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

Y_Labels(Encoded)

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

Train and test data were saperated

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

Data

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

Data Size :

27 MB

Quick overview of the dataset :

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

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

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

Problem Framework

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

Problem Statement

- Given a new datapoint we have to predict the Activity

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

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

```
No of Features: 561
```

Obtain the train data

In [0]:
```
# get the data from txt files to pandas dataffame
X_train = pd.read_csv('UCI_HAR_dataset/train/X_train.txt', delim_whitespace=True, header=None, names=features)

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

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

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

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

Out[2]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	angle(tBodyAccMean,gravity)	angle(tBodyAccJerkMean),gravityMean)	angle(tBodyGyroMean,gravity
6015	0.2797	-0.004397	-0.10952	0.359081	0.119909	-0.177541	0.337963	0.066883	-0.221876	0.474093	...	0.049658	0.602595	0.6

1 rows × 564 columns

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

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

Obtain the test data

In [0]:
```
# get the data from txt files to pandas dataffame
X_test = pd.read_csv('UCI_HAR_dataset/test/X_test.txt', delim_whitespace=True, header=None, names=features)

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

# get y labels from the txt file
y_test = pd.read_csv('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()
```

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

Out[4]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X	...	angle(tBodyAccMean,gravity)	angle(tBodyAccJerkMean),gravityMean)	angle(tBodyGyroMean,gravity
2261	0.279196	-0.018261	-0.103376	-0.996955	-0.982959	-0.988239	-0.9972	-0.982509	-0.986964	-0.940634	...	-0.268441	-0.215632	-0.4

1 rows × 564 columns

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

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

Data Cleaning

1. Check for Duplicates

In [0]:
```
print('No of duplicates in train: {}'.format(sum(train.duplicated())))
print('No of duplicates in test : {}'.format(sum(test.duplicated())))
```

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

2. Checking for NaN/null values

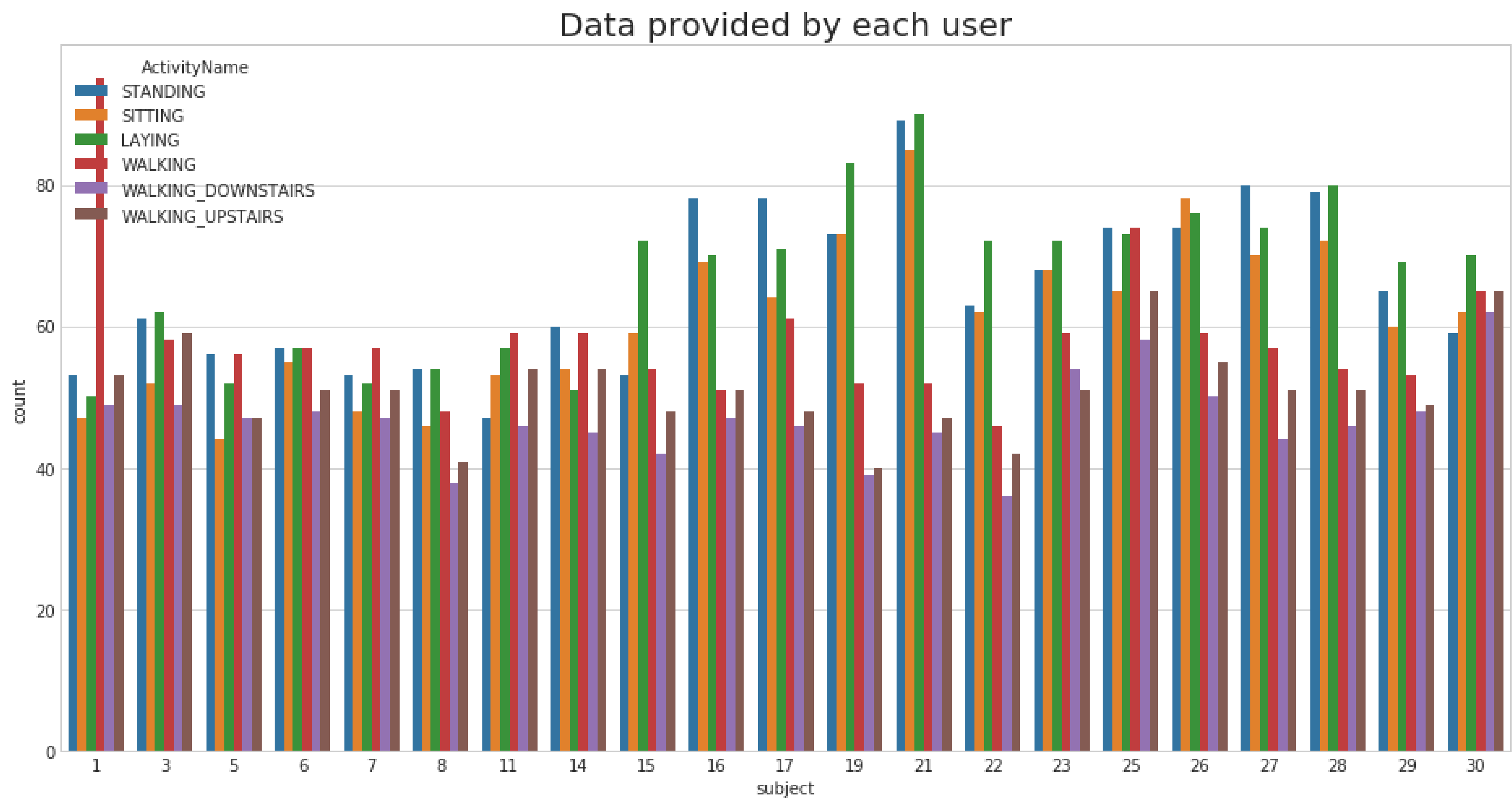
In [0]:
```
print('We have {} NaN/Null values in train'.format(train.isnull().values.sum()))
print('We have {} NaN/Null values in test'.format(test.isnull().values.sum()))
```

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

3. Check for data imbalance

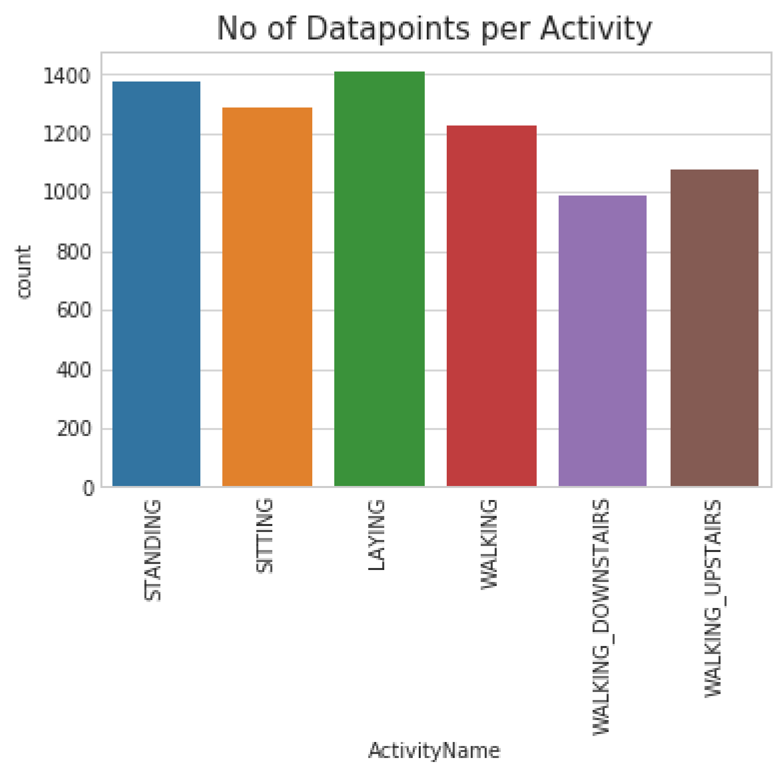
In [0]:
```
import matplotlib.pyplot as plt
import seaborn as sns

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

In [0]:
```
plt.figure(figsize=(16,8))
plt.title('Data provided by each user', fontsize=20)
sns.countplot(x='subject',hue='ActivityName', data = train)
plt.show()
```



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

In [0]:
```
plt.title('No of Datapoints per Activity', fontsize=15)
sns.countplot(train.ActivityName)
plt.xticks(rotation=90)
plt.show()
```



Observation

Our data is well balanced (almost)

4. Changing feature names

In [0]:
```
columns = train.columns

# Removing '()' from column names
columns = columns.str.replace('[()]','')
columns = columns.str.replace('[-]', '')
columns = columns.str.replace('[,]','')

train.columns = columns
test.columns = columns

test.columns
```

Out[11]:
```
Index(['tBodyAccmeanX', 'tBodyAccmeanY', 'tBodyAccmeanZ', 'tBodyAccstdX',
       'tBodyAccstdY', 'tBodyAccstdZ', 'tBodyAccmadX', 'tBodyAccmadY',
       'tBodyAccmadZ', 'tBodyAccmaxX',
       ...
       'angletBodyAccMeangravity', 'angletBodyAccJerkMeangravityMean',
       'angletBodyGyroMeangravityMean', 'angletBodyGyroJerkMeangravityMean',
       'angleXgravityMean', 'angleYgravityMean', 'angleZgravityMean',
       'subject', 'Activity', 'ActivityName'],
      dtype='object', length=564)
```

5. Save this dataframe in a csv files

In [0]:
```
train.to_csv('UCI_HAR_Dataset/csv_files/train.csv', index=False)
test.to_csv('UCI_HAR_Dataset/csv_files/test.csv', index=False)
```

Exploratory Data Analysis

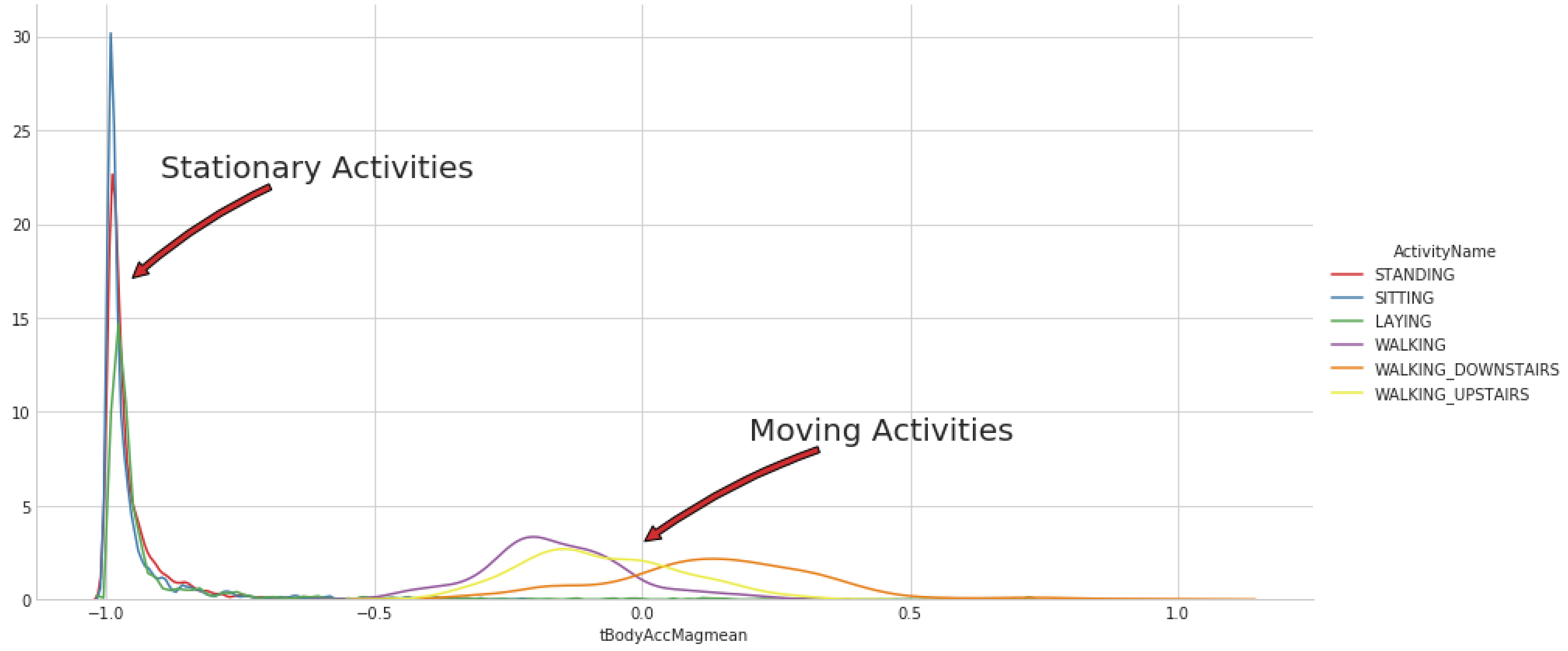
"***Without domain knowledge EDA has no meaning, without EDA a problem has no soul.***"

1. Featuring Engineering from Domain Knowledge

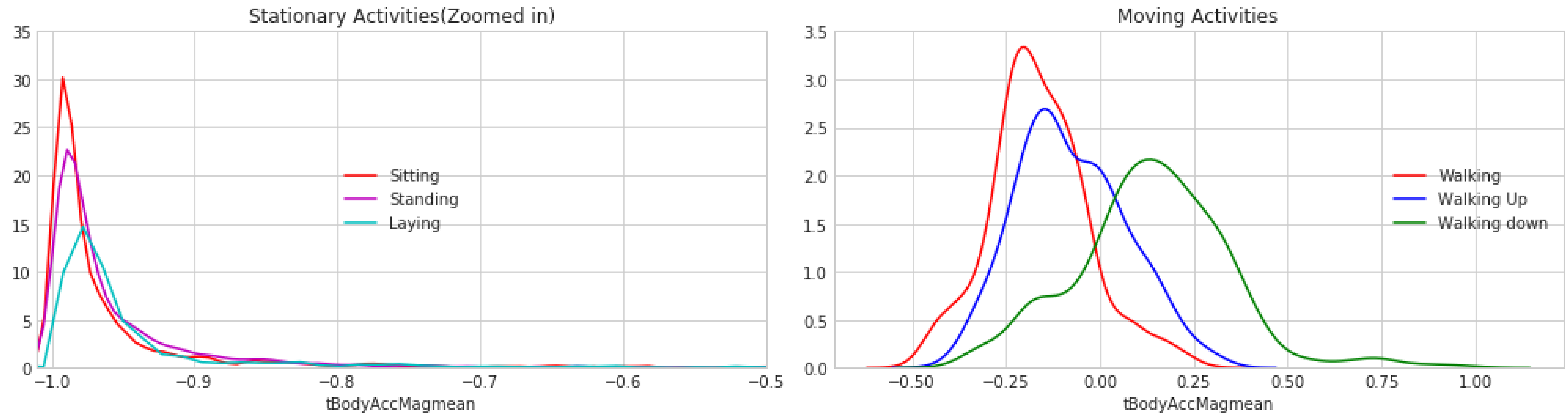
- **Static and Dynamic Activities**
  - In static activities (sit, stand, lie down) motion information will not be very useful.
  - In the dynamic activities (Walking, WalkingUpstairs,WalkingDownstairs) motion info will be significant.

2. Stationary and Moving activities are completely different

In [0]:
```
sns.set_palette("Set1", desat=0.80)
facetgrid = sns.FacetGrid(train, hue='ActivityName', size=6,aspect=2)
facetgrid.map(sns.distplot,'tBodyAccMagmean', hist=False)\
    .add_legend()
plt.annotate("Stationary Activities", xy=(-0.956,17), xytext=(-0.9, 23), size=20,\
            va='center', ha='left',\
            arrowprops=dict(arrowstyle="simple",connectionstyle="arc3,rad=0.1"))

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

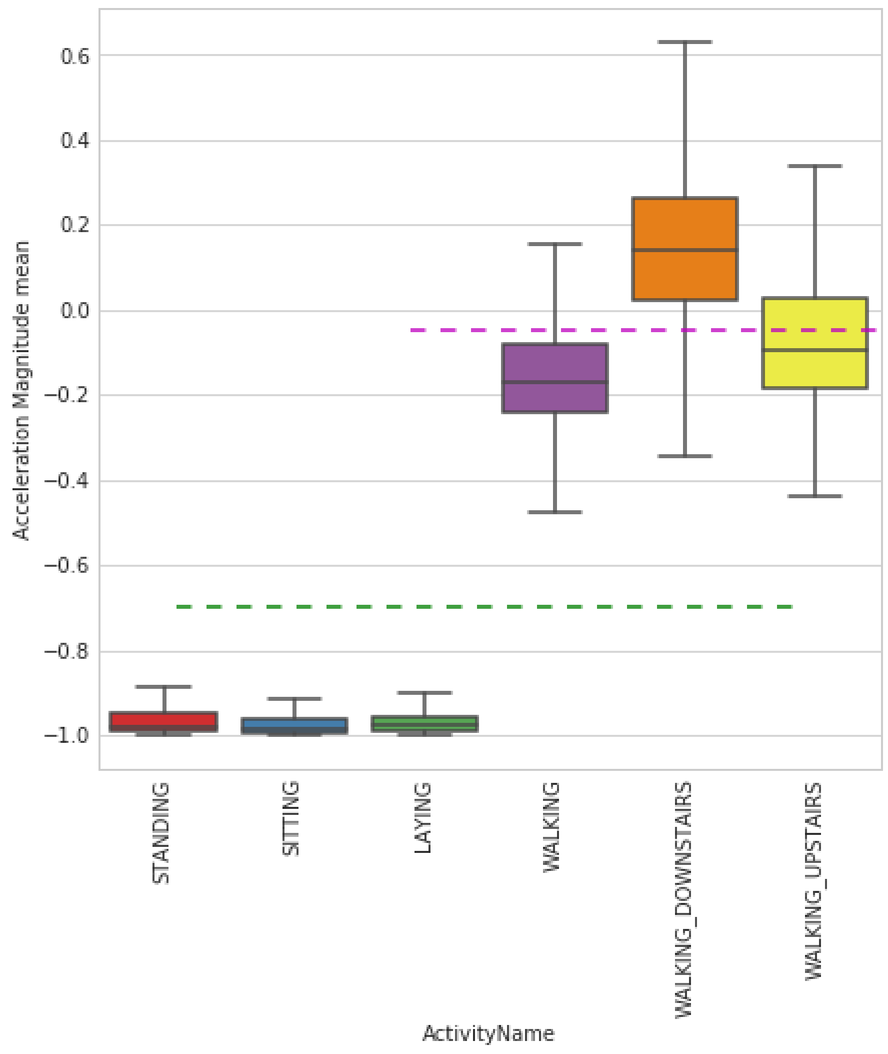


In [0]:
```
# for plotting purposes taking datapoints of each activity to a different dataframe
df1 = train[train['Activity']==1]
df2 = train[train['Activity']==2]
df3 = train[train['Activity']==3]
df4 = train[train['Activity']==4]
df5 = train[train['Activity']==5]
df6 = train[train['Activity']==6]

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

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


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



3. Magnitude of an acceleration can saperate it well

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

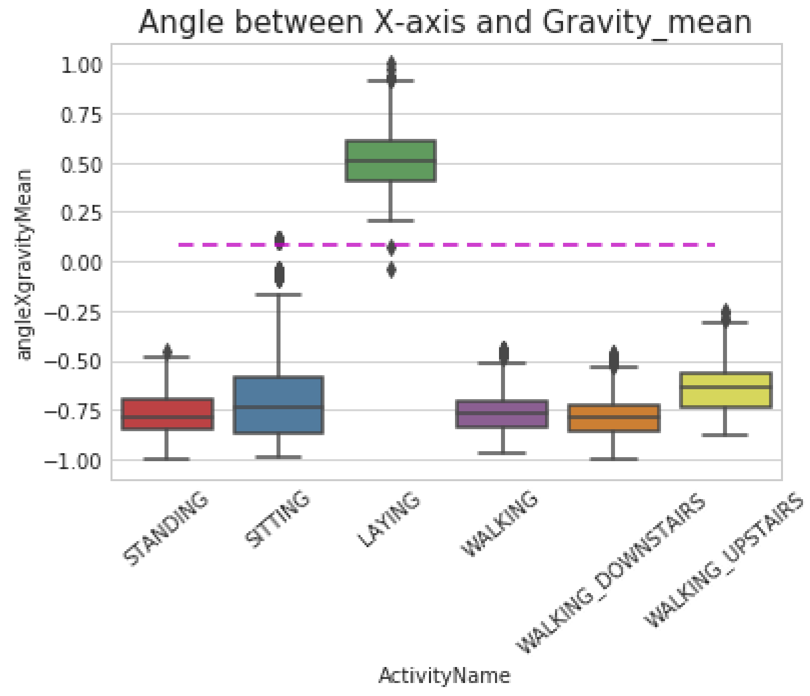


Observations:

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

4. Position of GravityAccelerationComponants also matters

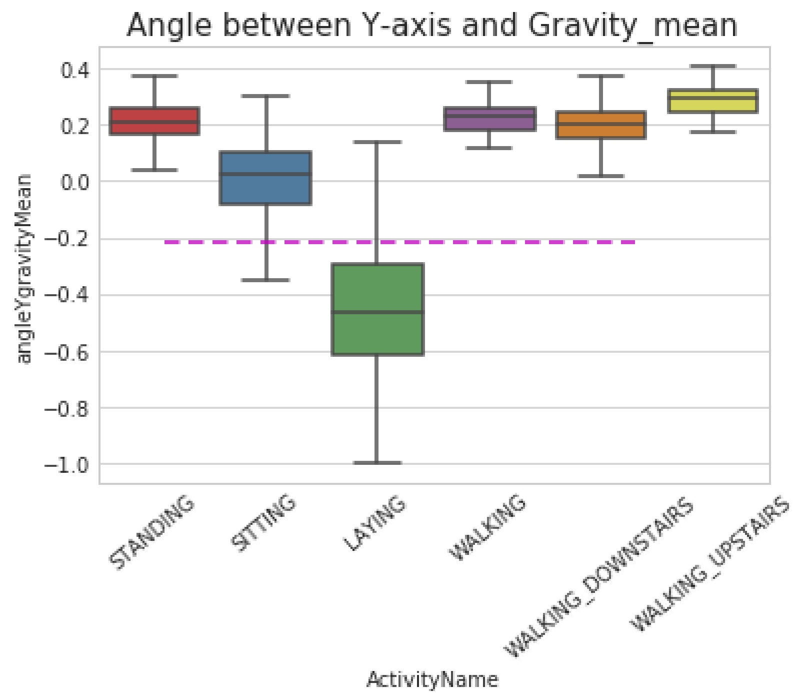
In [0]:
```
sns.boxplot(x='ActivityName', y='angleXgravityMean', data=train)
plt.axhline(y=0.08, xmin=0.1, xmax=0.9,c='m',dashes=(5,3))
plt.title('Angle between X-axis and Gravity_mean', fontsize=15)
plt.xticks(rotation = 40)
plt.show()
```



Observations:

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

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



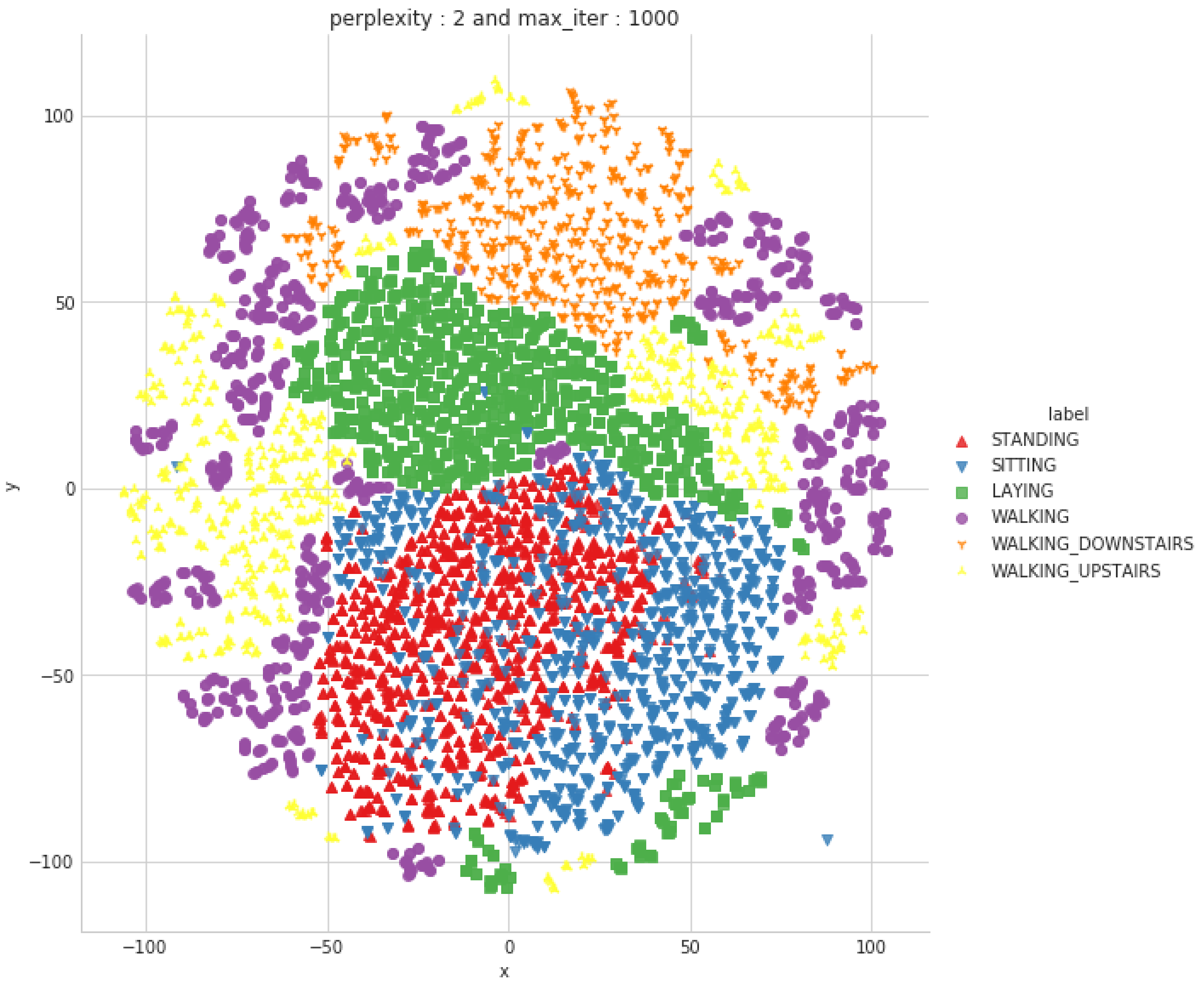
Apply t-sne on the data

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

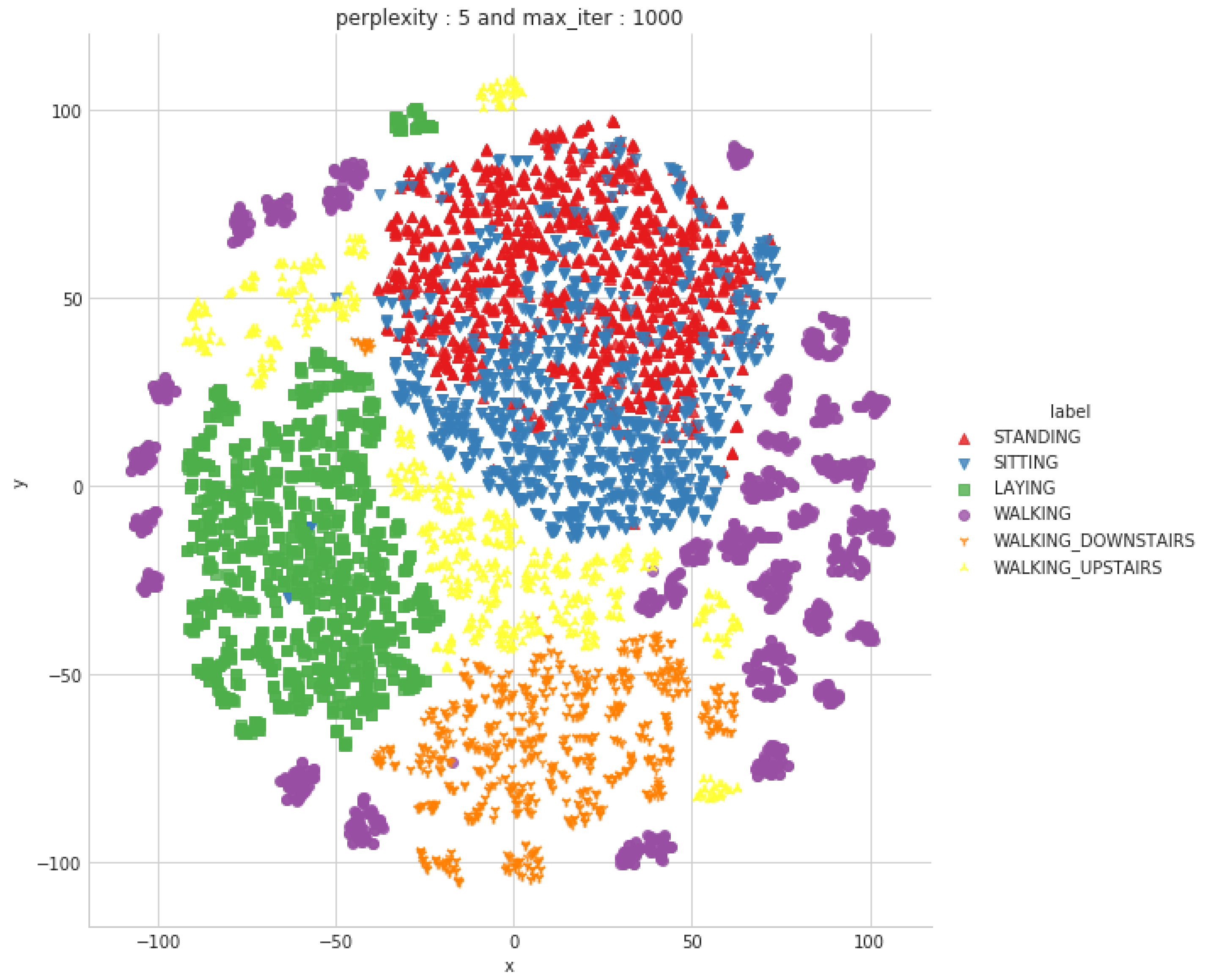
In [0]:
```
# performs t-sne with different perplexity values and their repective plots..

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

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

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

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

In [0]:
```
X_pre_tsne = train.drop(['subject', 'Activity','ActivityName'], axis=1)
y_pre_tsne = train['ActivityName']
perform_tsne(X_data = X_pre_tsne,y_data=y_pre_tsne, perplexities =[2,5,10,20,50])
```

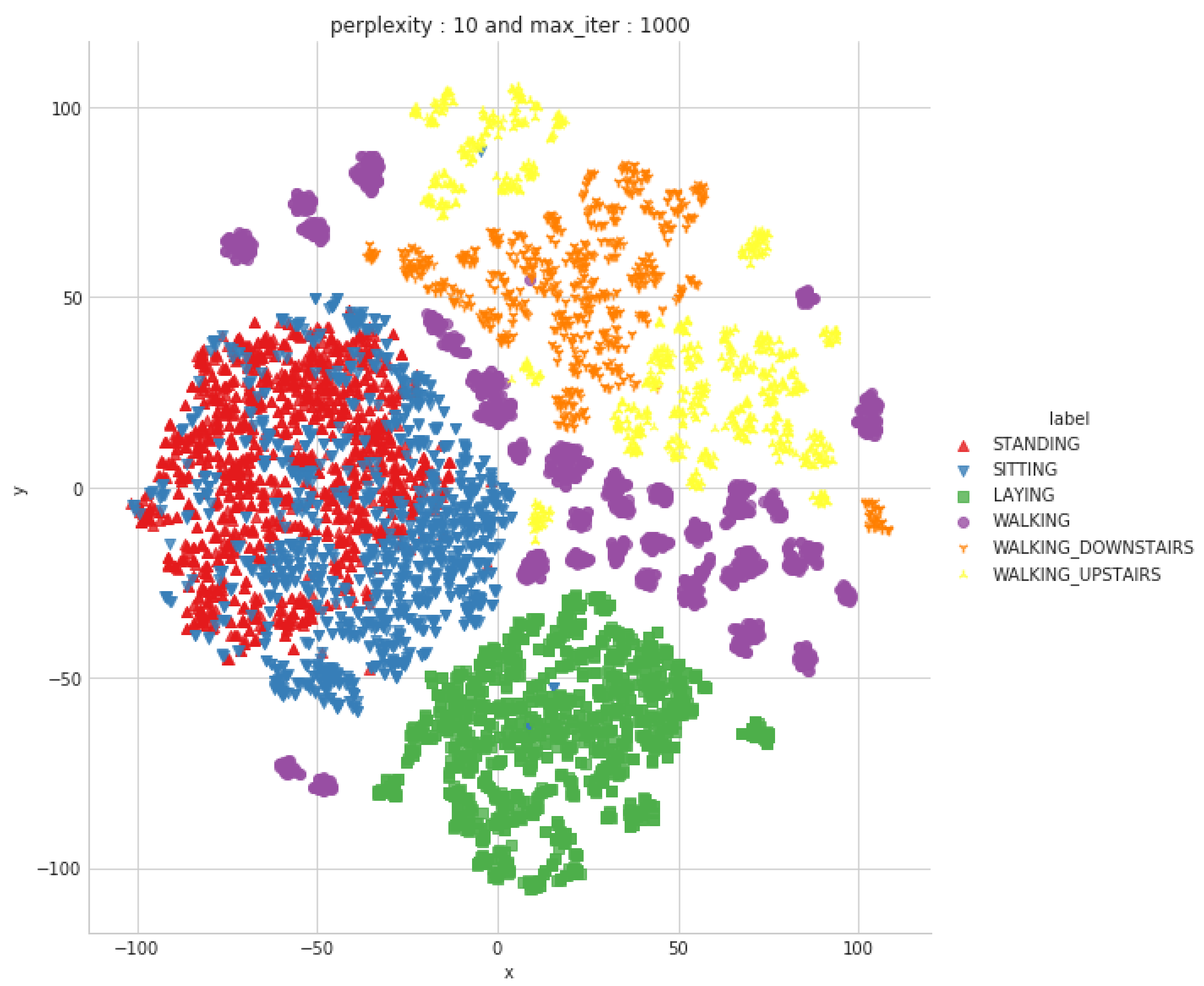
```
performing tsne with perplexity 2 and with 1000 iterations at max
[t-SNE] Computing 7 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.426s...
[t-SNE] Computed neighbors for 7352 samples in 72.001s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 0.635855
[t-SNE] Computed conditional probabilities in 0.071s
[t-SNE] Iteration 50: error = 124.8017578, gradient norm = 0.0253939 (50 iterations in 16.625s)
[t-SNE] Iteration 100: error = 107.2019501, gradient norm = 0.0284782 (50 iterations in 9.735s)
[t-SNE] Iteration 150: error = 100.9872894, gradient norm = 0.0185151 (50 iterations in 5.346s)
[t-SNE] Iteration 200: error = 97.6054382, gradient norm = 0.0142084 (50 iterations in 7.013s)
[t-SNE] Iteration 250: error = 95.3084183, gradient norm = 0.0132592 (50 iterations in 5.703s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 95.308418
[t-SNE] Iteration 300: error = 4.1209540, gradient norm = 0.0015668 (50 iterations in 7.156s)
[t-SNE] Iteration 350: error = 3.2113254, gradient norm = 0.0009953 (50 iterations in 8.022s)
[t-SNE] Iteration 400: error = 2.7819963, gradient norm = 0.0007203 (50 iterations in 9.419s)
[t-SNE] Iteration 450: error = 2.5178111, gradient norm = 0.0005655 (50 iterations in 9.370s)
[t-SNE] Iteration 500: error = 2.3341548, gradient norm = 0.0004804 (50 iterations in 7.681s)
[t-SNE] Iteration 550: error = 2.1961622, gradient norm = 0.0004183 (50 iterations in 7.097s)
[t-SNE] Iteration 600: error = 2.0867445, gradient norm = 0.0003664 (50 iterations in 9.274s)
[t-SNE] Iteration 650: error = 1.9967778, gradient norm = 0.0003279 (50 iterations in 7.697s)
[t-SNE] Iteration 700: error = 1.9210005, gradient norm = 0.0002984 (50 iterations in 8.174s)
[t-SNE] Iteration 750: error = 1.8558111, gradient norm = 0.0002776 (50 iterations in 9.747s)
[t-SNE] Iteration 800: error = 1.7989457, gradient norm = 0.0002569 (50 iterations in 8.687s)
[t-SNE] Iteration 850: error = 1.7490212, gradient norm = 0.0002394 (50 iterations in 8.407s)
[t-SNE] Iteration 900: error = 1.7043383, gradient norm = 0.0002224 (50 iterations in 8.351s)
[t-SNE] Iteration 950: error = 1.6641431, gradient norm = 0.0002098 (50 iterations in 7.841s)
[t-SNE] Iteration 1000: error = 1.6279151, gradient norm = 0.0001989 (50 iterations in 5.623s)
[t-SNE] Error after 1000 iterations: 1.627915
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```




```
Done

performing tsne with perplexity 5 and with 1000 iterations at max
[t-SNE] Computing 16 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.263s...
[t-SNE] Computed neighbors for 7352 samples in 48.983s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 0.961265
[t-SNE] Computed conditional probabilities in 0.122s
[t-SNE] Iteration 50: error = 114.1862640, gradient norm = 0.0184120 (50 iterations in 55.655s)
[t-SNE] Iteration 100: error = 97.6535568, gradient norm = 0.0174309 (50 iterations in 12.580s)
[t-SNE] Iteration 150: error = 93.1900101, gradient norm = 0.0101048 (50 iterations in 9.180s)
[t-SNE] Iteration 200: error = 91.2315445, gradient norm = 0.0074560 (50 iterations in 10.340s)
[t-SNE] Iteration 250: error = 90.0714417, gradient norm = 0.0057667 (50 iterations in 9.458s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 90.071442
[t-SNE] Iteration 300: error = 3.5796804, gradient norm = 0.0014691 (50 iterations in 8.718s)
[t-SNE] Iteration 350: error = 2.8173938, gradient norm = 0.0007508 (50 iterations in 10.180s)
[t-SNE] Iteration 400: error = 2.4344938, gradient norm = 0.0005251 (50 iterations in 10.506s)
[t-SNE] Iteration 450: error = 2.2156141, gradient norm = 0.0004069 (50 iterations in 10.072s)
[t-SNE] Iteration 500: error = 2.0703306, gradient norm = 0.0003340 (50 iterations in 10.511s)
[t-SNE] Iteration 550: error = 1.9646366, gradient norm = 0.0002816 (50 iterations in 9.792s)
[t-SNE] Iteration 600: error = 1.8835558, gradient norm = 0.0002471 (50 iterations in 9.098s)
[t-SNE] Iteration 650: error = 1.8184001, gradient norm = 0.0002184 (50 iterations in 8.656s)
[t-SNE] Iteration 700: error = 1.7647167, gradient norm = 0.0001961 (50 iterations in 9.063s)
[t-SNE] Iteration 750: error = 1.7193680, gradient norm = 0.0001796 (50 iterations in 9.754s)
[t-SNE] Iteration 800: error = 1.6803776, gradient norm = 0.0001655 (50 iterations in 9.540s)
[t-SNE] Iteration 850: error = 1.6465144, gradient norm = 0.0001538 (50 iterations in 9.953s)
[t-SNE] Iteration 900: error = 1.6166563, gradient norm = 0.0001421 (50 iterations in 10.270s)
[t-SNE] Iteration 950: error = 1.5901035, gradient norm = 0.0001335 (50 iterations in 6.609s)
[t-SNE] Iteration 1000: error = 1.5664237, gradient norm = 0.0001257 (50 iterations in 8.553s)
[t-SNE] Error after 1000 iterations: 1.566424
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```

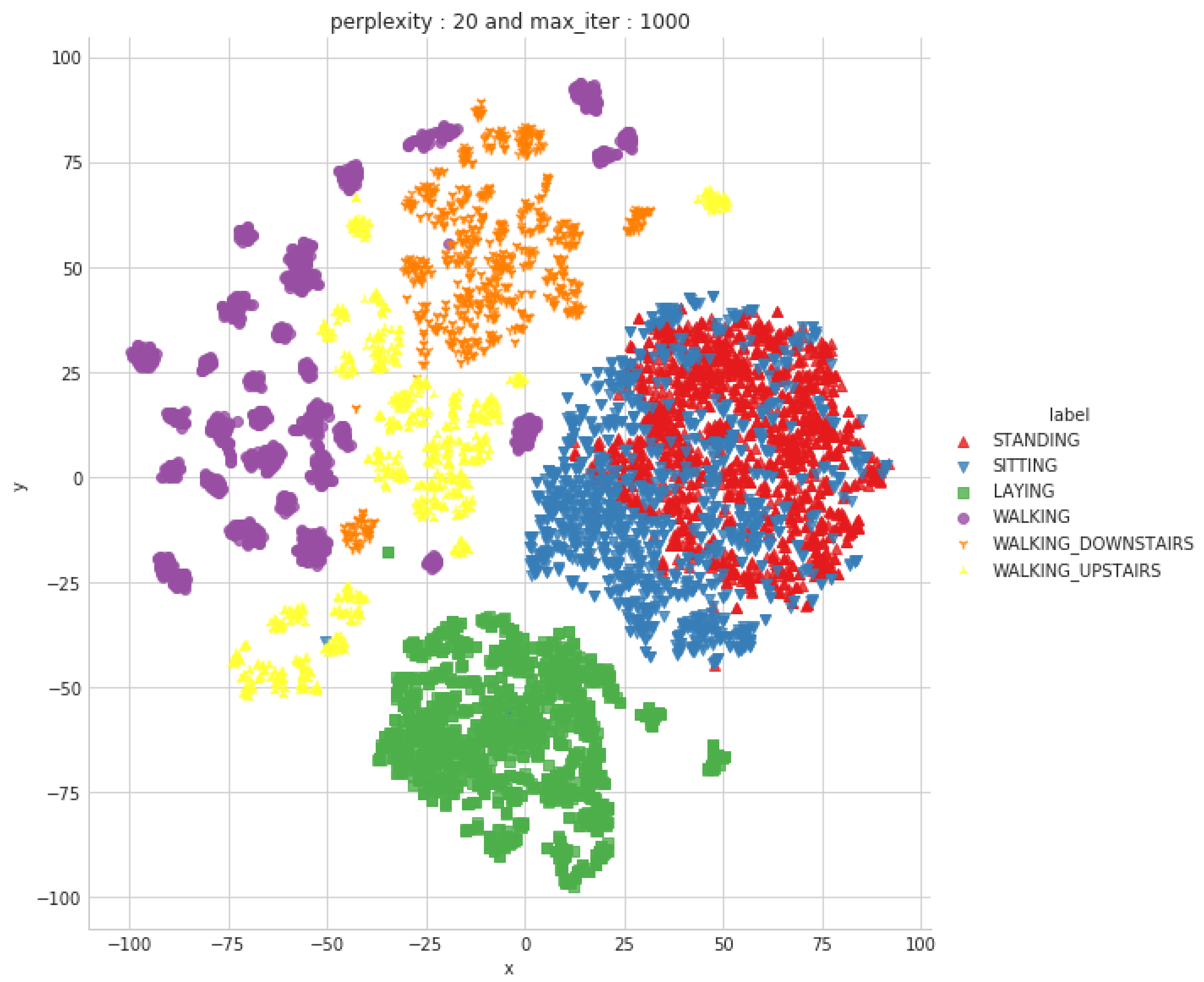



```
Done

performing tsne with perplexity 10 and with 1000 iterations at max
[t-SNE] Computing 31 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.410s...
[t-SNE] Computed neighbors for 7352 samples in 64.801s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.133828
[t-SNE] Computed conditional probabilities in 0.214s
[t-SNE] Iteration 50: error = 106.0169220, gradient norm = 0.0194293 (50 iterations in 24.550s)
[t-SNE] Iteration 100: error = 90.3036194, gradient norm = 0.0097653 (50 iterations in 11.936s)
[t-SNE] Iteration 150: error = 87.3132935, gradient norm = 0.0053059 (50 iterations in 11.246s)
[t-SNE] Iteration 200: error = 86.1169128, gradient norm = 0.0035844 (50 iterations in 11.864s)
[t-SNE] Iteration 250: error = 85.4133606, gradient norm = 0.0029100 (50 iterations in 11.944s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.413361
[t-SNE] Iteration 300: error = 3.1394315, gradient norm = 0.0013976 (50 iterations in 11.742s)
[t-SNE] Iteration 350: error = 2.4929206, gradient norm = 0.0006466 (50 iterations in 11.627s)
[t-SNE] Iteration 400: error = 2.1733041, gradient norm = 0.0004230 (50 iterations in 11.846s)
[t-SNE] Iteration 450: error = 1.9884514, gradient norm = 0.0003124 (50 iterations in 11.405s)
[t-SNE] Iteration 500: error = 1.8702440, gradient norm = 0.0002514 (50 iterations in 11.320s)
[t-SNE] Iteration 550: error = 1.7870129, gradient norm = 0.0002107 (50 iterations in 12.009s)
[t-SNE] Iteration 600: error = 1.7246909, gradient norm = 0.0001824 (50 iterations in 10.632s)
[t-SNE] Iteration 650: error = 1.6758548, gradient norm = 0.0001590 (50 iterations in 11.270s)
[t-SNE] Iteration 700: error = 1.6361949, gradient norm = 0.0001451 (50 iterations in 12.072s)
[t-SNE] Iteration 750: error = 1.6034756, gradient norm = 0.0001305 (50 iterations in 11.607s)
[t-SNE] Iteration 800: error = 1.5761518, gradient norm = 0.0001188 (50 iterations in 9.409s)
[t-SNE] Iteration 850: error = 1.5527289, gradient norm = 0.0001113 (50 iterations in 8.309s)
[t-SNE] Iteration 900: error = 1.5328671, gradient norm = 0.0001021 (50 iterations in 9.433s)
[t-SNE] Iteration 950: error = 1.5152045, gradient norm = 0.0000974 (50 iterations in 11.488s)
[t-SNE] Iteration 1000: error = 1.4999681, gradient norm = 0.0000933 (50 iterations in 10.593s)
[t-SNE] Error after 1000 iterations: 1.499968
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```

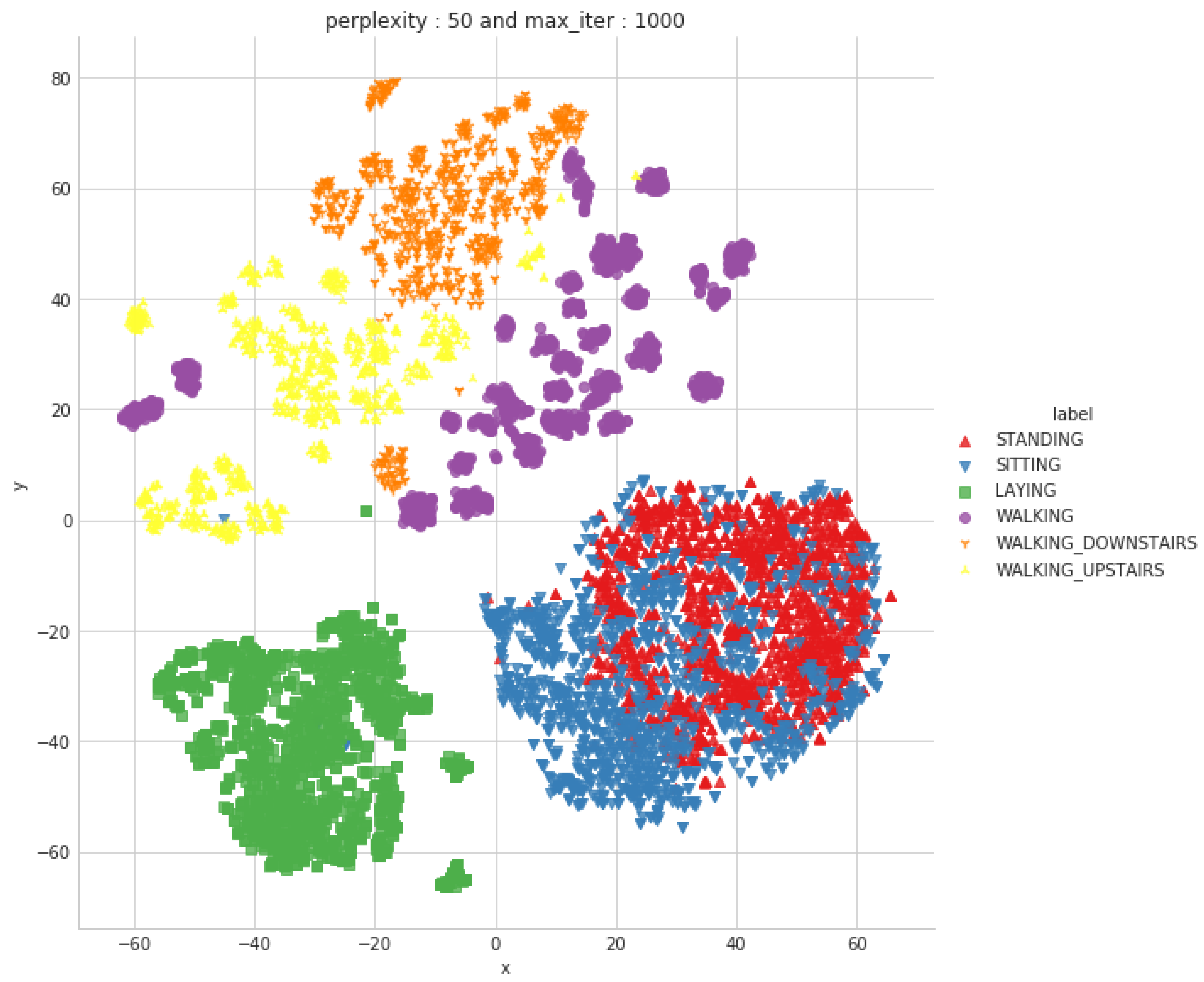



```
Done

performing tsne with perplexity 20 and with 1000 iterations at max
[t-SNE] Computing 61 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.425s...
[t-SNE] Computed neighbors for 7352 samples in 61.792s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.274335
```

```
[t-SNE] Computed conditional probabilities in 0.355s
[t-SNE] Iteration 50: error = 97.5202179, gradient norm = 0.0223863 (50 iterations in 21.168s)
[t-SNE] Iteration 100: error = 83.9500732, gradient norm = 0.0059110 (50 iterations in 17.306s)
[t-SNE] Iteration 150: error = 81.8804779, gradient norm = 0.0035797 (50 iterations in 14.258s)
[t-SNE] Iteration 200: error = 81.1615143, gradient norm = 0.0022536 (50 iterations in 14.130s)
[t-SNE] Iteration 250: error = 80.7704086, gradient norm = 0.0018108 (50 iterations in 15.340s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 80.770409
[t-SNE] Iteration 300: error = 2.6957574, gradient norm = 0.0012993 (50 iterations in 13.605s)
[t-SNE] Iteration 350: error = 2.1637220, gradient norm = 0.0005765 (50 iterations in 13.248s)
[t-SNE] Iteration 400: error = 1.9143614, gradient norm = 0.0003474 (50 iterations in 14.774s)
[t-SNE] Iteration 450: error = 1.7684202, gradient norm = 0.0002458 (50 iterations in 15.502s)
[t-SNE] Iteration 500: error = 1.6744757, gradient norm = 0.0001923 (50 iterations in 14.808s)
[t-SNE] Iteration 550: error = 1.6101606, gradient norm = 0.0001575 (50 iterations in 14.043s)
[t-SNE] Iteration 600: error = 1.5641028, gradient norm = 0.0001344 (50 iterations in 15.769s)
[t-SNE] Iteration 650: error = 1.5291905, gradient norm = 0.0001182 (50 iterations in 15.834s)
[t-SNE] Iteration 700: error = 1.5024391, gradient norm = 0.0001055 (50 iterations in 15.398s)
[t-SNE] Iteration 750: error = 1.4809053, gradient norm = 0.0000965 (50 iterations in 14.594s)
[t-SNE] Iteration 800: error = 1.4631859, gradient norm = 0.0000884 (50 iterations in 15.025s)
[t-SNE] Iteration 850: error = 1.4486470, gradient norm = 0.0000832 (50 iterations in 14.060s)
[t-SNE] Iteration 900: error = 1.4367288, gradient norm = 0.0000804 (50 iterations in 12.389s)
[t-SNE] Iteration 950: error = 1.4270191, gradient norm = 0.0000761 (50 iterations in 10.392s)
[t-SNE] Iteration 1000: error = 1.4189968, gradient norm = 0.0000787 (50 iterations in 12.355s)
[t-SNE] Error after 1000 iterations: 1.418997
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```




```
Done

performing tsne with perplexity 50 and with 1000 iterations at max
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.376s...
[t-SNE] Computed neighbors for 7352 samples in 73.164s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.437672
[t-SNE] Computed conditional probabilities in 0.844s
[t-SNE] Iteration 50: error = 86.1525574, gradient norm = 0.0242986 (50 iterations in 36.249s)
[t-SNE] Iteration 100: error = 75.9874649, gradient norm = 0.0061005 (50 iterations in 30.453s)
[t-SNE] Iteration 150: error = 74.7072296, gradient norm = 0.0024708 (50 iterations in 28.461s)
[t-SNE] Iteration 200: error = 74.2736282, gradient norm = 0.0018644 (50 iterations in 27.735s)
[t-SNE] Iteration 250: error = 74.0722427, gradient norm = 0.0014078 (50 iterations in 26.835s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 74.072243
[t-SNE] Iteration 300: error = 2.1539080, gradient norm = 0.0011796 (50 iterations in 25.445s)
[t-SNE] Iteration 350: error = 1.7567128, gradient norm = 0.0004845 (50 iterations in 21.282s)
[t-SNE] Iteration 400: error = 1.5888531, gradient norm = 0.0002798 (50 iterations in 21.015s)
[t-SNE] Iteration 450: error = 1.4956820, gradient norm = 0.0001894 (50 iterations in 23.332s)
[t-SNE] Iteration 500: error = 1.4359720, gradient norm = 0.0001420 (50 iterations in 23.083s)
[t-SNE] Iteration 550: error = 1.3947564, gradient norm = 0.0001117 (50 iterations in 19.626s)
[t-SNE] Iteration 600: error = 1.3653858, gradient norm = 0.0000949 (50 iterations in 22.752s)
[t-SNE] Iteration 650: error = 1.3441534, gradient norm = 0.0000814 (50 iterations in 23.972s)
[t-SNE] Iteration 700: error = 1.3284039, gradient norm = 0.0000742 (50 iterations in 20.636s)
[t-SNE] Iteration 750: error = 1.3171139, gradient norm = 0.0000700 (50 iterations in 20.407s)
[t-SNE] Iteration 800: error = 1.3085558, gradient norm = 0.0000657 (50 iterations in 24.951s)
[t-SNE] Iteration 850: error = 1.3017821, gradient norm = 0.0000603 (50 iterations in 24.719s)
[t-SNE] Iteration 900: error = 1.2962619, gradient norm = 0.0000586 (50 iterations in 24.500s)
[t-SNE] Iteration 950: error = 1.2914882, gradient norm = 0.0000573 (50 iterations in 24.132s)
[t-SNE] Iteration 1000: error = 1.2874244, gradient norm = 0.0000546 (50 iterations in 22.840s)
[t-SNE] Error after 1000 iterations: 1.287424
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```



Done

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

Obtain the train and test data

In [0]:
```
train = pd.read_csv('UCI_HAR_dataset/csv_files/train.csv')
test = pd.read_csv('UCI_HAR_dataset/csv_files/test.csv')
print(train.shape, test.shape)
```

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

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

Out[3]:

	tBodyAccmeanX	tBodyAccmeanY	tBodyAccmeanZ	tBodyAccstdX	tBodyAccstdY	tBodyAccstdZ	tBodyAccmadX	tBodyAccmadY	tBodyAccmadZ	tBodyAccmaxX	...	angletBodyAccMeangravity	angletBodyAccJerkMeangra
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.112754	
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	0.053477	-
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	-0.118559	

3 rows × 564 columns

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

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

In [0]:
```
print('X_train and y_train : ({},{})'.format(X_train.shape, y_train.shape))
print('X_test  and y_test  : ({},{})'.format(X_test.shape, y_test.shape))
```

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

Let's model with our data

Labels that are useful in plotting confusion matrix

In [0]:
```
labels=['LAYING', 'SITTING','STANDING','WALKING','WALKING_DOWNSTAIRS','WALKING_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['training_time']))


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


    # calculate overall accuracty of the model
    accuracy = metrics.accuracy_score(y_true=y_test, y_pred=y_pred)
    # store accuracy in results
    results['accuracy'] = accuracy
    print('---------------------')
    print('|      Accuracy      |')
    print('---------------------')
    print('\n    {}\n\n'.format(accuracy))


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

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

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

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

    return results
```

Method to print the gridsearch Attributes

In [0]:
```
def print_grid_search_attributes(model):
    # Estimator that gave highest score among all the estimators formed in GridSearch
    print('--------------------------')
    print('|      Best Estimator     |')
    print('--------------------------')
    print('\n\t{}\n'.format(model.best_estimator_))


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


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


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

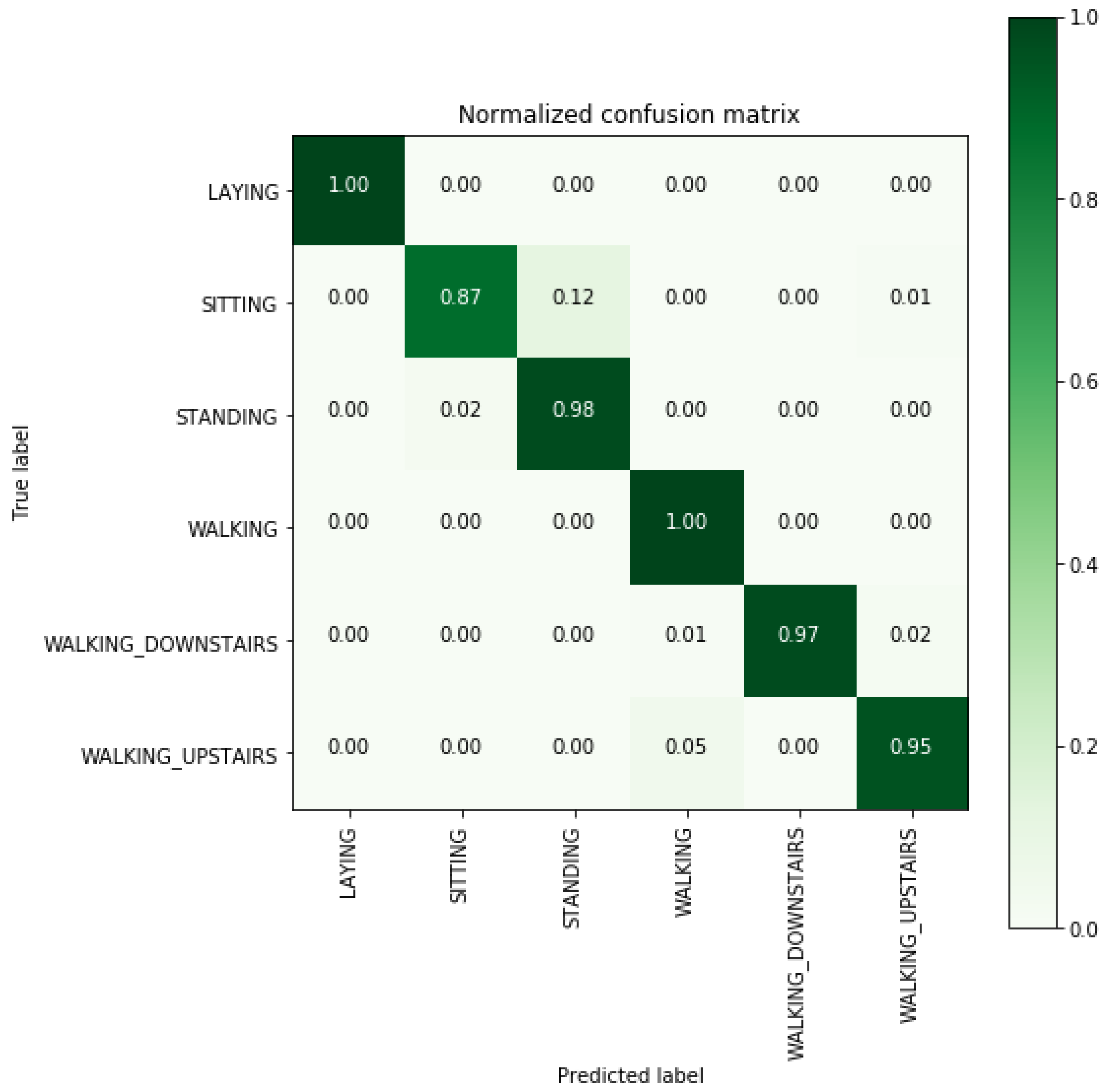
1. Logistic Regression with Grid Search

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

from sklearn.model_selection import GridSearchCV
```

In [0]:

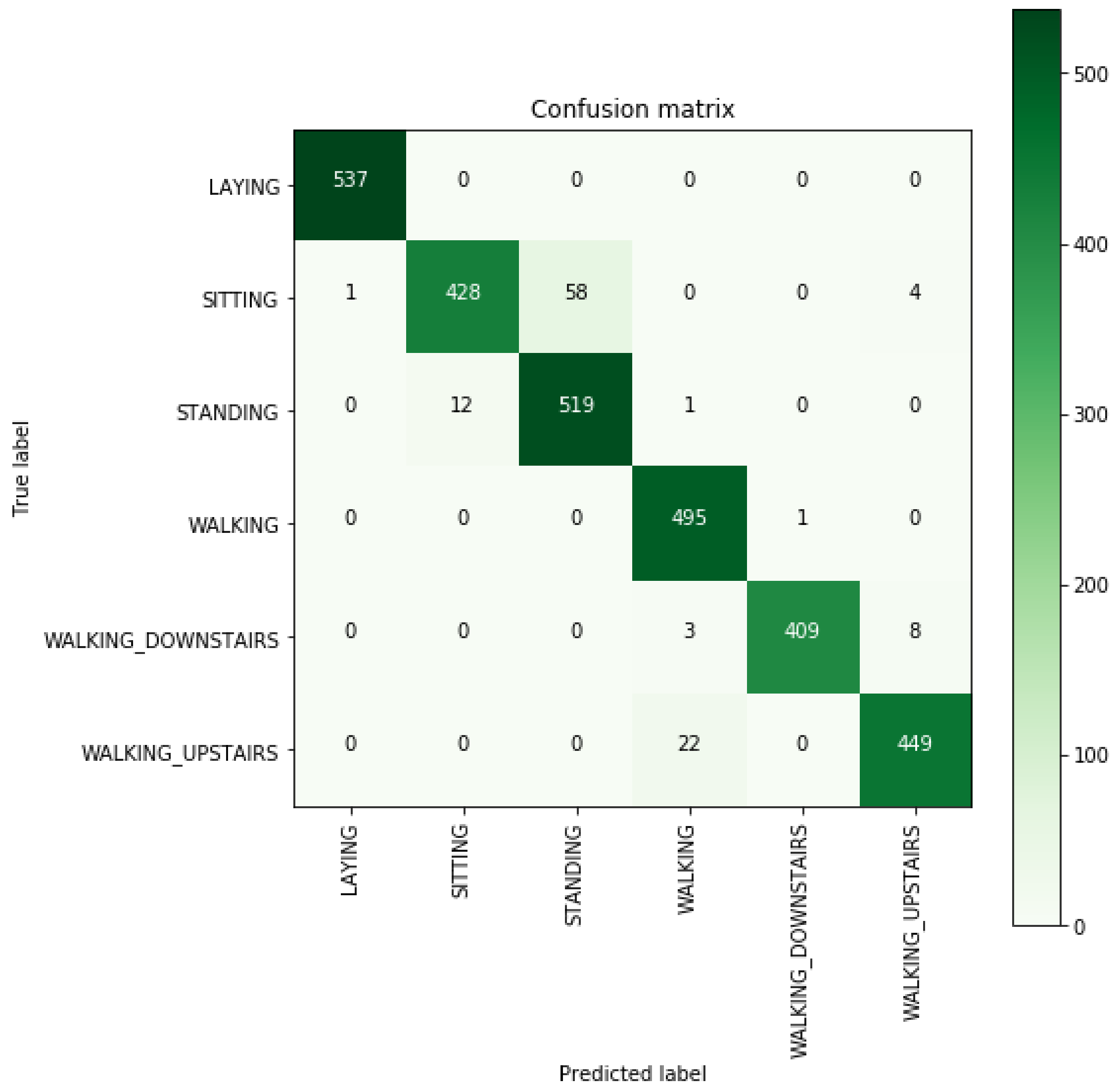
```
# start Grid search
parameters = {'C':[0.01, 0.1, 1, 10, 20, 30], 'penalty':['l2','l1']}
log_reg = linear_model.LogisticRegression()
log_reg_grid = GridSearchCV(log_reg, param_grid=parameters, cv=3, verbose=1, n_jobs=-1)
log_reg_grid_results =  perform_model(log_reg_grid, X_train, y_train, X_test,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

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

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

Done


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


Predicting test data
Done


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


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

    0.9626739056667798


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

 [[537   0   0   0   0   0]
 [  1 428  58   0   0   4]
 [  0  12 519   1   0   0]
 [  0   0   0 495   1   0]
 [  0   0   0   3 409   8]
 [  0   0   0  22   0 449]]
```




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

            LAYING       1.00      1.00      1.00       537
           SITTING       0.97      0.87      0.92       491
          STANDING       0.90      0.98      0.94       532
           WALKING       0.95      1.00      0.97       496
WALKING_DOWNSTAIRS       1.00      0.97      0.99       420
  WALKING_UPSTAIRS       0.97      0.95      0.96       471

       avg / total       0.96      0.96      0.96      2947
```

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



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

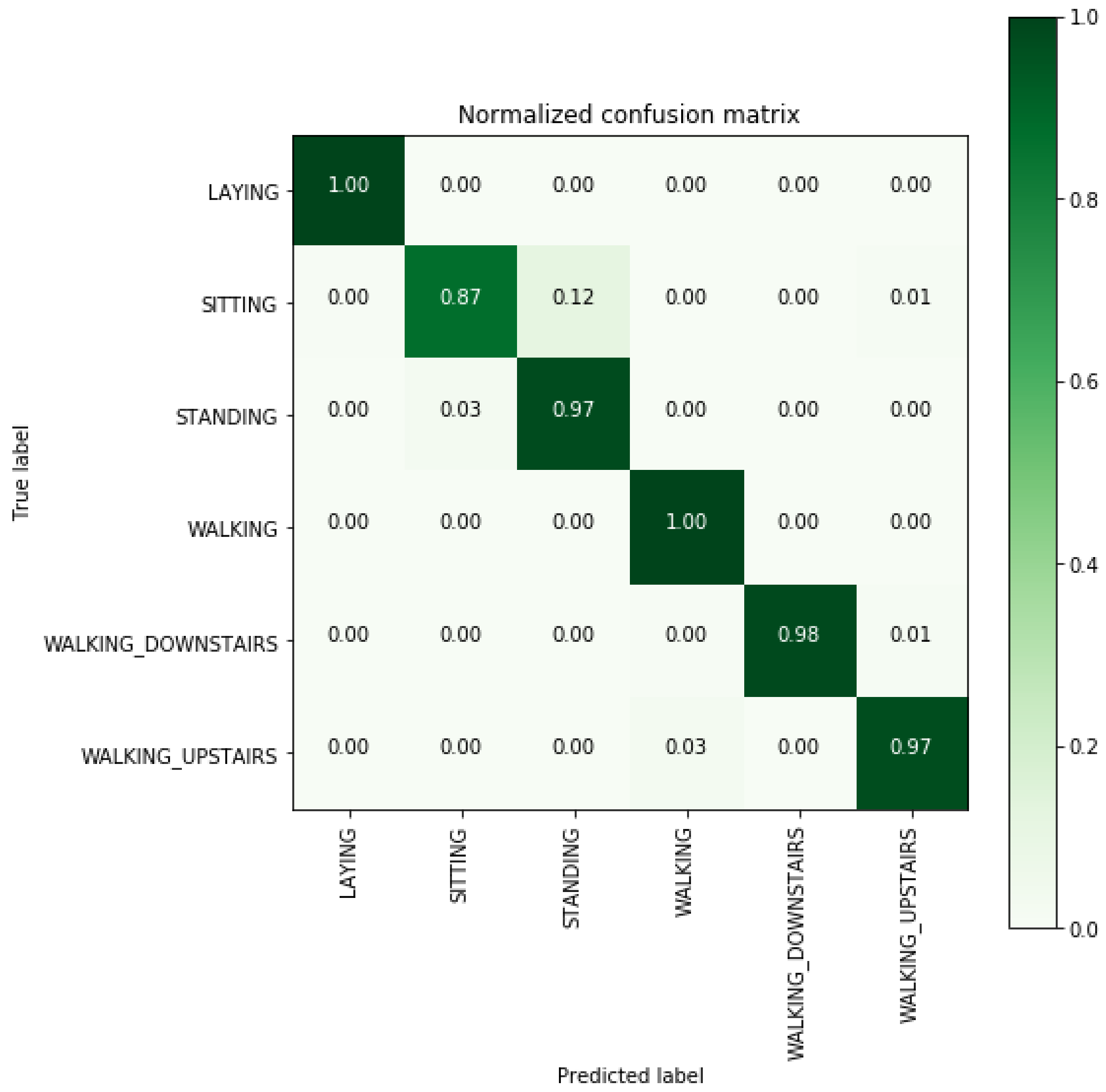
```
--------------------------
|      Best Estimator     |
--------------------------

        LogisticRegression(C=30, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)

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

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

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.9461371055495104
```

2. Linear SVC with GridSearch

In [0]:
```
from sklearn.svm import LinearSVC
```

In [0]:
```
parameters = {'C':[0.125, 0.5, 1, 2, 8, 16]}
lr_svc = LinearSVC(tol=0.00005)
lr_svc_grid = GridSearchCV(lr_svc, param_grid=parameters, n_jobs=-1, verbose=1)
lr_svc_grid_results = perform_model(lr_svc_grid, X_train, y_train, X_test, y_test, class_labels=labels)
```

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

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

Done


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


Predicting test data
Done


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


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

    0.9660671869697998


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

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




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

            LAYING       1.00      1.00      1.00       537
           SITTING       0.97      0.87      0.92       491
          STANDING       0.90      0.97      0.94       532
           WALKING       0.97      1.00      0.99       496
WALKING_DOWNSTAIRS       1.00      0.98      0.99       420
  WALKING_UPSTAIRS       0.98      0.97      0.97       471

       avg / total       0.97      0.97      0.97      2947
```

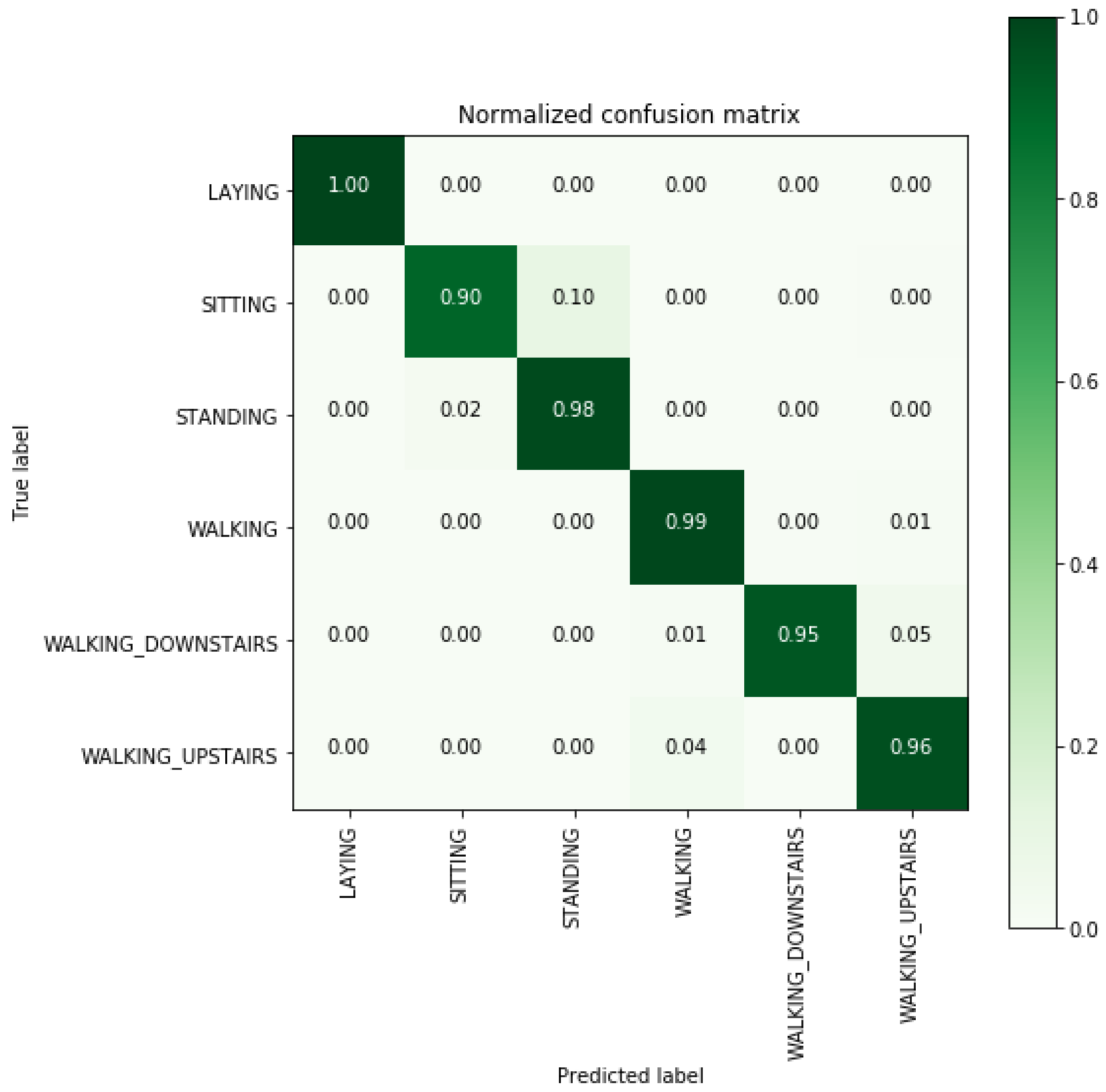
In [0]:
```
print_grid_search_attributes(lr_svc_grid_results['model'])
```

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

        LinearSVC(C=8, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=5e-05,
     verbose=0)

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

        {'C': 8}

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.9465451577801959
```

3. Kernel SVM with GridSearch

In [0]:
```
from sklearn.svm import SVC
parameters = {'C':[2,8,16],\
              'gamma': [ 0.0078125, 0.125, 2]}
rbf_svm = SVC(kernel='rbf')
rbf_svm_grid = GridSearchCV(rbf_svm,param_grid=parameters, n_jobs=-1)
rbf_svm_grid_results = perform_model(rbf_svm_grid, X_train, y_train, X_test, y_test, class_labels=labels)
```

```
training the model..
Done


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


Predicting test data
Done


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


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

    0.9626739056667798


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

 [[537   0   0   0   0   0]
 [  0 441  48   0   0   2]
 [  0  12 520   0   0   0]
 [  0   0   0 489   2   5]
 [  0   0   0   4 397  19]
 [  0   0   0  17   1 453]]
```




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

       avg / total       0.96      0.96      0.96      2947
```

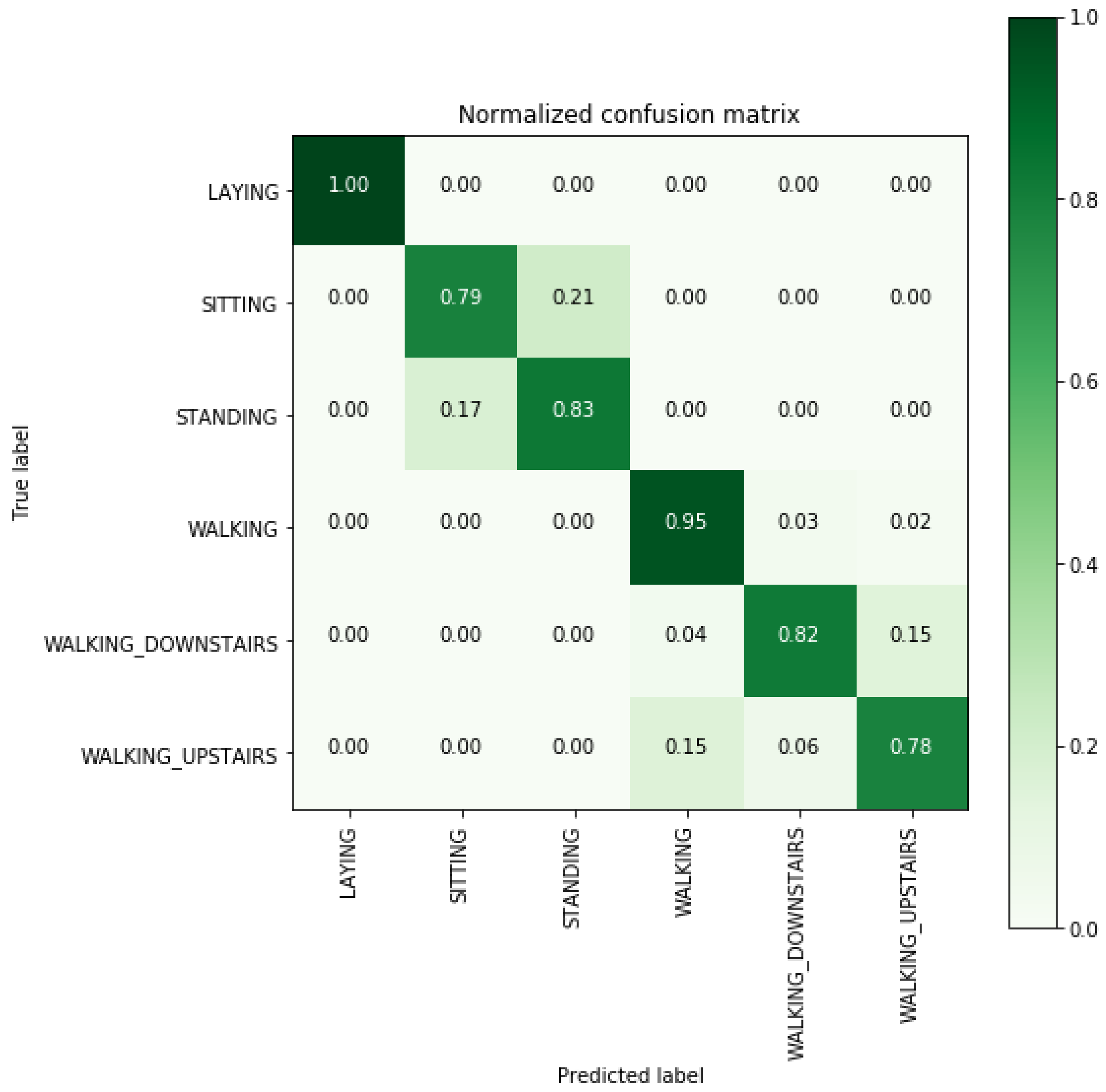
In [0]:
```
print_grid_search_attributes(rbf_svm_grid_results['model'])
```

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

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

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

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

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.9440968443960827
```

4. Decision Trees with GridSearchCV

In [0]:
```
from sklearn.tree import DecisionTreeClassifier
parameters = {'max_depth':np.arange(3,10,2)}
dt = DecisionTreeClassifier()
dt_grid = GridSearchCV(dt,param_grid=parameters, n_jobs=-1)
dt_grid_results = perform_model(dt_grid, X_train, y_train, X_test, y_test, class_labels=labels)
print_grid_search_attributes(dt_grid_results['model'])
```

```
training the model..
Done


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


Predicting test data
Done


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


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

    0.8642687478791992


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

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

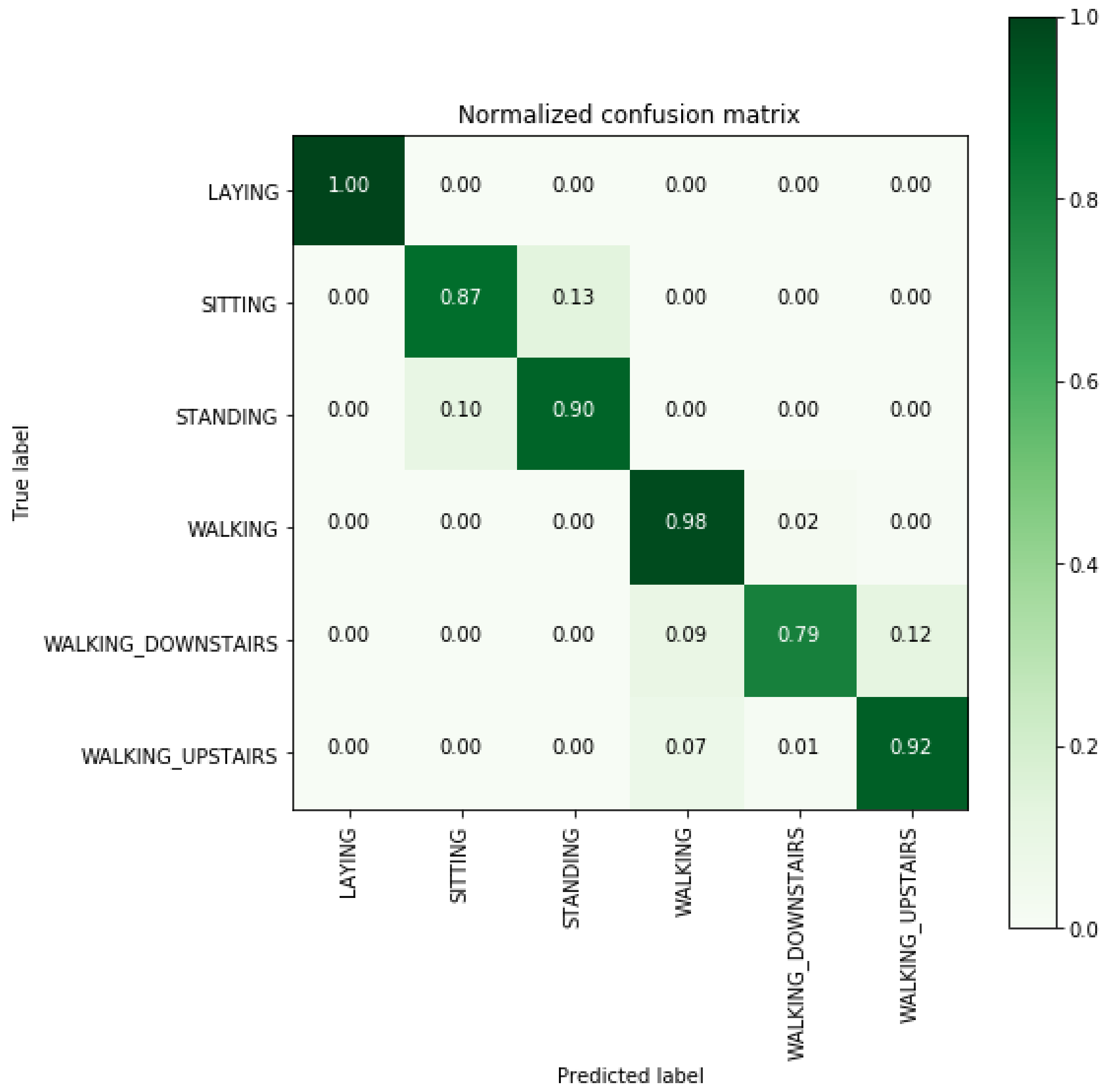



```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       537
           SITTING       0.81      0.79      0.80       491
          STANDING       0.81      0.83      0.82       532
           WALKING       0.84      0.95      0.89       496
WALKING_DOWNSTAIRS       0.88      0.82      0.85       420
  WALKING_UPSTAIRS       0.84      0.78      0.81       471

       avg / total       0.86      0.86      0.86      2947

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

        DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=7,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')

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

        {'max_depth': 7}

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.8369151251360174
```

5. Random Forest Classifier with GridSearch

In [0]:
```
from sklearn.ensemble import RandomForestClassifier
params = {'n_estimators': np.arange(10,201,20), 'max_depth':np.arange(3,15,2)}
rfc = RandomForestClassifier()
rfc_grid = GridSearchCV(rfc, param_grid=params, n_jobs=-1)
rfc_grid_results = perform_model(rfc_grid, X_train, y_train, X_test, y_test, class_labels=labels)
print_grid_search_attributes(rfc_grid_results['model'])
```

```
training the model..
Done


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


Predicting test data
Done


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


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

    0.9131319986426875


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

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

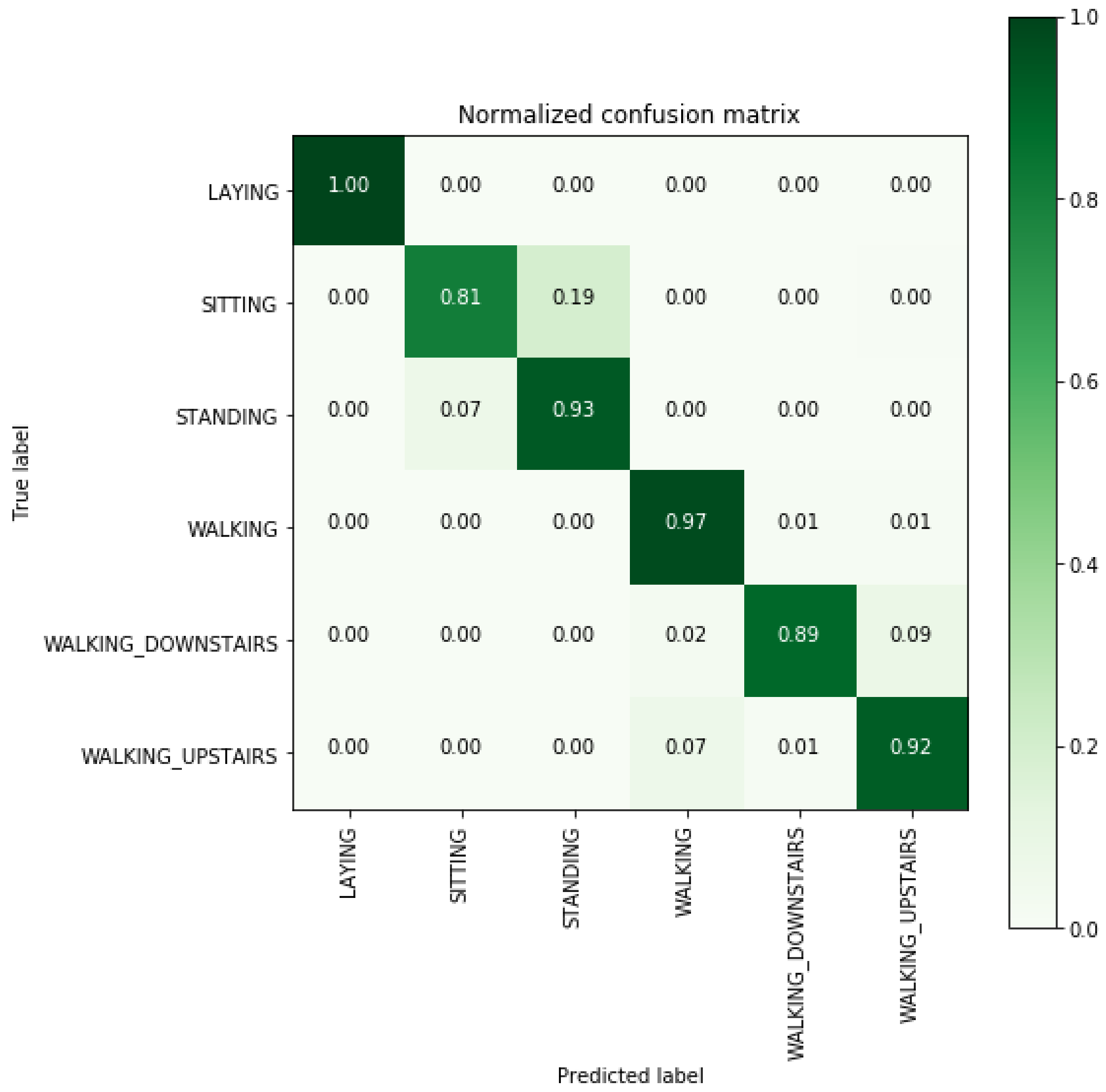



```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

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

       avg / total       0.92      0.91      0.91      2947

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

        RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=7, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=70, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)

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

        {'max_depth': 7, 'n_estimators': 70}

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.9141730141458106
```

6. Gradient Boosted Decision Trees With GridSearch

In [0]:
```python
from sklearn.ensemble import GradientBoostingClassifier
param_grid = {'max_depth': np.arange(5,8,1), \
             'n_estimators':np.arange(130,170,10)}
gbdt = GradientBoostingClassifier()
gbdt_grid = GridSearchCV(gbdt, param_grid=param_grid, n_jobs=-1)
gbdt_grid_results = perform_model(gbdt_grid, X_train, y_train, X_test, y_test, class_labels=labels)
print_grid_search_attributes(gbdt_grid_results['model'])
```

```
training the model..
Done


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


Predicting test data
Done


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


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

    0.9222938581608415


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

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




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

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

       avg / total       0.92      0.92      0.92      2947

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

        GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=5,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=140,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False)

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

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

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.904379760609358
```

7. Comparing all models

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

print('Linear SVC          : {:.04}%       {:.04}% '.format(lr_svc_grid_results['accuracy'] * 100,\
                                                        100-(lr_svc_grid_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_results['accuracy'] * 100,\
                                                        100-(dt_grid_results['accuracy'] * 100)))

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

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

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

Conclusion :

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

Deep Learning

In [0]:
```
import seaborn as sns
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn import metrics
import numpy as np
import pandas as pd
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
from pandas_ml import ConfusionMatrix
import warnings
warnings.simplefilter("ignore")
```

In [0]:
```
all_signals_list = ["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]:
```
def data_read(filename):
    return pd.read_csv(filename, delim_whitespace = True, header = None)
```

In [0]:
```
def signals_load(trainOrTest):
    complete_data = []
    for signal in all_signals_list:
        complete_data.append(data_read("Human_Activity_Recognition/HAR/UCI_HAR_Dataset/"+ trainOrTest +"/Inertial Signals/"+ signal + trainOrTest +".txt").as_matrix())
    return np.transpose(complete_data, (1, 2, 0))
```

In [0]:
```
def load_y(subset):
    filename = "Human_Activity_Recognition/HAR/UCI_HAR_Dataset/"+subset+"/y_"+subset+".txt"
    y = data_read(filename)
    return pd.get_dummies(y[0]).as_matrix()
```

In [0]:
```
def load_full_data():
    x_train = signals_load("train")
    y_train = load_y("train")
    x_test = signals_load("test")
    y_test = load_y("test")
    return x_train, y_train, x_test, y_test
```

In [0]:
```
x_train, y_train, x_test, y_test = load_full_data()
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

```
(7352, 128, 9) (7352, 6) (2947, 128, 9) (2947, 6)
```

In [0]:
```
np.save("x_train", x_train) #This will be called upon when we hyperparameter tune using hyperas
np.save("y_train", y_train)
np.save("x_test", x_test)
np.save("y_test", y_test)
```

Single LSTM layer

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

In [0]:
```
timesteps = len(x_train[0])
input_dim = len(x_train[0][0])
n_classes = _count_classes(y_train)
n_hidden = 32

print(timesteps)
print(input_dim)
print(len(x_train))
```

```
128
9
7352
```

In [0]:
```
# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(n_hidden, input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model.add(Dropout(0.5))
# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='sigmoid'))
model.summary()
```

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

In [0]:
```
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

In [0]:
```
# Training the model
model.fit(x_train,
          y_train,
          batch_size=batch_size,
          validation_data=(x_test, y_test),
          epochs=epochs)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 92s 13ms/step - loss: 1.3018 - acc: 0.4395 - val_loss: 1.1254 - val_acc: 0.4662
Epoch 2/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.9666 - acc: 0.5880 - val_loss: 0.9491 - val_acc: 0.5714
Epoch 3/30
7352/7352 [==============================] - 97s 13ms/step - loss: 0.7812 - acc: 0.6408 - val_loss: 0.8286 - val_acc: 0.5850
Epoch 4/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.6941 - acc: 0.6574 - val_loss: 0.7297 - val_acc: 0.6128
Epoch 5/30
7352/7352 [==============================] - 92s 13ms/step - loss: 0.6336 - acc: 0.6912 - val_loss: 0.7359 - val_acc: 0.6787
Epoch 6/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.5859 - acc: 0.7134 - val_loss: 0.7015 - val_acc: 0.6939
Epoch 7/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.5692 - acc: 0.7477 - val_loss: 0.5995 - val_acc: 0.7387
Epoch 8/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.4899 - acc: 0.7809 - val_loss: 0.5762 - val_acc: 0.7387
Epoch 9/30
7352/7352 [==============================] - 90s 12ms/step - loss: 0.4482 - acc: 0.7886 - val_loss: 0.7413 - val_acc: 0.7126
Epoch 10/30
7352/7352 [==============================] - 90s 12ms/step - loss: 0.4132 - acc: 0.8077 - val_loss: 0.5048 - val_acc: 0.7513
Epoch 11/30
7352/7352 [==============================] - 89s 12ms/step - loss: 0.3985 - acc: 0.8274 - val_loss: 0.5234 - val_acc: 0.7452
Epoch 12/30
7352/7352 [==============================] - 91s 12ms/step - loss: 0.3378 - acc: 0.8638 - val_loss: 0.4114 - val_acc: 0.8833
Epoch 13/30
7352/7352 [==============================] - 91s 12ms/step - loss: 0.2947 - acc: 0.9051 - val_loss: 0.4386 - val_acc: 0.8731
Epoch 14/30
7352/7352 [==============================] - 90s 12ms/step - loss: 0.2448 - acc: 0.9291 - val_loss: 0.3768 - val_acc: 0.8921
Epoch 15/30
7352/7352 [==============================] - 91s 12ms/step - loss: 0.2157 - acc: 0.9331 - val_loss: 0.4441 - val_acc: 0.8931
Epoch 16/30
7352/7352 [==============================] - 90s 12ms/step - loss: 0.2053 - acc: 0.9366 - val_loss: 0.4162 - val_acc: 0.8968
Epoch 17/30
7352/7352 [==============================] - 89s 12ms/step - loss: 0.2028 - acc: 0.9404 - val_loss: 0.4538 - val_acc: 0.8962
Epoch 18/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.1911 - acc: 0.9419 - val_loss: 0.3964 - val_acc: 0.8999
Epoch 19/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1912 - acc: 0.9407 - val_loss: 0.3165 - val_acc: 0.9030
Epoch 20/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1732 - acc: 0.9446 - val_loss: 0.4546 - val_acc: 0.8904
Epoch 21/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1782 - acc: 0.9444 - val_loss: 0.3346 - val_acc: 0.9063
Epoch 22/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1812 - acc: 0.9418 - val_loss: 0.8164 - val_acc: 0.8582
Epoch 23/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1824 - acc: 0.9426 - val_loss: 0.4240 - val_acc: 0.9036
Epoch 24/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1726 - acc: 0.9429 - val_loss: 0.4067 - val_acc: 0.9148
Epoch 25/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1737 - acc: 0.9411 - val_loss: 0.3396 - val_acc: 0.9074
Epoch 26/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1650 - acc: 0.9461 - val_loss: 0.3806 - val_acc: 0.9019
Epoch 27/30
7352/7352 [==============================] - 89s 12ms/step - loss: 0.1925 - acc: 0.9415 - val_loss: 0.6464 - val_acc: 0.8850
Epoch 28/30
7352/7352 [==============================] - 91s 12ms/step - loss: 0.1965 - acc: 0.9425 - val_loss: 0.3363 - val_acc: 0.9203
Epoch 29/30
7352/7352 [==============================] - 92s 12ms/step - loss: 0.1889 - acc: 0.9431 - val_loss: 0.3737 - val_acc: 0.9158
Epoch 30/30
7352/7352 [==============================] - 95s 13ms/step - loss: 0.1945 - acc: 0.9414 - val_loss: 0.3088 - val_acc: 0.9097
```

Out[23]: <keras.callbacks.History at 0x29b5ee36a20>

In [0]:
```
# Confusion Matrix
print(confusion_matrix(Y_test, model.predict(X_test)))
```

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

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

In [0]:
```
score = model.evaluate(X_test, Y_test)
```

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

In [0]:
```
score
```

Out[28]:
```
[0.3087582236972612, 0.9097387173396675]
```

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

2 LSTM layers

Refer: https://stackoverflow.com/questions/43533610/how-to-use-hyperopt-for-hyperparameter-optimization-of-keras-deep-learning-netwo (https://stackoverflow.com/questions/43533610/how-to-use-hyperopt-for-hyperparameter-optimization-of-keras-deep-learning-netwo)

https://github.com/maxpumperla/hyperas (https://github.com/maxpumperla/hyperas)

https://towardsdatascience.com/keras-hyperparameter-tuning-in-google-colab-using-hyperas-624fa4bbf673 (https://towardsdatascience.com/keras-hyperparameter-tuning-in-google-colab-using-hyperas-624fa4bbf673)

In [0]:
```
def plot_confusion_matrix(y_test, y_pred):
    #Plot confusion matrix

    #We are using 2 types of confusion matrix here. SKLearn confusion matrix and pandas_ml confusion matrix.
    #SKLearn confusion matrix is used to plot it diagramatically whereas pandas_ml confusion matrix is used just for intresting stats like TPR, TNR etc..

    y_true = np.array(y_test) #Converting y_test and y_pred to array for input into pandas_ml Confusion matrix
    y_pred = np.array(y_pred)
    labels = ['Negative','Positive']
    print(confusion_matrix(y_test, y_pred)) #This prints TP, TN, FP, FN numerically before plotting it diagramatically.
    cm = ConfusionMatrix(y_true,y_pred) #This the confusion matrix of pandas_ml which provides interesting stats.
    confusion_matrix_plot = confusion_matrix(y_test,y_pred) #We are plotting confusion matrix of sklearn
    heatmap = sns.heatmap(confusion_matrix_plot, annot=True,cmap='Blues', fmt='g',xticklabels=["WALKING", "WALKING_UPSTAIRS", "WALKING_DOWNSTAIRS", "SITTING", "STANDING", "LYING"],yt
    plt.title('Confusion matrix of the classifier')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()
    print("*"*50)
    print("The True Positive Rate observed is:\n",cm.TPR) #This prints the True Positive Rate of the confusion matrix (using pandas_ml confusion matrix).
    print("The True Negative Rate observed is:\n",cm.TNR)
    print("The False Positive Rate observed is:\n",cm.FPR)
    print("The False Negative Rate observed is:\n",cm.FNR)
    print("*"*50)
    print("The stats observed for confusion matrix are:")
    cm.print_stats()#Prints all the stats of the confusion matrix plotted (using pandas_ml confusion matrix).
```

Hyperas parameter tuning 1

In [0]:
```
def data():
    x_train = np.load("x_train.npy")
    y_train = np.load("y_train.npy")
    x_test = np.load("x_test.npy")
    y_test = np.load("y_test.npy")
    return x_train, y_train, x_test, y_test
```

In [0]:
```
def lstm_model(x_train, y_train, x_test, y_test):

    epochs = 10
    batch_size = 64
    timesteps = x_train.shape[1]
    input_dim = len(x_train[0][0])
    n_classes = 6

    model = Sequential()

    model.add(LSTM({{choice([32, 64])}}, return_sequences = True, input_shape = (timesteps, input_dim)))
    model.add(Dropout({{uniform(0, 1)}}))

    model.add(LSTM({{choice([16, 32])}}))
    model.add(Dropout({{uniform(0, 1)}}))

    model.add(Dense(n_classes, activation='sigmoid'))

    print(model.summary())

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

    result = model.fit(x_train, y_train, batch_size = batch_size, epochs=epochs, verbose=1, validation_split=0.01)

    scores = model.evaluate(x_test, y_test, verbose=0)

    print('Best test accuracy obtained is',scores[1])

    return {'loss': -scores[1], 'status': STATUS_OK, 'model': model}
```

In [0]:
```
best_run, best_model = optim.minimize(model=lstm_model, data=data, algo=tpe.suggest, max_evals=4, trials=Trials(), notebook_name = "Human Activity Detection")
x_train, y_train, x_test, y_test = data()

score = best_model.evaluate(x_test, y_test)
```

```
>>> Imports:
#coding=utf-8

try:
    from google.colab import auth
except:
    pass

try:
    from oauth2client.client import GoogleCredentials
except:
    pass

try:
    import getpass
except:
    pass

try:
    import numpy as np
except:
    pass

try:
    import pandas as pd
except:
    pass

try:
    from keras.models import Sequential
except:
    pass

try:
    from keras.layers import LSTM
except:
    pass

try:
    from keras.layers.core import Dense, Dropout
except:
    pass

try:
    from hyperopt import Trials, STATUS_OK, tpe
except:
    pass

try:
    from hyperas import optim
except:
    pass

try:
    from hyperas.distributions import choice, uniform
except:
    pass

try:
    import warnings
except:
    pass

try:
    from pydrive.auth import GoogleAuth
except:
    pass

try:
    from pydrive.drive import GoogleDrive
except:
    pass

try:
    from google.colab import auth
except:
    pass

try:
    from oauth2client.client import GoogleCredentials
except:
    pass

>>> Hyperas search space:

def get_space():
    return {
        'LSTM': hp.choice('LSTM', [32, 64]),
        'Dropout': hp.uniform('Dropout', 0, 1),
        'LSTM_1': hp.choice('LSTM_1', [16, 32]),
        'Dropout_1': hp.uniform('Dropout_1', 0, 1),
    }

>>> Data
  1: 
  2: x_train = np.load("x_train.npy")
  3: y_train = np.load("y_train.npy")
  4: x_test = np.load("x_test.npy")
  5: y_test = np.load("y_test.npy")
  6: 
  7: 
  8: 
>>> Resulting replaced keras model:

  1: def keras_fmin_fnct(space):
  2: 
  3: 
  4:     epochs = 10
  5:     batch_size = 64
  6:     timesteps = x_train.shape[1]
  7:     input_dim = len(x_train[0][0])
  8:     n_classes = 6
  9: 
 10:     model = Sequential()
 11: 
 12:     model.add(LSTM(space['LSTM'], return_sequences = True, input_shape = (timesteps, input_dim)))
 13:     model.add(Dropout(space['Dropout']))
 14: 
 15:     model.add(LSTM(space['LSTM_1']))
 16:     model.add(Dropout(space['Dropout_1']))
 17: 
```

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

    print(model.summary())

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

    result = model.fit(x_train, y_train, batch_size = batch_size, epochs=epochs, verbose=1, validation_split=0.01)

    scores = model.evaluate(x_test, y_test, verbose=0)

    print('Best test accuracy obtained is',scores[1])

    return {'loss': -scores[1], 'status': STATUS_OK, 'model': model}

```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_1 (LSTM)                (None, 128, 64)           18944     
_________________________________________________________________
dropout_1 (Dropout)          (None, 128, 64)           0         
_________________________________________________________________
lstm_2 (LSTM)                (None, 16)                5184      
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 102       
=================================================================
Total params: 24,230
Trainable params: 24,230
Non-trainable params: 0
_________________________________________________________________
None
Train on 7278 samples, validate on 74 samples
Epoch 1/10
7278/7278 [==============================] - 25s 3ms/step - loss: 1.5589 - acc: 0.3399 - val_loss: 1.4988 - val_acc: 0.1892
Epoch 2/10
7278/7278 [==============================] - 24s 3ms/step - loss: 1.3612 - acc: 0.4085 - val_loss: 1.3892 - val_acc: 0.3514
Epoch 3/10
7278/7278 [==============================] - 24s 3ms/step - loss: 1.2205 - acc: 0.4729 - val_loss: 1.2866 - val_acc: 0.3784
Epoch 4/10
7278/7278 [==============================] - 24s 3ms/step - loss: 1.1239 - acc: 0.5216 - val_loss: 1.1969 - val_acc: 0.4054
Epoch 5/10
7278/7278 [==============================] - 24s 3ms/step - loss: 1.0402 - acc: 0.5650 - val_loss: 1.1510 - val_acc: 0.4459
Epoch 6/10
7278/7278 [==============================] - 24s 3ms/step - loss: 1.0101 - acc: 0.5894 - val_loss: 1.1339 - val_acc: 0.4324
Epoch 7/10
7278/7278 [==============================] - 24s 3ms/step - loss: 0.9355 - acc: 0.6145 - val_loss: 1.1175 - val_acc: 0.4865
Epoch 8/10
7278/7278 [==============================] - 24s 3ms/step - loss: 0.8975 - acc: 0.6256 - val_loss: 1.0989 - val_acc: 0.3919
Epoch 9/10
7278/7278 [==============================] - 24s 3ms/step - loss: 0.8573 - acc: 0.6358 - val_loss: 1.0976 - val_acc: 0.4324
Epoch 10/10
7278/7278 [==============================] - 24s 3ms/step - loss: 0.8450 - acc: 0.6550 - val_loss: 1.0423 - val_acc: 0.5676
Best test accuracy obtained is 0.6728876823888701
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_3 (LSTM)                (None, 128, 32)           5376      
_________________________________________________________________
dropout_3 (Dropout)          (None, 128, 32)           0         
_________________________________________________________________
lstm_4 (LSTM)                (None, 16)                3136      
_________________________________________________________________
dropout_4 (Dropout)          (None, 16)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 102       
=================================================================
Total params: 8,614
Trainable params: 8,614
Non-trainable params: 0
_________________________________________________________________
None
Train on 7278 samples, validate on 74 samples
Epoch 1/10
7278/7278 [==============================] - 20s 3ms/step - loss: 1.7266 - acc: 0.2531 - val_loss: 1.7397 - val_acc: 0.0135
Epoch 2/10
7278/7278 [==============================] - 18s 3ms/step - loss: 1.6438 - acc: 0.3017 - val_loss: 1.6612 - val_acc: 0.2027
Epoch 3/10
7278/7278 [==============================] - 19s 3ms/step - loss: 1.5582 - acc: 0.3381 - val_loss: 1.5510 - val_acc: 0.4054
Epoch 4/10
7278/7278 [==============================] - 18s 3ms/step - loss: 1.4724 - acc: 0.3516 - val_loss: 1.4504 - val_acc: 0.3378
Epoch 5/10
7278/7278 [==============================] - 19s 3ms/step - loss: 1.4024 - acc: 0.3714 - val_loss: 1.3689 - val_acc: 0.4324
Epoch 6/10
7278/7278 [==============================] - 18s 3ms/step - loss: 1.3779 - acc: 0.3693 - val_loss: 1.3131 - val_acc: 0.4324
Epoch 7/10
7278/7278 [==============================] - 18s 3ms/step - loss: 1.3468 - acc: 0.3674 - val_loss: 1.2716 - val_acc: 0.4054
Epoch 8/10
7278/7278 [==============================] - 18s 3ms/step - loss: 1.3145 - acc: 0.3616 - val_loss: 1.2393 - val_acc: 0.4054
Epoch 9/10
7278/7278 [==============================] - 19s 3ms/step - loss: 1.2991 - acc: 0.3642 - val_loss: 1.2129 - val_acc: 0.4054
Epoch 10/10
7278/7278 [==============================] - 19s 3ms/step - loss: 1.3016 - acc: 0.3762 - val_loss: 1.1999 - val_acc: 0.4054
Best test accuracy obtained is 0.498812351543943
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_5 (LSTM)                (None, 128, 32)           5376      
_________________________________________________________________
dropout_5 (Dropout)          (None, 128, 32)           0         
_________________________________________________________________
lstm_6 (LSTM)                (None, 32)                8320      
_________________________________________________________________
dropout_6 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 198       
=================================================================
Total params: 13,894
Trainable params: 13,894
Non-trainable params: 0
_________________________________________________________________
None
Train on 7278 samples, validate on 74 samples
Epoch 1/10
7278/7278 [==============================] - 22s 3ms/step - loss: 1.5652 - acc: 0.3534 - val_loss: 1.8373 - val_acc: 0.0000e+00
Epoch 2/10
7278/7278 [==============================] - 20s 3ms/step - loss: 1.2254 - acc: 0.4739 - val_loss: 3.1153 - val_acc: 0.0270
Epoch 3/10
7278/7278 [==============================] - 20s 3ms/step - loss: 1.0148 - acc: 0.5109 - val_loss: 3.6187 - val_acc: 0.1351
Epoch 4/10
7278/7278 [==============================] - 20s 3ms/step - loss: 0.8821 - acc: 0.5256 - val_loss: 4.3753 - val_acc: 0.0676
Epoch 5/10
7278/7278 [==============================] - 20s 3ms/step - loss: 0.8136 - acc: 0.5403 - val_loss: 4.8265 - val_acc: 0.1351
Epoch 6/10
7278/7278 [==============================] - 20s 3ms/step - loss: 0.7819 - acc: 0.5486 - val_loss: 4.0687 - val_acc: 0.1757
Epoch 7/10
7278/7278 [==============================] - 20s 3ms/step - loss: 0.7649 - acc: 0.5573 - val_loss: 4.8145 - val_acc: 0.1216
```

```
Epoch 8/10
7278/7278 [==============================] - 20s 3ms/step - loss: 0.7182 - acc: 0.5919 - val_loss: 4.6194 - val_acc: 0.0405
Epoch 9/10
7278/7278 [==============================] - 20s 3ms/step - loss: 0.6767 - acc: 0.6322 - val_loss: 2.6339 - val_acc: 0.1622
Epoch 10/10
7278/7278 [==============================] - 20s 3ms/step - loss: 0.6318 - acc: 0.6485 - val_loss: 2.0401 - val_acc: 0.2027
Best test accuracy obtained is 0.4533423820834747
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_7 (LSTM)                (None, 128, 64)           18944     
_________________________________________________________________
dropout_7 (Dropout)          (None, 128, 64)           0         
_________________________________________________________________
lstm_8 (LSTM)                (None, 32)                12416     
_________________________________________________________________
dropout_8 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 198       
=================================================================
Total params: 31,558
Trainable params: 31,558
Non-trainable params: 0
_________________________________________________________________
None
Train on 7278 samples, validate on 74 samples
Epoch 1/10
7278/7278 [==============================] - 29s 4ms/step - loss: 1.3602 - acc: 0.4485 - val_loss: 1.4391 - val_acc: 0.2297
Epoch 2/10
7278/7278 [==============================] - 26s 4ms/step - loss: 1.0418 - acc: 0.5485 - val_loss: 1.2397 - val_acc: 0.2432
Epoch 3/10
7278/7278 [==============================] - 26s 4ms/step - loss: 0.9127 - acc: 0.6151 - val_loss: 1.1559 - val_acc: 0.2432
Epoch 4/10
7278/7278 [==============================] - 26s 4ms/step - loss: 0.7731 - acc: 0.6654 - val_loss: 1.1272 - val_acc: 0.4189
Epoch 5/10
7278/7278 [==============================] - 26s 4ms/step - loss: 0.7067 - acc: 0.7053 - val_loss: 1.0828 - val_acc: 0.3649
Epoch 6/10
7278/7278 [==============================] - 26s 4ms/step - loss: 0.6940 - acc: 0.7270 - val_loss: 1.0546 - val_acc: 0.5000
Epoch 7/10
7278/7278 [==============================] - 26s 4ms/step - loss: 0.6086 - acc: 0.7677 - val_loss: 0.7399 - val_acc: 0.5946
Epoch 8/10
7278/7278 [==============================] - 26s 4ms/step - loss: 0.5456 - acc: 0.7874 - val_loss: 0.8971 - val_acc: 0.7297
Epoch 9/10
7278/7278 [==============================] - 26s 4ms/step - loss: 0.5081 - acc: 0.8203 - val_loss: 0.5811 - val_acc: 0.7838
Epoch 10/10
7278/7278 [==============================] - 26s 4ms/step - loss: 0.4576 - acc: 0.8405 - val_loss: 0.4011 - val_acc: 0.9730
Best test accuracy obtained is 0.8686800135731252
2947/2947 [==============================] - 3s 1ms/step
```

In [0]:

```
best_score = best_model.evaluate(x_test, y_test)
print('Best accuracy found is {}%'.format(best_score[1]*100))
```

```
2947/2947 [==============================] - 3s 1ms/step
Best accuracy found is 86.86800135731252%
```

In [0]:

```
print('Best parameters found are:')
best_run
```

```
Best parameters found are:
```

Out[18]:

```
{'Dropout': 0.41266207281071243,
 'Dropout_1': 0.4844455237320119,
 'LSTM': 1,
 'LSTM_1': 1}
```

In [0]:

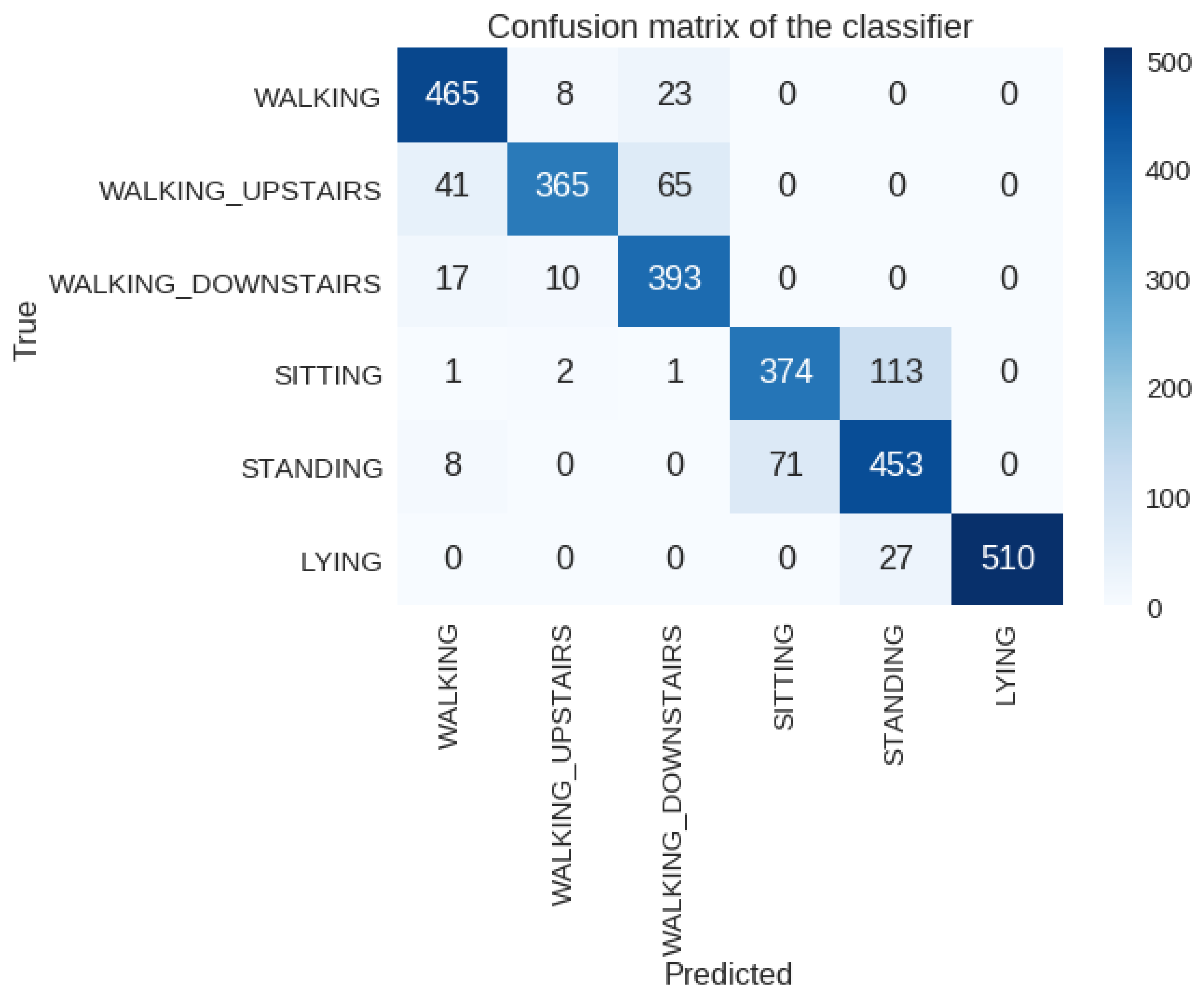
```
y_predicted = best_model.predict(x_test)
```

In [0]:

```
y_true = [np.argmax(action) for action in y_test]
y_pred = [np.argmax(action) for action in y_predicted]
```

In [0]:
```
plot_confusion_matrix(y_true, y_pred)
```

```
[[465   8  23   0   0   0]
 [ 41 365  65   0   0   0]
 [ 17  10 393   0   0   0]
 [  1   2   1 374 113   0]
 [  8   0   0  71 453   0]
 [  0   0   0   0  27 510]]
```




```
**************************************************
0.9375
0.7749469214437368
0.9357142857142857
0.7617107942973523
0.8515037593984962
0.9497206703910615
The True Positive Rate observed is:
 0.8686800135731252
0.9726642186862505
0.9919224555735057
0.964780371982588
0.9710912052117264
0.9420289855072463
1.0
The True Negative Rate observed is:
 0.9738072233881917
0.02733578131374949
0.008077544426494346
0.03521962801741195
0.028908794788273615
0.057971014492753624
0.0
The False Positive Rate observed is:
 0.02619277661180824
0.0625
0.22505307855626328
0.06428571428571428
0.23828920570264767
0.14849624060150377
0.05027932960893855
The False Negative Rate observed is:
 0.13131998642687479
**************************************************
The stats observed for confusion matrix are:
Confusion Matrix:

Predicted    0    1    2    3    4    5  __all__
Actual
0          465    8   23    0    0    0      496
1           41  365   65    0    0    0      471
2           17   10  393    0    0    0      420
3            1    2    1  374  113    0      491
4            8    0    0   71  453    0      532
5            0    0    0    0   27  510      537
__all__    532  385  482  445  593  510     2947


Overall Statistics:

Accuracy: 0.8686800135731252
95% CI: (0.8559495843671653, 0.88067336102849)
No Information Rate: ToDo
P-Value [Acc > NIR]: 0.0
Kappa: 0.8422412007988025
Mcnemar's Test P-Value: ToDo


Class Statistics:

Classes                                        0          1          2  \
Population                                  2947       2947       2947
P: Condition positive                        496        471        420
N: Condition negative                       2451       2476       2527
Test outcome positive                        532        385        482
Test outcome negative                       2415       2562       2465
TP: True Positive                            465        365        393
TN: True Negative                           2384       2456       2438
FP: False Positive                            67         20         89
FN: False Negative                            31        106         27
TPR: (Sensitivity, hit rate, recall)      0.9375   0.774947   0.935714
TNR=SPC: (Specificity)                  0.972664   0.991922    0.96478
PPV: Pos Pred Value (Precision)          0.87406   0.948052   0.815353
NPV: Neg Pred Value                     0.987164   0.958626   0.989047
FPR: False-out                         0.0273358 0.00807754  0.0352196
FDR: False Discovery Rate                0.12594  0.0519481   0.184647
FNR: Miss Rate                            0.0625   0.225053  0.0642857
ACC: Accuracy                           0.966746   0.957245   0.960638
F1 score                                0.904669   0.852804   0.871397
MCC: Matthews correlation coefficient   0.885356   0.833849   0.851092
Informedness                            0.910164   0.766869   0.900495
Markedness                              0.861224   0.906678   0.804399
Prevalence                              0.168307   0.159824   0.142518
LR+: Positive likelihood ratio           34.2957    95.9384     26.568
LR-: Negative likelihood ratio         0.0642565   0.226886  0.0666325
DOR: Diagnostic odds ratio               533.731    422.849    398.724
```

```
FOR: False omission rate              0.0128364   0.0413739  0.0109533

Classes                                       3           4          5
Population                                 2947        2947       2947
P: Condition positive                       491         532        537
N: Condition negative                      2456        2415       2410
Test outcome positive                       445         593        510
Test outcome negative                      2502        2354       2437
TP: True Positive                           374         453        510
TN: True Negative                          2385        2275       2410
FP: False Positive                           71         140          0
FN: False Negative                          117          79         27
TPR: (Sensitivity, hit rate, recall)   0.761711    0.851504   0.949721
TNR=SPC: (Specificity)                 0.971091    0.942029          1
PPV: Pos Pred Value (Precision)        0.840449    0.763912          1
NPV: Neg Pred Value                    0.953237     0.96644   0.988921
FPR: False-out                        0.0289088    0.057971          0
FDR: False Discovery Rate              0.159551    0.236088          0
FNR: Miss Rate                         0.238289    0.148496  0.0502793
ACC: Accuracy                          0.936206    0.925687   0.990838
F1 score                               0.799145    0.805333   0.974212
MCC: Matthews correlation coefficient  0.762637    0.761287   0.969123
Informedness                           0.732802    0.793533   0.949721
Markedness                             0.793687    0.730352   0.988921
Prevalence                              0.16661    0.180523   0.182219
LR+: Positive likelihood ratio          26.3488     14.6884        inf
LR-: Negative likelihood ratio         0.245383    0.157634  0.0502793
DOR: Diagnostic odds ratio              107.378     93.1804        inf
FOR: False omission rate              0.0467626   0.0335599  0.0110792
```

Hyperas parameter tuning 2

In [0]:
```python
def data():
    x_train = np.load("x_train.npy")
    y_train = np.load("y_train.npy")
    x_test = np.load("x_test.npy")
    y_test = np.load("y_test.npy")
    return x_train, y_train, x_test, y_test
```

In [0]:
```python
def lstm_model(x_train, y_train, x_test, y_test):

    epochs = 20
    batch_size = 64
    timesteps = x_train.shape[1]
    input_dim = len(x_train[0][0])
    n_classes = 6

    model = Sequential()

    model.add(LSTM({{choice([32, 64])}}, return_sequences = True, input_shape = (timesteps, input_dim)))
    model.add(Dropout({{uniform(0, 1)}}))

    model.add(LSTM({{choice([16, 32])}}))
    model.add(Dropout({{uniform(0, 1)}}))

    model.add(Dense(n_classes, activation='sigmoid'))

    print(model.summary())

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

    result = model.fit(x_train, y_train, batch_size = batch_size, epochs=epochs, verbose=1, validation_split=0.01)

    scores = model.evaluate(x_test, y_test, verbose=0)

    print('Best test accuracy obtained is',scores[1])

    return {'loss': -scores[1], 'status': STATUS_OK, 'model': model}
```

In [0]:
```
best_run, best_model = optim.minimize(model=lstm_model, data=data, algo=tpe.suggest, max_evals=4, trials=Trials(), notebook_name = "Human Activity Detection")
x_train, y_train, x_test, y_test = data()

score = best_model.evaluate(x_test, y_test)
```

```
>>> Imports:
#coding=utf-8

try:
    from google.colab import auth
except:
    pass

try:
    from oauth2client.client import GoogleCredentials
except:
    pass

try:
    import getpass
except:
    pass

try:
    import numpy as np
except:
    pass

try:
    import pandas as pd
except:
    pass

try:
    from keras.models import Sequential
except:
    pass

try:
    from keras.layers import LSTM
except:
    pass

try:
    from keras.layers.core import Dense, Dropout
except:
    pass

try:
    from hyperopt import Trials, STATUS_OK, tpe
except:
    pass

try:
    from hyperas import optim
except:
    pass

try:
    from hyperas.distributions import choice, uniform
except:
    pass

try:
    import warnings
except:
    pass

try:
    from pydrive.auth import GoogleAuth
except:
    pass

try:
    from pydrive.drive import GoogleDrive
except:
    pass

try:
    from google.colab import auth
except:
    pass

try:
    from oauth2client.client import GoogleCredentials
except:
    pass

try:
    from pandas_ml import ConfusionMatrix
except:
    pass

try:
    import joblib
except:
    pass

try:
    import itertools
except:
    pass

try:
    import numpy as np
except:
    pass

try:
    import matplotlib.pyplot as plt
except:
    pass

try:
    from sklearn.metrics import confusion_matrix
except:
    pass

try:
    import seaborn as sns
except:
    pass

try:
    from sklearn.metrics import accuracy_score
except:
```

```
    pass

try:
    from sklearn.metrics import classification_report
except:
    pass

try:
    from sklearn.metrics import confusion_matrix
except:
    pass

try:
    from sklearn import metrics
except:
    pass

try:
    from pandas_ml import ConfusionMatrix
except:
    pass

>>> Hyperas search space:

def get_space():
    return {
        'LSTM': hp.choice('LSTM', [32, 64]),
        'Dropout': hp.uniform('Dropout', 0, 1),
        'LSTM_1': hp.choice('LSTM_1', [16, 32]),
        'Dropout_1': hp.uniform('Dropout_1', 0, 1),
    }

>>> Data
  1:
  2: x_train = np.load("x_train.npy")
  3: y_train = np.load("y_train.npy")
  4: x_test = np.load("x_test.npy")
  5: y_test = np.load("y_test.npy")
  6:
  7:
  8:
>>> Resulting replaced keras model:

  1: def keras_fmin_fnct(space):
  2:
  3:
  4:     epochs = 20
  5:     batch_size = 64
  6:     timesteps = x_train.shape[1]
  7:     input_dim = len(x_train[0][0])
  8:     n_classes = 6
  9:
 10:     model = Sequential()
 11:
 12:     model.add(LSTM(space['LSTM'], return_sequences = True, input_shape = (timesteps, input_dim)))
 13:     model.add(Dropout(space['Dropout']))
 14:
 15:     model.add(LSTM(space['LSTM_1']))
 16:     model.add(Dropout(space['Dropout_1']))
 17:
 18:     model.add(Dense(n_classes, activation='sigmoid'))
 19:
 20:     print(model.summary())
 21:
 22:     model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='rmsprop')
 23:
 24:     result = model.fit(x_train, y_train, batch_size = batch_size, epochs=epochs, verbose=1, validation_split=0.01)
 25:
 26:     scores = model.evaluate(x_test, y_test, verbose=0)
 27:
 28:     print('Best test accuracy obtained is',scores[1])
 29:
 30:     return {'loss': -scores[1], 'status': STATUS_OK, 'model': model}
 31:
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 128, 64)           18944
_________________________________________________________________
dropout_1 (Dropout)          (None, 128, 64)           0
_________________________________________________________________
lstm_2 (LSTM)                (None, 16)                5184
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 102
=================================================================
Total params: 24,230
Trainable params: 24,230
Non-trainable params: 0
_________________________________________________________________
None
Train on 7278 samples, validate on 74 samples
Epoch 1/20
7278/7278 [==============================] - 29s 4ms/step - loss: 1.5711 - acc: 0.3670 - val_loss: 1.6629 - val_acc: 0.2432
Epoch 2/20
7278/7278 [==============================] - 26s 4ms/step - loss: 1.3756 - acc: 0.4522 - val_loss: 1.4983 - val_acc: 0.4054
Epoch 3/20
7278/7278 [==============================] - 26s 4ms/step - loss: 1.2221 - acc: 0.5109 - val_loss: 1.4024 - val_acc: 0.4054
Epoch 4/20
7278/7278 [==============================] - 25s 4ms/step - loss: 1.1429 - acc: 0.5309 - val_loss: 1.3529 - val_acc: 0.4054
Epoch 5/20
7278/7278 [==============================] - 26s 4ms/step - loss: 1.0573 - acc: 0.5624 - val_loss: 1.2993 - val_acc: 0.4054
Epoch 6/20
7278/7278 [==============================] - 26s 4ms/step - loss: 0.9968 - acc: 0.5812 - val_loss: 1.2380 - val_acc: 0.4054
Epoch 7/20
7278/7278 [==============================] - 26s 4ms/step - loss: 0.9510 - acc: 0.5925 - val_loss: 1.2004 - val_acc: 0.4054
Epoch 8/20
7278/7278 [==============================] - 26s 4ms/step - loss: 0.9021 - acc: 0.6004 - val_loss: 1.1676 - val_acc: 0.4054
Epoch 9/20
7278/7278 [==============================] - 26s 4ms/step - loss: 0.8940 - acc: 0.6105 - val_loss: 1.1585 - val_acc: 0.4054
Epoch 10/20
7278/7278 [==============================] - 26s 4ms/step - loss: 0.8688 - acc: 0.6125 - val_loss: 1.1412 - val_acc: 0.4054
Epoch 11/20
7278/7278 [==============================] - 26s 4ms/step - loss: 0.8373 - acc: 0.6138 - val_loss: 1.1313 - val_acc: 0.4054
Epoch 12/20
7278/7278 [==============================] - 26s 4ms/step - loss: 0.8441 - acc: 0.6168 - val_loss: 1.1377 - val_acc: 0.4054
Epoch 13/20
7278/7278 [==============================] - 26s 4ms/step - loss: 0.8121 - acc: 0.6245 - val_loss: 1.1207 - val_acc: 0.4054
Epoch 14/20
7278/7278 [==============================] - 26s 4ms/step - loss: 0.8017 - acc: 0.6275 - val_loss: 1.1163 - val_acc: 0.4054
Epoch 15/20
7278/7278 [==============================] - 26s 4ms/step - loss: 0.7781 - acc: 0.6319 - val_loss: 1.1167 - val_acc: 0.4054
Epoch 16/20
7278/7278 [==============================] - 26s 4ms/step - loss: 0.7677 - acc: 0.6320 - val_loss: 1.1131 - val_acc: 0.4054
Epoch 17/20
7278/7278 [==============================] - 26s 4ms/step - loss: 0.7623 - acc: 0.6371 - val_loss: 1.1106 - val_acc: 0.4054
```

```
Epoch 18/20
7278/7278 [==============================] - 26s 4ms/step - loss: 0.7750 - acc: 0.6298 - val_loss: 1.1085 - val_acc: 0.2162
Epoch 19/20
7278/7278 [==============================] - 26s 4ms/step - loss: 0.7659 - acc: 0.6356 - val_loss: 1.1085 - val_acc: 0.2162
Epoch 20/20
7278/7278 [==============================] - 26s 4ms/step - loss: 0.7522 - acc: 0.6378 - val_loss: 1.1066 - val_acc: 0.2162
Best test accuracy obtained is 0.6297930098405158
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_3 (LSTM)                (None, 128, 32)           5376      
_________________________________________________________________
dropout_3 (Dropout)          (None, 128, 32)           0         
_________________________________________________________________
lstm_4 (LSTM)                (None, 16)                3136      
_________________________________________________________________
dropout_4 (Dropout)          (None, 16)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 102       
=================================================================
Total params: 8,614
Trainable params: 8,614
Non-trainable params: 0
_________________________________________________________________
None
Train on 7278 samples, validate on 74 samples
Epoch 1/20
7278/7278 [==============================] - 22s 3ms/step - loss: 1.7066 - acc: 0.2505 - val_loss: 1.7280 - val_acc: 0.0000e+00
Epoch 2/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.6090 - acc: 0.2965 - val_loss: 1.6978 - val_acc: 0.0676
Epoch 3/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.5381 - acc: 0.3390 - val_loss: 1.6638 - val_acc: 0.0000e+00
Epoch 4/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.5165 - acc: 0.3413 - val_loss: 1.6234 - val_acc: 0.0000e+00
Epoch 5/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.4773 - acc: 0.3494 - val_loss: 1.6010 - val_acc: 0.0000e+00
Epoch 6/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.4574 - acc: 0.3545 - val_loss: 1.5805 - val_acc: 0.0000e+00
Epoch 7/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.4241 - acc: 0.3684 - val_loss: 1.5475 - val_acc: 0.0000e+00
Epoch 8/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.4173 - acc: 0.3691 - val_loss: 1.5309 - val_acc: 0.0000e+00
Epoch 9/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.3919 - acc: 0.3721 - val_loss: 1.5006 - val_acc: 0.0000e+00
Epoch 10/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.3754 - acc: 0.3766 - val_loss: 1.4882 - val_acc: 0.0000e+00
Epoch 11/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.3512 - acc: 0.3768 - val_loss: 1.4711 - val_acc: 0.0000e+00
Epoch 12/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.3482 - acc: 0.3670 - val_loss: 1.4556 - val_acc: 0.2027
Epoch 13/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.3470 - acc: 0.3794 - val_loss: 1.4418 - val_acc: 0.2162
Epoch 14/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.3136 - acc: 0.3785 - val_loss: 1.4317 - val_acc: 0.2162
Epoch 15/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.3025 - acc: 0.3769 - val_loss: 1.4205 - val_acc: 0.2162
Epoch 16/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.2945 - acc: 0.3868 - val_loss: 1.4144 - val_acc: 0.2162
Epoch 17/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.2973 - acc: 0.3882 - val_loss: 1.4055 - val_acc: 0.2162
Epoch 18/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.2916 - acc: 0.3904 - val_loss: 1.4026 - val_acc: 0.2162
Epoch 19/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.2784 - acc: 0.4007 - val_loss: 1.3969 - val_acc: 0.2162
Epoch 20/20
7278/7278 [==============================] - 20s 3ms/step - loss: 1.2875 - acc: 0.3941 - val_loss: 1.3943 - val_acc: 0.2162
Best test accuracy obtained is 0.6253817441465898
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_5 (LSTM)                (None, 128, 32)           5376      
_________________________________________________________________
dropout_5 (Dropout)          (None, 128, 32)           0         
_________________________________________________________________
lstm_6 (LSTM)                (None, 32)                8320      
_________________________________________________________________
dropout_6 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 198       
=================================================================
Total params: 13,894
Trainable params: 13,894
Non-trainable params: 0
_________________________________________________________________
None
Train on 7278 samples, validate on 74 samples
Epoch 1/20
7278/7278 [==============================] - 24s 3ms/step - loss: 1.5512 - acc: 0.3557 - val_loss: 1.7369 - val_acc: 0.0000e+00
Epoch 2/20
7278/7278 [==============================] - 21s 3ms/step - loss: 1.3250 - acc: 0.4077 - val_loss: 1.8506 - val_acc: 0.0000e+00
Epoch 3/20
7278/7278 [==============================] - 21s 3ms/step - loss: 1.1176 - acc: 0.5077 - val_loss: 1.6154 - val_acc: 0.2027
Epoch 4/20
7278/7278 [==============================] - 21s 3ms/step - loss: 0.9342 - acc: 0.5922 - val_loss: 1.3516 - val_acc: 0.2297
Epoch 5/20
7278/7278 [==============================] - 21s 3ms/step - loss: 0.8167 - acc: 0.6235 - val_loss: 1.1654 - val_acc: 0.2162
Epoch 6/20
7278/7278 [==============================] - 22s 3ms/step - loss: 0.7567 - acc: 0.6340 - val_loss: 1.2681 - val_acc: 0.2162
Epoch 7/20
7278/7278 [==============================] - 22s 3ms/step - loss: 0.7394 - acc: 0.6287 - val_loss: 1.3762 - val_acc: 0.2162
Epoch 8/20
7278/7278 [==============================] - 21s 3ms/step - loss: 0.6881 - acc: 0.6620 - val_loss: 1.5418 - val_acc: 0.2162
Epoch 9/20
7278/7278 [==============================] - 21s 3ms/step - loss: 0.6700 - acc: 0.6712 - val_loss: 1.8856 - val_acc: 0.2162
Epoch 10/20
7278/7278 [==============================] - 21s 3ms/step - loss: 0.6238 - acc: 0.7010 - val_loss: 2.1653 - val_acc: 0.2162
Epoch 11/20
7278/7278 [==============================] - 21s 3ms/step - loss: 0.5793 - acc: 0.7303 - val_loss: 2.8013 - val_acc: 0.2162
Epoch 12/20
7278/7278 [==============================] - 21s 3ms/step - loss: 0.5426 - acc: 0.7609 - val_loss: 2.2062 - val_acc: 0.2432
Epoch 13/20
7278/7278 [==============================] - 22s 3ms/step - loss: 0.4866 - acc: 0.7738 - val_loss: 3.2200 - val_acc: 0.2162
Epoch 14/20
7278/7278 [==============================] - 21s 3ms/step - loss: 0.4569 - acc: 0.7859 - val_loss: 2.2487 - val_acc: 0.0676
Epoch 15/20
7278/7278 [==============================] - 21s 3ms/step - loss: 0.4235 - acc: 0.7933 - val_loss: 2.2888 - val_acc: 0.2297
Epoch 16/20
7278/7278 [==============================] - 21s 3ms/step - loss: 0.4133 - acc: 0.7968 - val_loss: 1.9614 - val_acc: 0.2703
Epoch 17/20
7278/7278 [==============================] - 21s 3ms/step - loss: 0.3871 - acc: 0.8009 - val_loss: 1.1401 - val_acc: 0.4189
Epoch 18/20
7278/7278 [==============================] - 21s 3ms/step - loss: 0.3883 - acc: 0.7954 - val_loss: 1.8715 - val_acc: 0.3784
Epoch 19/20
7278/7278 [==============================] - 21s 3ms/step - loss: 0.3765 - acc: 0.8019 - val_loss: 3.8171 - val_acc: 0.0811
Epoch 20/20
7278/7278 [==============================] - 21s 3ms/step - loss: 0.3649 - acc: 0.8086 - val_loss: 1.6876 - val_acc: 0.3514
Best test accuracy obtained is 0.5890736342042755
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_7 (LSTM)                (None, 128, 64)           18944     
_________________________________________________________________
dropout_7 (Dropout)          (None, 128, 64)           0         
_________________________________________________________________
lstm_8 (LSTM)                (None, 32)                12416     
_________________________________________________________________
dropout_8 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 198       
=================================================================
Total params: 31,558
Trainable params: 31,558
Non-trainable params: 0
_________________________________________________________________
None
Train on 7278 samples, validate on 74 samples
Epoch 1/20
7278/7278 [==============================] - 30s 4ms/step - loss: 1.3833 - acc: 0.4148 - val_loss: 1.4854 - val_acc: 0.0811
Epoch 2/20
7278/7278 [==============================] - 27s 4ms/step - loss: 1.0964 - acc: 0.5595 - val_loss: 1.2928 - val_acc: 0.4865
Epoch 3/20
7278/7278 [==============================] - 27s 4ms/step - loss: 0.9546 - acc: 0.5984 - val_loss: 1.1911 - val_acc: 0.4595
Epoch 4/20
7278/7278 [==============================] - 27s 4ms/step - loss: 0.8068 - acc: 0.6654 - val_loss: 1.0861 - val_acc: 0.5135
Epoch 5/20
7278/7278 [==============================] - 28s 4ms/step - loss: 0.7227 - acc: 0.7084 - val_loss: 1.0744 - val_acc: 0.4730
Epoch 6/20
7278/7278 [==============================] - 28s 4ms/step - loss: 0.6869 - acc: 0.7197 - val_loss: 0.9725 - val_acc: 0.5000
Epoch 7/20
7278/7278 [==============================] - 28s 4ms/step - loss: 0.6442 - acc: 0.7399 - val_loss: 0.8230 - val_acc: 0.6216
Epoch 8/20
7278/7278 [==============================] - 28s 4ms/step - loss: 0.6014 - acc: 0.7483 - val_loss: 1.1658 - val_acc: 0.4459
Epoch 9/20
7278/7278 [==============================] - 28s 4ms/step - loss: 0.5633 - acc: 0.7561 - val_loss: 0.7938 - val_acc: 0.5270
Epoch 10/20
7278/7278 [==============================] - 28s 4ms/step - loss: 0.5181 - acc: 0.7758 - val_loss: 0.7012 - val_acc: 0.5946
Epoch 11/20
7278/7278 [==============================] - 28s 4ms/step - loss: 0.4690 - acc: 0.7890 - val_loss: 0.5889 - val_acc: 0.6216
Epoch 12/20
7278/7278 [==============================] - 27s 4ms/step - loss: 0.4395 - acc: 0.8082 - val_loss: 0.5710 - val_acc: 0.6351
Epoch 13/20
7278/7278 [==============================] - 27s 4ms/step - loss: 0.4332 - acc: 0.8059 - val_loss: 0.5639 - val_acc: 0.6351
Epoch 14/20
7278/7278 [==============================] - 27s 4ms/step - loss: 0.3960 - acc: 0.8063 - val_loss: 0.7376 - val_acc: 0.5405
Epoch 15/20
7278/7278 [==============================] - 27s 4ms/step - loss: 0.3857 - acc: 0.8142 - val_loss: 0.5594 - val_acc: 0.6216
Epoch 16/20
7278/7278 [==============================] - 27s 4ms/step - loss: 0.4111 - acc: 0.8134 - val_loss: 0.5982 - val_acc: 0.6216
Epoch 17/20
7278/7278 [==============================] - 28s 4ms/step - loss: 0.3452 - acc: 0.8539 - val_loss: 0.3621 - val_acc: 0.9324
Epoch 18/20
7278/7278 [==============================] - 28s 4ms/step - loss: 0.2983 - acc: 0.8974 - val_loss: 0.2826 - val_acc: 1.0000
Epoch 19/20
7278/7278 [==============================] - 28s 4ms/step - loss: 0.2797 - acc: 0.9226 - val_loss: 0.1726 - val_acc: 1.0000
Epoch 20/20
7278/7278 [==============================] - 28s 4ms/step - loss: 0.2291 - acc: 0.9376 - val_loss: 0.0646 - val_acc: 1.0000
Best test accuracy obtained is 0.9066847641669494
2947/2947 [==============================] - 4s 1ms/step
```

In [0]:
```python
best_score = best_model.evaluate(x_test, y_test)
print('Best accuracy found is {}%'.format(best_score[1]*100))
```

```
2947/2947 [==============================] - 4s 1ms/step
Best accuracy found is 90.66847641669494%
```

In [0]:
```python
print('Best parameters found are:')
best_run
```

```
Best parameters found are:
```

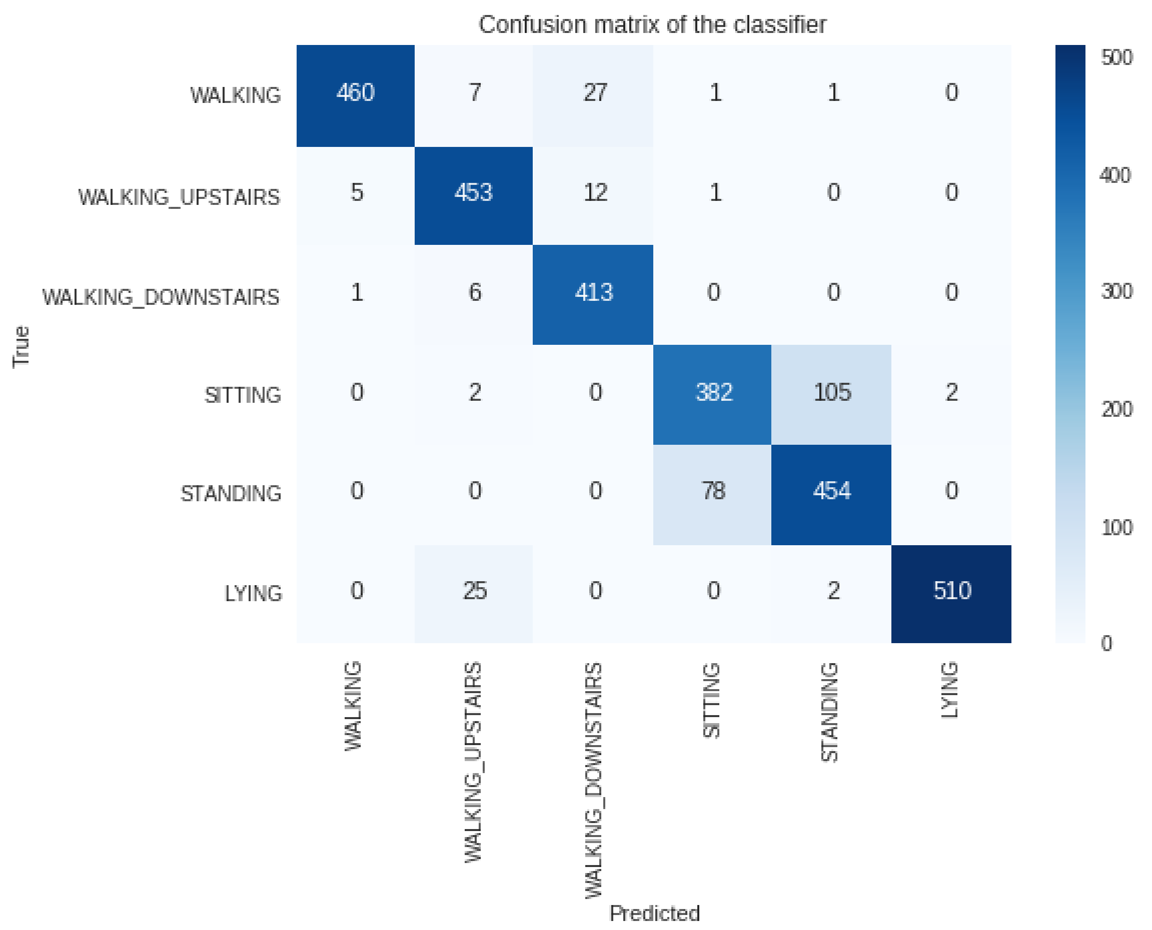
Out[17]:
```
{'Dropout': 0.41266207281071243,
 'Dropout_1': 0.4844455237320119,
 'LSTM': 1,
 'LSTM_1': 1}
```

In [0]:
```python
y_predicted = best_model.predict(x_test)
```

In [0]:
```python
y_true = [np.argmax(action) for action in y_test]
y_pred = [np.argmax(action) for action in y_predicted]
```

In [0]:
```
plot_confusion_matrix(y_true, y_pred)
```

```
[[460   7  27   1   1   0]
 [  5 453  12   1   0   0]
 [  1   6 413   0   0   0]
 [  0   2   0 382 105   2]
 [  0   0   0  78 454   0]
 [  0  25   0   0   2 510]]
```




```
**************************************************
0.9274193548387096
0.9617834394904459
0.9833333333333333
0.7780040733197556
0.8533834586466166
0.9497206703910615
The True Positive Rate observed is:
 0.9066847641669494
0.9975520195838433
0.9838449111470113
0.9845666798575385
0.9674267100977199
0.9552795031055901
0.9991701244813278
The True Negative Rate observed is:
 0.9811551841060815
0.0024479804161566705
0.01615508885298869
0.015433320142461416
0.03257328990228013
0.04472049689440994
0.0008298755186721991
The False Positive Rate observed is:
 0.018844815893918485
0.07258064516129033
0.03821656050955414
0.016666666666666666
0.2219959266802444
0.14661654135338345
0.05027932960893855
The False Negative Rate observed is:
 0.09331523583305056
**************************************************
The stats observed for confusion matrix are:
Confusion Matrix:

Predicted    0    1    2    3    4    5  __all__
Actual
0          460    7   27    1    1    0      496
1            5  453   12    1    0    0      471
2            1    6  413    0    0    0      420
3            0    2    0  382  105    2      491
4            0    0    0   78  454    0      532
5            0   25    0    0    2  510      537
__all__    466  493  452  462  562  512     2947


Overall Statistics:

Accuracy: 0.9066847641669494
95% CI: (0.8956055960159328, 0.9169485368472383)
No Information Rate: ToDo
P-Value [Acc > NIR]: 0.0
Kappa: 0.8879213537532188
Mcnemar's Test P-Value: ToDo


Class Statistics:

Classes                                        0          1          2  \
Population                                  2947       2947       2947
P: Condition positive                        496        471        420
N: Condition negative                       2451       2476       2527
Test outcome positive                        466        493        452
Test outcome negative                       2481       2454       2495
TP: True Positive                            460        453        413
TN: True Negative                           2445       2436       2488
FP: False Positive                             6         40         39
FN: False Negative                            36         18          7
TPR: (Sensitivity, hit rate, recall)    0.927419   0.961783   0.983333
TNR=SPC: (Specificity)                  0.997552   0.983845   0.984567
PPV: Pos Pred Value (Precision)         0.987124   0.918864   0.913717
NPV: Neg Pred Value                      0.98549   0.992665   0.997194
FPR: False-out                        0.00244798  0.0161551  0.0154333
FDR: False Discovery Rate              0.0128755  0.0811359  0.0862832
FNR: Miss Rate                         0.0725806  0.0382166  0.0166667
ACC: Accuracy                           0.985748   0.980319   0.984391
F1 score                                0.956341   0.939834   0.947248
MCC: Matthews correlation coefficient   0.948494   0.928422   0.938973
Informedness                            0.924971   0.945628     0.9679
Markedness                              0.972614   0.911529   0.910911
Prevalence                              0.168307   0.159824   0.142518
LR+: Positive likelihood ratio           378.851    59.5344     63.715
LR-: Negative likelihood ratio         0.0727588  0.0388441  0.0169279
DOR: Diagnostic odds ratio               5206.94    1532.65     3763.9
FOR: False omission rate               0.0145103 0.00733496 0.00280561

Classes                                       3          4          5
Population                                 2947       2947       2947
```

```
P: Condition positive                         491        532          537
N: Condition negative                        2456       2415         2410
Test outcome positive                         462        562          512
Test outcome negative                        2485       2385         2435
TP: True Positive                             382        454          510
TN: True Negative                            2376       2307         2408
FP: False Positive                             80        108            2
FN: False Negative                            109         78           27
TPR: (Sensitivity, hit rate, recall)     0.778004   0.853383     0.949721
TNR=SPC: (Specificity)                   0.967427    0.95528      0.99917
PPV: Pos Pred Value (Precision)           0.82684   0.807829     0.996094
NPV: Neg Pred Value                      0.956137   0.967296     0.988912
FPR: False-out                          0.0325733  0.0447205  0.000829876
FDR: False Discovery Rate                 0.17316   0.192171   0.00390625
FNR: Miss Rate                           0.221996   0.146617    0.0502793
ACC: Accuracy                            0.935867   0.936885     0.990159
F1 score                                 0.801679   0.829982     0.972355
MCC: Matthews correlation coefficient    0.763973   0.791716      0.96678
Informedness                             0.745431   0.808663     0.948891
Markedness                               0.782977   0.775125     0.985005
Prevalence                                0.16661   0.180523     0.182219
LR+: Positive likelihood ratio            23.8847    19.0826      1144.41
LR-: Negative likelihood ratio           0.229471    0.15348    0.0503211
DOR: Diagnostic odds ratio                104.086    124.333      22742.2
FOR: False omission rate                0.0438632  0.0327044    0.0110883
```

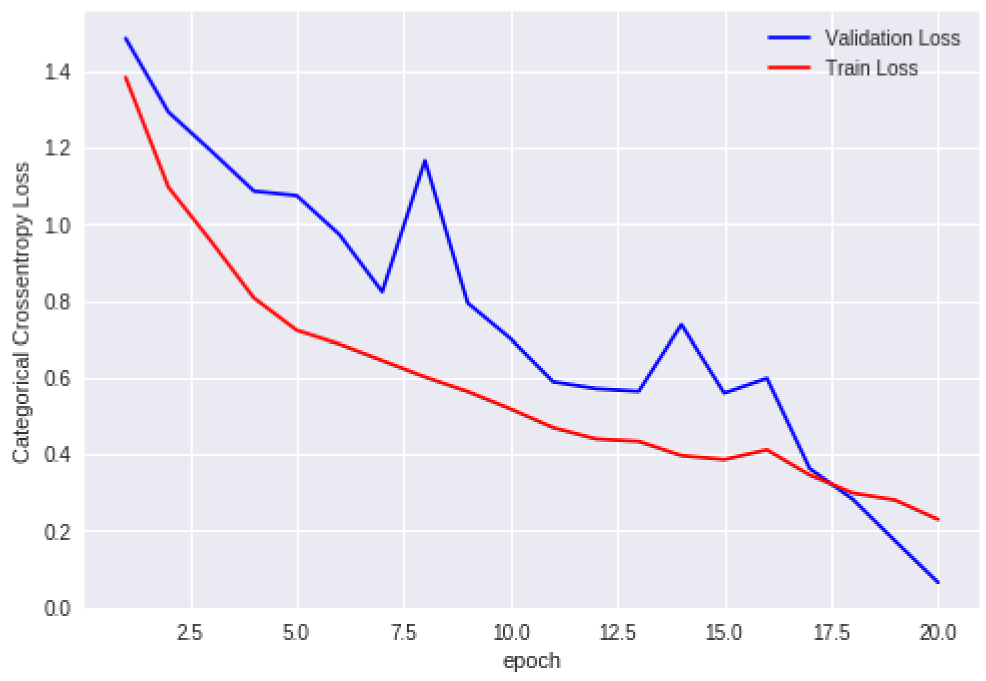
In [0]:

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

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(linestyle='-')
    fig.canvas.draw()
```

In [0]:

```python
import matplotlib.pyplot as plt
#model_scores = best_model.evaluate(x_test, y_test_cat, verbose=0)
print('Test score:', best_score[0])
print('Test accuracy:', best_score[1])

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

# list of epoch numbers
x = list(range(1,21))

hist = best_model.history
vy = hist.history['val_loss']
ty = hist.history['loss']

plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.3531641918902918
Test accuracy: 0.9066847641669494
```



Results and summary

1) We have two types of datasets, a 561 features dataset that is engineered by a domain expert and another dataset with raw signals.

2) After EDA we observed that we might find it difficult to distinguish between 'standing' and 'sitting'.

3) We apply classical machine learning techniques on the 561 features dataset.

4) Linear SVC seemed to give the best results.

5) We apply LSTM model on raw data and record the observations.

6) It is observed that 2 LSTM layers with 20 epochs seemed to have better accuracy and seemed to distinguish between 'sitting' and 'standing' better.

6) Though LSTM model didn't perform as good as classical machine learning models its important to note that LSTM models performed this well on just raw data.

In [0]:
```
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ['Number of LSTMS', 'Hyperparameters', 'Train loss', 'Validation Loss', 'Test Accuracy']
x.add_row(['1', 'LSTM_1: 32' ,0.1945, 0.3087, 0.90974])
x.add_row(['','dropouts : 0.5','','',''])
x.add_row(['','epochs : 30','','',''])
x.add_row(['','','','',''])
x.add_row(['2', 'LSTM_1 : 64', 0.4576, 0.4011, '86.86800'])
x.add_row(['','LSTM_2 : 32','','',''])
x.add_row(['','dropout_1 : 0.41266','','',''])
x.add_row(['','dropout_2 : 0.48444','','',''])
x.add_row(['','epochs : 10','','',''])
x.add_row(['','','','',''])
x.add_row(['2', 'LSTM_1 : 64', 0.2291, 0.0646, 90.66847])
x.add_row(['','LSTM_2 : 32','','',''])
x.add_row(['','dropout_1 : 0.41266','','',''])
x.add_row(['','dropout_2 : 0.48444','','',''])
x.add_row(['','epochs : 20','','',''])
print(x.get_string())
```

```
+-----------------+---------------------+------------+-----------------+---------------+
| Number of LSTMS |   Hyperparameters   | Train loss | Validation Loss | Test Accuracy |
+-----------------+---------------------+------------+-----------------+---------------+
|        1        |      LSTM_1: 32     |   0.1945   |      0.3087     |    0.90974    |
|                 |    dropouts : 0.5   |            |                 |               |
|                 |     epochs : 30     |            |                 |               |
|                 |                     |            |                 |               |
|        2        |     LSTM_1 : 64     |   0.4576   |      0.4011     |    86.86800   |
|                 |     LSTM_2 : 32     |            |                 |               |
|                 | dropout_1 : 0.41266 |            |                 |               |
|                 | dropout_2 : 0.48444 |            |                 |               |
|                 |     epochs : 10     |            |                 |               |
|                 |                     |            |                 |               |
|        2        |     LSTM_1 : 64     |   0.2291   |      0.0646     |    90.66847   |
|                 |     LSTM_2 : 32     |            |                 |               |
|                 | dropout_1 : 0.41266 |            |                 |               |
|                 | dropout_2 : 0.48444 |            |                 |               |
|                 |     epochs : 20     |            |                 |               |
+-----------------+---------------------+------------+-----------------+---------------+
```