3, return_sequences = True))    # 2 LSTM
model_3.add(Dropout(0.75))
model_3.add(LSTM(n_hidden_3))                             # 3 LSTM

model_3.add(Dropout(0.75))
model_3.add(Dense(n_classes, activation = 'sigmoid'))
model_3.compile(loss = 'binary_crossentropy', optimizer = 'rmsprop', me
trics = ['accuracy'])
print(model_3.summary())
```

```
WARNING:tensorflow:Large dropout rate: 0.75 (>0.5). In TensorFlow 2.x,
dropout() uses dropout rate instead of keep_prob. Please ensure that th
is is intended.
WARNING:tensorflow:Large dropout rate: 0.75 (>0.5). In TensorFlow 2.x,
dropout() uses dropout rate instead of keep_prob. Please ensure that th
is is intended.
WARNING:tensorflow:Large dropout rate: 0.75 (>0.5). In TensorFlow 2.x,
dropout() uses dropout rate instead of keep_prob. Please ensure that th
is is intended.
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_4 (LSTM)                (None, 128, 150)          96000
_________________________________________________________________
dropout_4 (Dropout)          (None, 128, 150)          0
_________________________________________________________________
lstm_5 (LSTM)                (None, 128, 150)          180600
_________________________________________________________________
dropout_5 (Dropout)          (None, 128, 150)          0
_________________________________________________________________
lstm_6 (LSTM)                (None, 150)               180600
_________________________________________________________________
dropout_6 (Dropout)          (None, 150)               0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 906
=================================================================
Total params: 458,106
Trainable params: 458,106
Non-trainable params: 0
_________________________________________________________________
None
```

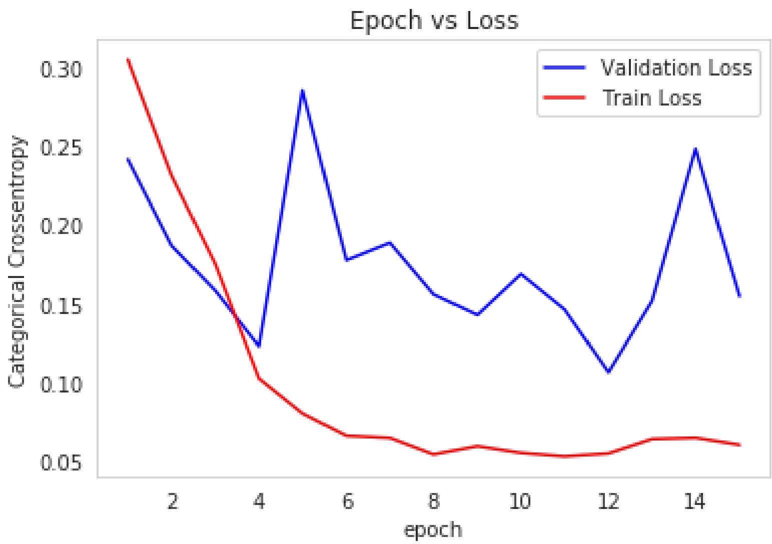
In [97]:

```
history_3 = model_3.fit(X_train, Y_train, epochs = epochs, batch_size =
 batch_size, validation_data = (X_test, Y_test))

# Final evaluation of the model
scores_3 = model_3.evaluate(X_test, Y_test, verbose = 1)
```

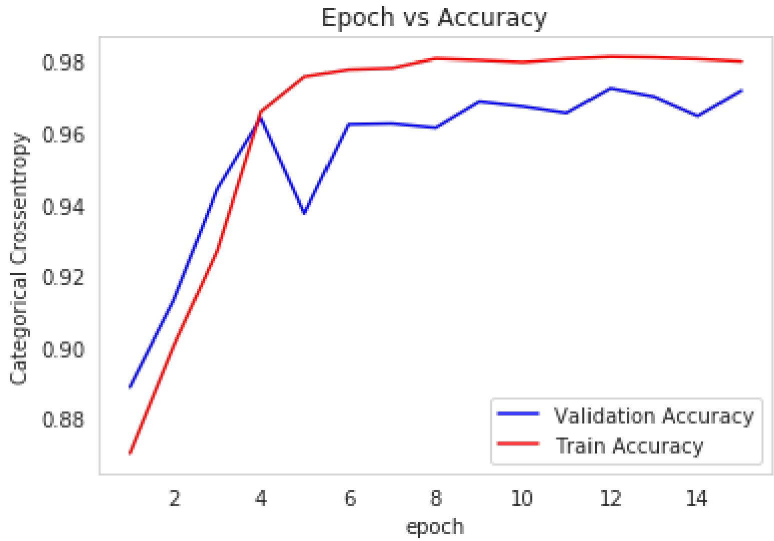
```
Train on 7352 samples, validate on 2947 samples
Epoch 1/15
7352/7352 [==============================] - 262s 36ms/step - loss: 0.3
047 - acc: 0.8703 - val_loss: 0.2415 - val_acc: 0.8889
Epoch 2/15
7352/7352 [==============================] - 257s 35ms/step - loss: 0.2
314 - acc: 0.9003 - val_loss: 0.1869 - val_acc: 0.9131
Epoch 3/15
7352/7352 [==============================] - 256s 35ms/step - loss: 0.1
751 - acc: 0.9270 - val_loss: 0.1582 - val_acc: 0.9442
Epoch 4/15
7352/7352 [==============================] - 257s 35ms/step - loss: 0.1
025 - acc: 0.9659 - val_loss: 0.1229 - val_acc: 0.9641
Epoch 5/15
7352/7352 [==============================] - 262s 36ms/step - loss: 0.0
802 - acc: 0.9757 - val_loss: 0.2854 - val_acc: 0.9375
Epoch 6/15
7352/7352 [==============================] - 264s 36ms/step - loss: 0.0
661 - acc: 0.9776 - val_loss: 0.1776 - val_acc: 0.9623
Epoch 7/15
7352/7352 [==============================] - 259s 35ms/step - loss: 0.0
647 - acc: 0.9781 - val_loss: 0.1887 - val_acc: 0.9626
Epoch 8/15
7352/7352 [==============================] - 263s 36ms/step - loss: 0.0
542 - acc: 0.9807 - val_loss: 0.1559 - val_acc: 0.9614
Epoch 9/15
7352/7352 [==============================] - 258s 35ms/step - loss: 0.0
594 - acc: 0.9803 - val_loss: 0.1428 - val_acc: 0.9687
Epoch 10/15
7352/7352 [==============================] - 254s 35ms/step - loss: 0.0
552 - acc: 0.9797 - val_loss: 0.1688 - val_acc: 0.9674
Epoch 11/15
7352/7352 [==============================] - 259s 35ms/step - loss: 0.0
530 - acc: 0.9806 - val_loss: 0.1463 - val_acc: 0.9655
Epoch 12/15
7352/7352 [==============================] - 268s 36ms/step - loss: 0.0
548 - acc: 0.9812 - val_loss: 0.1065 - val_acc: 0.9724
Epoch 13/15
7352/7352 [==============================] - 260s 35ms/step - loss: 0.0
```

```
639 - acc: 0.9810 - val_loss: 0.1517 - val_acc: 0.9700
Epoch 14/15
7352/7352 [==============================] - 262s 36ms/step - loss: 0.0
648 - acc: 0.9806 - val_loss: 0.2481 - val_acc: 0.9647
Epoch 15/15
7352/7352 [==============================] - 252s 34ms/step - loss: 0.0
603 - acc: 0.9799 - val_loss: 0.1547 - val_acc: 0.9718
2947/2947 [==============================] - 16s 5ms/step
```

In [98]:
```python
v_l_3, t_l_3 = plotting(history_3, 'loss')
```



In [99]:
```python
v_a_3, t_a_3 = plotting(history_3, 'acc')
```



In [100]:
```
tr_a_3 = np.round(max(t_a_3),3)
va_a_3 = np.round(max(v_a_3),3)

print("Train accuracy:", tr_a_3)
print("Validation accuracy:", va_a_3, '\n')

tr_l_3 = np.round(min(t_l_3),3)
va_l_3 = np.round(min(v_a_3),3)

print("Train loss:", tr_l_3)
print("Validation loss:", va_l_3)
```

```
Train accuracy: 0.981
Validation accuracy: 0.972

Train loss: 0.053
Validation loss: 0.889
```

In [101]:
```
# Confusion Matrix
print(confusion_matrix(Y_test, model_3.predict(X_test)))
```

```
Pred                    LAYING  SITTING  ...  WALKING_DOWNSTAIRS  WALKING_U
```

```
PSTAIRS
True                                     ...

LAYING                    537       0   ...                    0
      0
SITTING                     0     390   ...                    0
      1
STANDING                    0      65   ...                    0
      0
WALKING                     0       0   ...                   26
      0
WALKING_DOWNSTAIRS          0       0   ...                  417
      0
WALKING_UPSTAIRS            0       0   ...                   14
    424

[6 rows x 6 columns]
```

Result

In [102]:

```python
from prettytable import PrettyTable

print('\n')
a = PrettyTable()
a.field_names = ['S.No', 'LSTM Units', 'LSTM Layers', 'Drop Out', 'Test
 Loss', 'Test Accuracy']
a.add_row([1, 64, 1, 0.25, va_l_1, va_a_1])
a.add_row([2, 100, 2, 0.5, va_l_2, va_a_2])
a.add_row([3, 150, 3, 0.75, va_l_3, va_a_3])

print(a.get_string(title = "LSTM 1 and 3 Activation: sigmoid,    Optimi
zer: adam"))
```

```
+------+------------+-------------+----------+-----------+------------
--+
| S.No | LSTM Units | LSTM Layers | Drop Out | Test Loss | Test Accurac
y |
```

1	64	1	0.25	0.084	0.972
2	100	2	0.5	0.913	0.975
3	150	3	0.75	0.889	0.972

Procedure

1. Reading and storing the feature file containing all the names of the feature or columns in a list.

2. Import the independent data values- train and test, and then attach the data values to the column names with the feature list.

3. Similar process is followed to obtain dependent variables- train and test.

4. Checking the database size of both dependent (x) and independent (y) variables, such as train and test dataset.

5. Check the data value type.

6. Visualization with countplot graph of the count of independent variables and unique values. It's the same with the pie chart.

7. Visualizing the distribution to check if the dataset is balanced or imbalance and in what ratio it is imbalance.

8. Removing dashes '-' and spaces ( ) from feature names to shorten the featuren name length.

8. Checking if there are any null values. If found null values, either imputation is applied to fill the null values or else feature column(s) is removed depending on the count of null values. Also,

visualizing null values graphically.

9. Checking for duplicates. If found any, duplicates are removed. Static and Dynamic Activities

1. In static activities (sit, stand, lie down) motion information will not be very useful.

2. In the dynamic activities (Walking, WalkingUpstairs,WalkingDownstairs) motion info will be significant.

3. Visualising stationary activities (sitting, standing, laying) and moving activities (walking, walking downstairs, walking upstairs) to know how different these activities are.

4. With boxplot, visualising how magnitude of an acceleration can separate activities well. 5. With boxplot, visualising how position gravity acceleration component can separate activities well.

6. Using T-SNE, we are reducing 561 dimension to 2 dimension and visualising how well activities can be separated.

7. Visualizing in 2 dimension with change in perplexity to know which perplexity is better for visualising the separation of data.

LSTM:

1. Taking the raw data and only gyroscope and accelerometer related data as it is the main feature which actually records the stationary and/or motion or in other words 'activities'.

2. With confusion matrix, we are analyzing how our model is predicting i.e how activities are predicted corresponding to original activities and how well model is predicting correctly.

3. Tuning LSTM units with 64, 100 and 150.

4. Tuning dropout rates with 0.25, 0.5 and 0.75.

Conclusion -

For every hyperparameter tuning, we are visualizing how test loss and test accuracy is improing or deteriorating. At the end, concluding which hyperparameter is better to get better accuracy and lower test loss.