

## ▼ HumanActivityRecognition

This project is to build a model that predicts the human activities such as Walking, Walking\_Upsta Laying.

This dataset is collected from 30 persons(referred as subjects in this dataset), performing differer The data is recorded with the help of sensors (accelerometer and Gyroscope) in that smartphone. the data manually.

### How data was recorded

By using the sensors(Gyroscope and accelerometer) in a smartphone, they have captured '3-axial l 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-w seconds each with 50% overlap. ie., each window has 128 readings.
2. From Each window, a feature vector was obtained by calculating variables from the time and

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

3. The accelertion signal was saperated into Body and Gravity acceleration signals(**tBodyAcc-X** pass filter with corner frequency of 0.3Hz.
4. After that, the body linear acceleration and angular velocity were derived in time to obtian *jer* **tBodyGyroJerk-XYZ**).
5. The magnitude of these 3-dimensional signals were calculated using the Euclidian norm. Thi with names like *tBodyAccMag*, *tGravityAccMag*, *tBodyAccJerkMag*, *tBodyGyroMag* and *tBody*
6. Finally, We've got frequency domain signals from some of the available signals by applying a obtained were labeled with **prefix 'f'** just like original signals with **prefix 't'**. These signals are l 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 estimate some set of variables from the above signals. ie., We will estimate the following we recorded 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()**: Autorregression 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 two vectors.

9. We can obtain some other vectors by taking the average of signals in a single window sample

- 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 **training data** and remaining **30%** sub

## 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

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

## ▸ Quick overview of the dataset :

- Accelerometer and Gyroscope readings are taken from 30 volunteers(referred as subjects) w
  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,
- 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,entrc
- 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

Double-click (or enter) to edit

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

🔗 Go to this URL in a browser: [https://accounts.google.com/o/oauth2/auth?client\\_id=9473](https://accounts.google.com/o/oauth2/auth?client_id=9473)

Enter your authorization code:  
 .....  
 Mounted at /content/drive

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

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




```
print(features)
```

```
:\installed\Anaconda3\lib\site-packages\pandas\io\parsers.py:678: UserWarning: Duplica
return _read(filepath_or_buffer, kwds)
```

	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
<b>2261</b>	0.279196	-0.018261	-0.103376	-0.996955	-0.982959	-0.988239	-0.9972	-0.9972	-0.9972

rows × 564 columns

test.shape


 (2947, 564)

Double-click (or enter) to edit

## ▼ Data Cleaning

### ▼ 1. Check for Duplicates


```
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

Double-click (or enter) to edit

### ▼ 2. Checking for NaN/null values

```
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

Double-click (or enter) to edit

Double-click (or enter) to edit

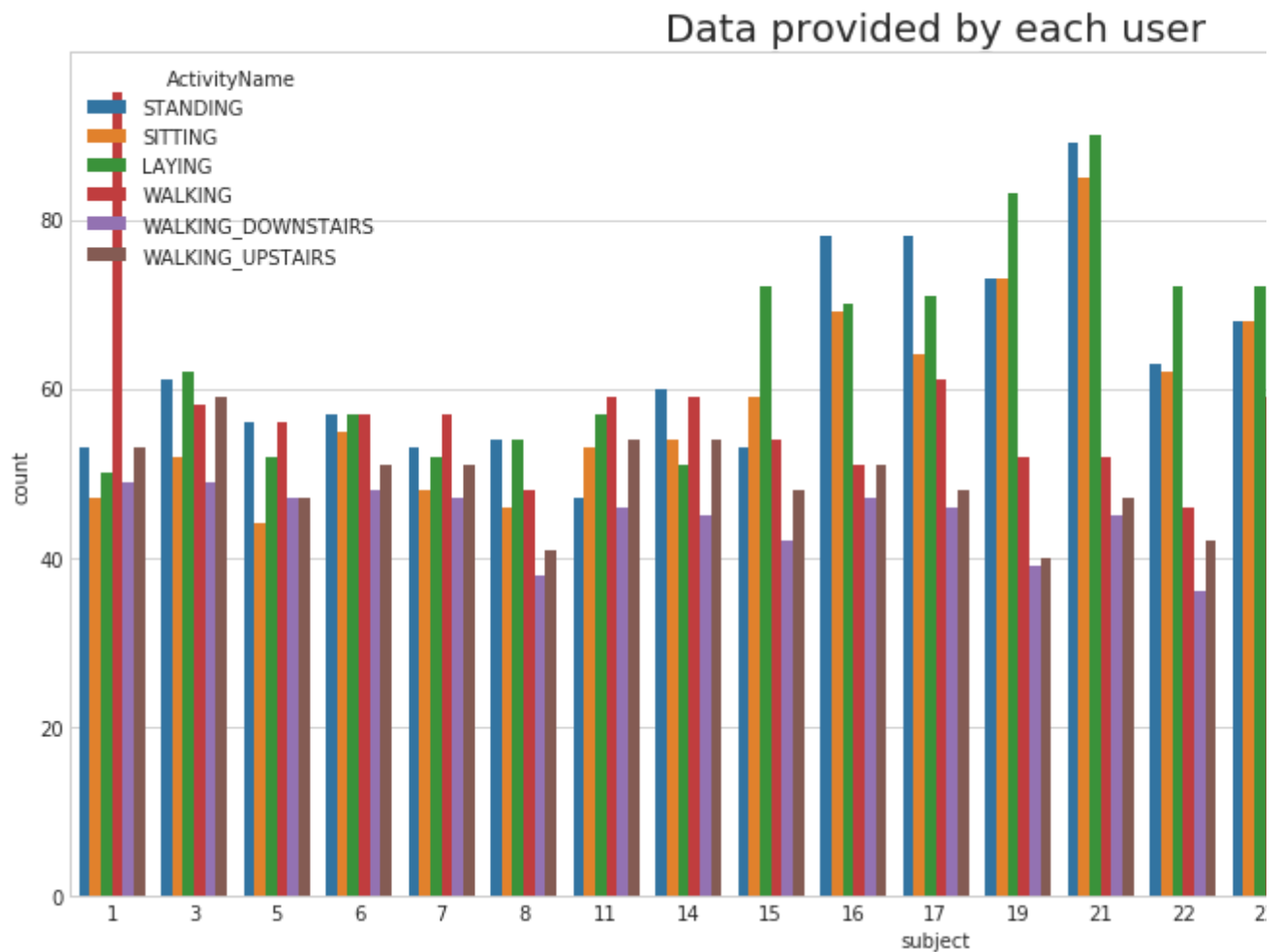
### ▼ 3. Check for data imbalance

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns

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

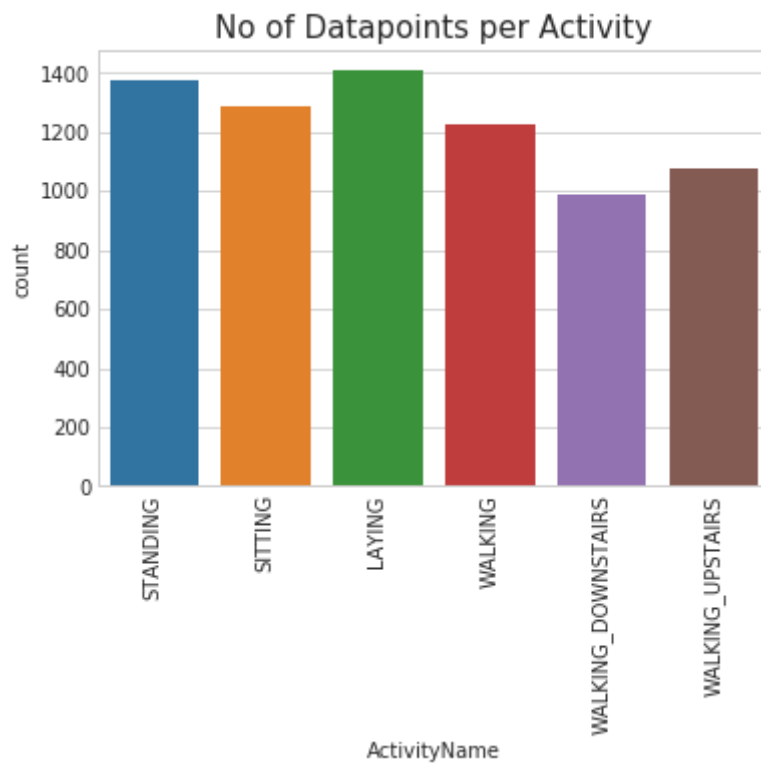
plt.figure(figsize=(16,8))
plt.title('Data provided by each user', fontsize=20)
sns.countplot(x='subject', hue='ActivityName', data = train)
plt.show()
```



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

```
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

```
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
```

```
Index(['tBodyAccmeanX', 'tBodyAccmeanY', 'tBodyAccmeanZ', 'tBodyAccstdX',
      'tBodyAccstdY', 'tBodyAccstdZ', 'tBodyAccmadX', 'tBodyAccmadY',
      'tBodyAccmadZ', 'tBodyAccmaxX',
      ...,
      'angleBodyAccMeangravity', 'angleBodyAccJerkMeangravityMean',
      'angleBodyGyroMeangravityMean', 'angleBodyGyroJerkMeangravityMean',
      'angleXgravityMean', 'angleYgravityMean', 'angleZgravityMean',
      'subject', 'Activity', 'ActivityName'],
      dtype='object', length=564)
```



## ▼ 5. Save this dataframe in a csv files

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

Double-click (or enter) to edit

## ▼ Exploratory Data Analysis

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

### ▶ 1. Featuring Engineering from Domain Knowledge

↳ 1 cell hidden

### ▶ 2. Stationary and Moving activities are completely different

↳ 2 cells hidden

### ▶ 3. Magnitude of an acceleration can saperate it well

↳ 2 cells hidden

### ▶ 4. Position of GravityAccelerationComponants also matters

↳ 6 cells hidden

## ▼ Apply t-sne on the data

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

# 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(
            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')
```

```
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 i

[t-SNE] Iteration 100: error = 107.2019501, gradient norm = 0.0284782 (50 iterations

[t-SNE] Iteration 150: error = 100.9872894, gradient norm = 0.0185151 (50 iterations

[t-SNE] Iteration 200: error = 97.6054382, gradient norm = 0.0142084 (50 iterations i

[t-SNE] Iteration 250: error = 95.3084183, gradient norm = 0.0132592 (50 iterations i

[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

[t-SNE] Iteration 350: error = 3.2113254, gradient norm = 0.0009953 (50 iterations in

[t-SNE] Iteration 400: error = 2.7819963, gradient norm = 0.0007203 (50 iterations in

[t-SNE] Iteration 450: error = 2.5178111, gradient norm = 0.0005655 (50 iterations in

[t-SNE] Iteration 500: error = 2.3341548, gradient norm = 0.0004804 (50 iterations in

[t-SNE] Iteration 550: error = 2.1961622, gradient norm = 0.0004183 (50 iterations in

[t-SNE] Iteration 600: error = 2.0867445, gradient norm = 0.0003664 (50 iterations in

[t-SNE] Iteration 650: error = 1.9967778, gradient norm = 0.0003279 (50 iterations in

[t-SNE] Iteration 700: error = 1.9210005, gradient norm = 0.0002984 (50 iterations in

[t-SNE] Iteration 750: error = 1.8558111, gradient norm = 0.0002776 (50 iterations in

[t-SNE] Iteration 800: error = 1.7989457, gradient norm = 0.0002569 (50 iterations in

[t-SNE] Iteration 850: error = 1.7490212, gradient norm = 0.0002394 (50 iterations in

[t-SNE] Iteration 900: error = 1.7043383, gradient norm = 0.0002224 (50 iterations in

[t-SNE] Iteration 950: error = 1.6641431, gradient norm = 0.0002098 (50 iterations in

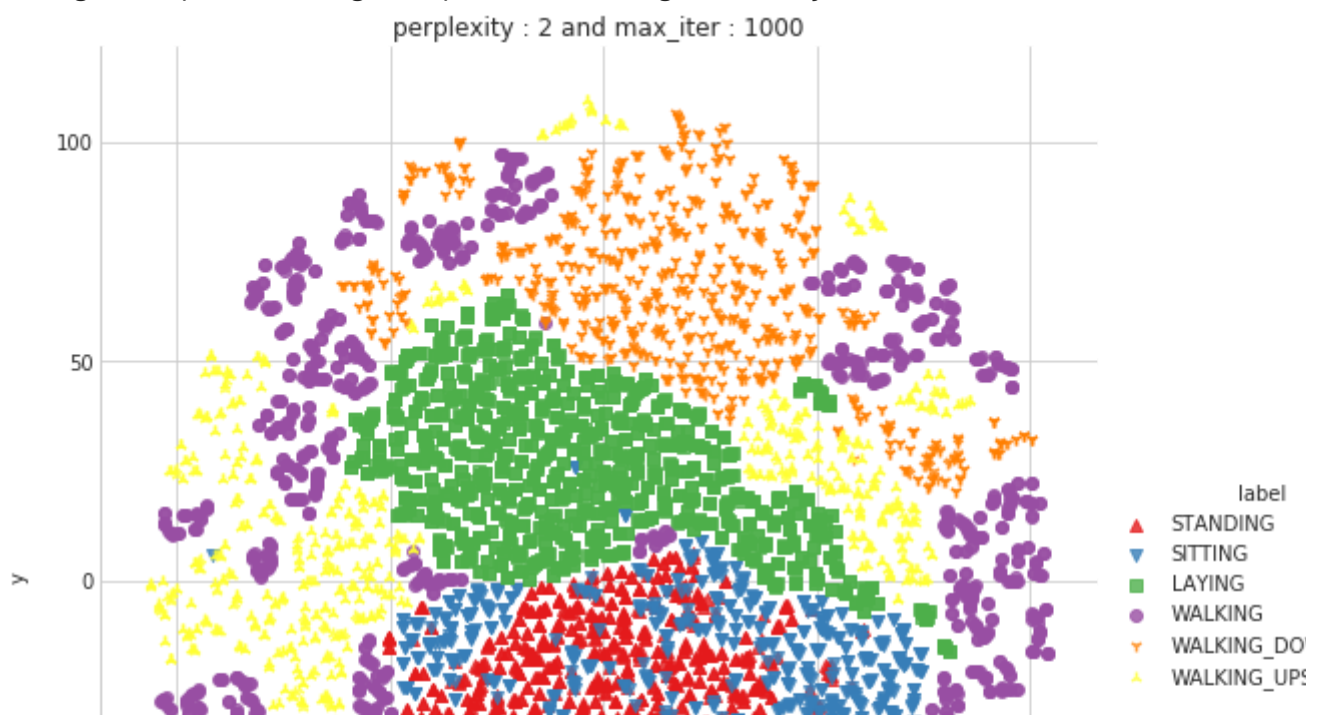
[t-SNE] Iteration 1000: error = 1.6279151, gradient norm = 0.0001989 (50 iterations i

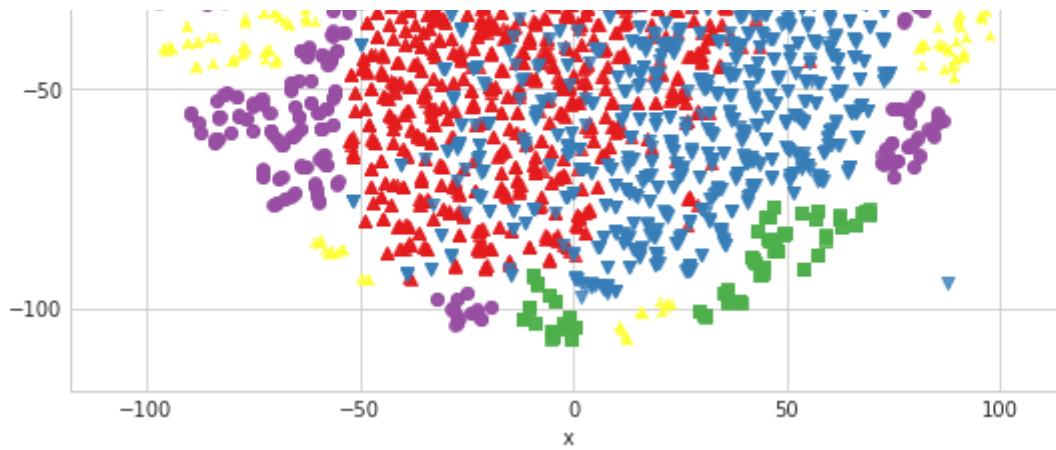
[t-SNE] Error after 1000 iterations: 1.627915

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 i

[t-SNE] Iteration 100: error = 97.6535568, gradient norm = 0.0174309 (50 iterations i

[t-SNE] Iteration 150: error = 93.1900101, gradient norm = 0.0101048 (50 iterations i

[t-SNE] Iteration 200: error = 91.2315445, gradient norm = 0.0074560 (50 iterations i

[t-SNE] Iteration 250: error = 90.0714417, gradient norm = 0.0057667 (50 iterations i

[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

[t-SNE] Iteration 350: error = 2.8173938, gradient norm = 0.0007508 (50 iterations in

[t-SNE] Iteration 400: error = 2.4344938, gradient norm = 0.0005251 (50 iterations in

[t-SNE] Iteration 450: error = 2.2156141, gradient norm = 0.0004069 (50 iterations in

[t-SNE] Iteration 500: error = 2.0703306, gradient norm = 0.0003340 (50 iterations in

[t-SNE] Iteration 550: error = 1.9646366, gradient norm = 0.0002816 (50 iterations in

[t-SNE] Iteration 600: error = 1.8835558, gradient norm = 0.0002471 (50 iterations in

[t-SNE] Iteration 650: error = 1.8184001, gradient norm = 0.0002184 (50 iterations in

[t-SNE] Iteration 700: error = 1.7647167, gradient norm = 0.0001961 (50 iterations in

[t-SNE] Iteration 750: error = 1.7193680, gradient norm = 0.0001796 (50 iterations in

[t-SNE] Iteration 800: error = 1.6803776, gradient norm = 0.0001655 (50 iterations in

[t-SNE] Iteration 850: error = 1.6465144, gradient norm = 0.0001538 (50 iterations in

[t-SNE] Iteration 900: error = 1.6166563, gradient norm = 0.0001421 (50 iterations in

[t-SNE] Iteration 950: error = 1.5901035, gradient norm = 0.0001335 (50 iterations in

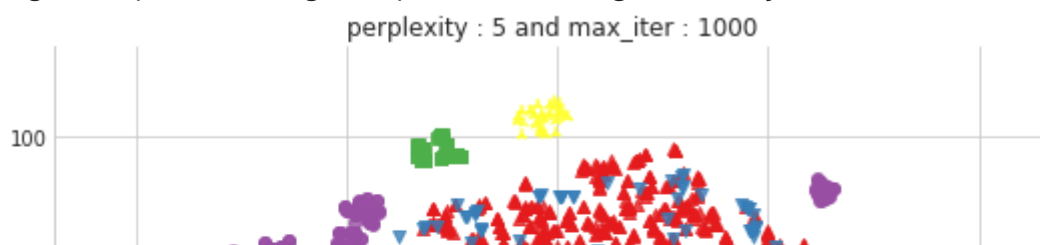
[t-SNE] Iteration 1000: error = 1.5664237, gradient norm = 0.0001257 (50 iterations i

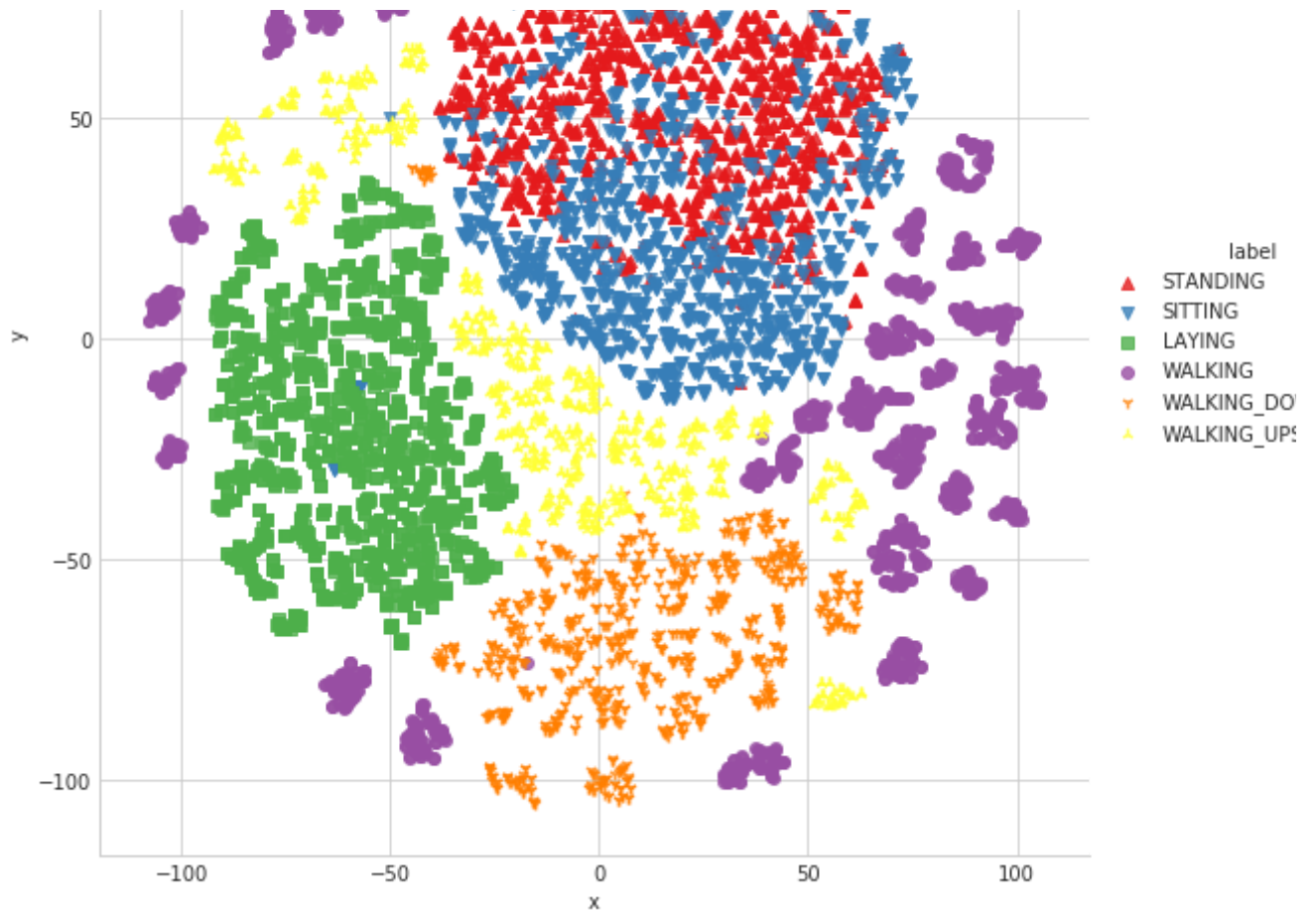
[t-SNE] Error after 1000 iterations: 1.566424

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 i

[t-SNE] Iteration 100: error = 90.3036194, gradient norm = 0.0097653 (50 iterations i

[t-SNE] Iteration 150: error = 87.3132935, gradient norm = 0.0053059 (50 iterations i

[t-SNE] Iteration 200: error = 86.1169128, gradient norm = 0.0035844 (50 iterations i

[t-SNE] Iteration 250: error = 85.4133606, gradient norm = 0.0029100 (50 iterations i

[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

[t-SNE] Iteration 350: error = 2.4929206, gradient norm = 0.0006466 (50 iterations in

[t-SNE] Iteration 400: error = 2.1733041, gradient norm = 0.0004230 (50 iterations in

[t-SNE] Iteration 450: error = 1.9884514, gradient norm = 0.0003124 (50 iterations in

[t-SNE] Iteration 500: error = 1.8702440, gradient norm = 0.0002514 (50 iterations in

[t-SNE] Iteration 550: error = 1.7870129, gradient norm = 0.0002107 (50 iterations in

[t-SNE] Iteration 600: error = 1.7246909, gradient norm = 0.0001824 (50 iterations in

[t-SNE] Iteration 650: error = 1.6758548, gradient norm = 0.0001590 (50 iterations in

[t-SNE] Iteration 700: error = 1.6361949, gradient norm = 0.0001451 (50 iterations in

[t-SNE] Iteration 750: error = 1.6034756, gradient norm = 0.0001305 (50 iterations in

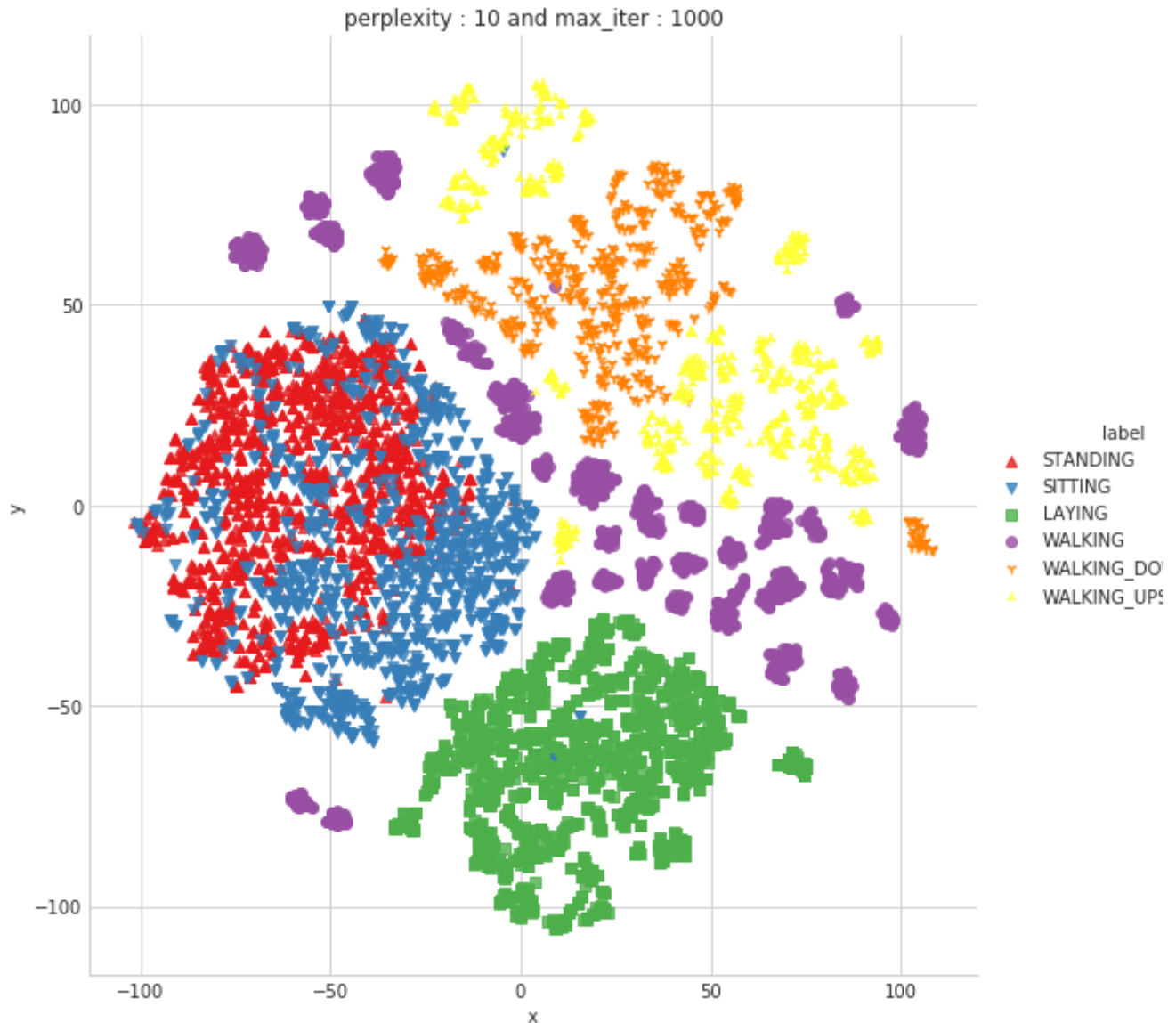
[t-SNE] Iteration 800: error = 1.5761518, gradient norm = 0.0001188 (50 iterations in

[t-SNE] Iteration 850: error = 1.5527289, gradient norm = 0.0001113 (50 iterations in

[t-SNE] Iteration 900: error = 1.5328671, gradient norm = 0.0001021 (50 iterations in



```
[t-SNE] Iteration 900: error = 1.5526671, gradient norm = 0.0001021 (50 iterations in
[t-SNE] Iteration 950: error = 1.5152045, gradient norm = 0.0000974 (50 iterations in
[t-SNE] Iteration 1000: error = 1.4999681, gradient norm = 0.0000933 (50 iterations i
[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
[t-SNE] Iteration 100: error = 83.9500732, gradient norm = 0.0059110 (50 iterations i
[t-SNE] Iteration 150: error = 81.8804779, gradient norm = 0.0035797 (50 iterations i
[t-SNE] Iteration 200: error = 81.1615143, gradient norm = 0.0022536 (50 iterations i
[t-SNE] Iteration 250: error = 80.7704086, gradient norm = 0.0018108 (50 iterations i
```

```
[t-SNE] Iteration 250: error = 80.770409, gradient norm = 0.0012993 (50 iterations in
[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
[t-SNE] Iteration 350: error = 2.1637220, gradient norm = 0.0005765 (50 iterations in
[t-SNE] Iteration 400: error = 1.9143614, gradient norm = 0.0003474 (50 iterations in
[t-SNE] Iteration 450: error = 1.7684202, gradient norm = 0.0002458 (50 iterations in
[t-SNE] Iteration 500: error = 1.6744757, gradient norm = 0.0001923 (50 iterations in
[t-SNE] Iteration 550: error = 1.6101606, gradient norm = 0.0001575 (50 iterations in
[t-SNE] Iteration 600: error = 1.5641028, gradient norm = 0.0001344 (50 iterations in
[t-SNE] Iteration 650: error = 1.5291905, gradient norm = 0.0001182 (50 iterations in
[t-SNE] Iteration 700: error = 1.5024391, gradient norm = 0.0001055 (50 iterations in
[t-SNE] Iteration 750: error = 1.4809053, gradient norm = 0.0000965 (50 iterations in
[t-SNE] Iteration 800: error = 1.4631859, gradient norm = 0.0000884 (50 iterations in
[t-SNE] Iteration 850: error = 1.4486470, gradient norm = 0.0000832 (50 iterations in
[t-SNE] Iteration 900: error = 1.4367288, gradient norm = 0.0000804 (50 iterations in
[t-SNE] Iteration 950: error = 1.4270191, gradient norm = 0.0000761 (50 iterations in
[t-SNE] Iteration 1000: error = 1.4189968, gradient norm = 0.0000787 (50 iterations in
[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
```

```

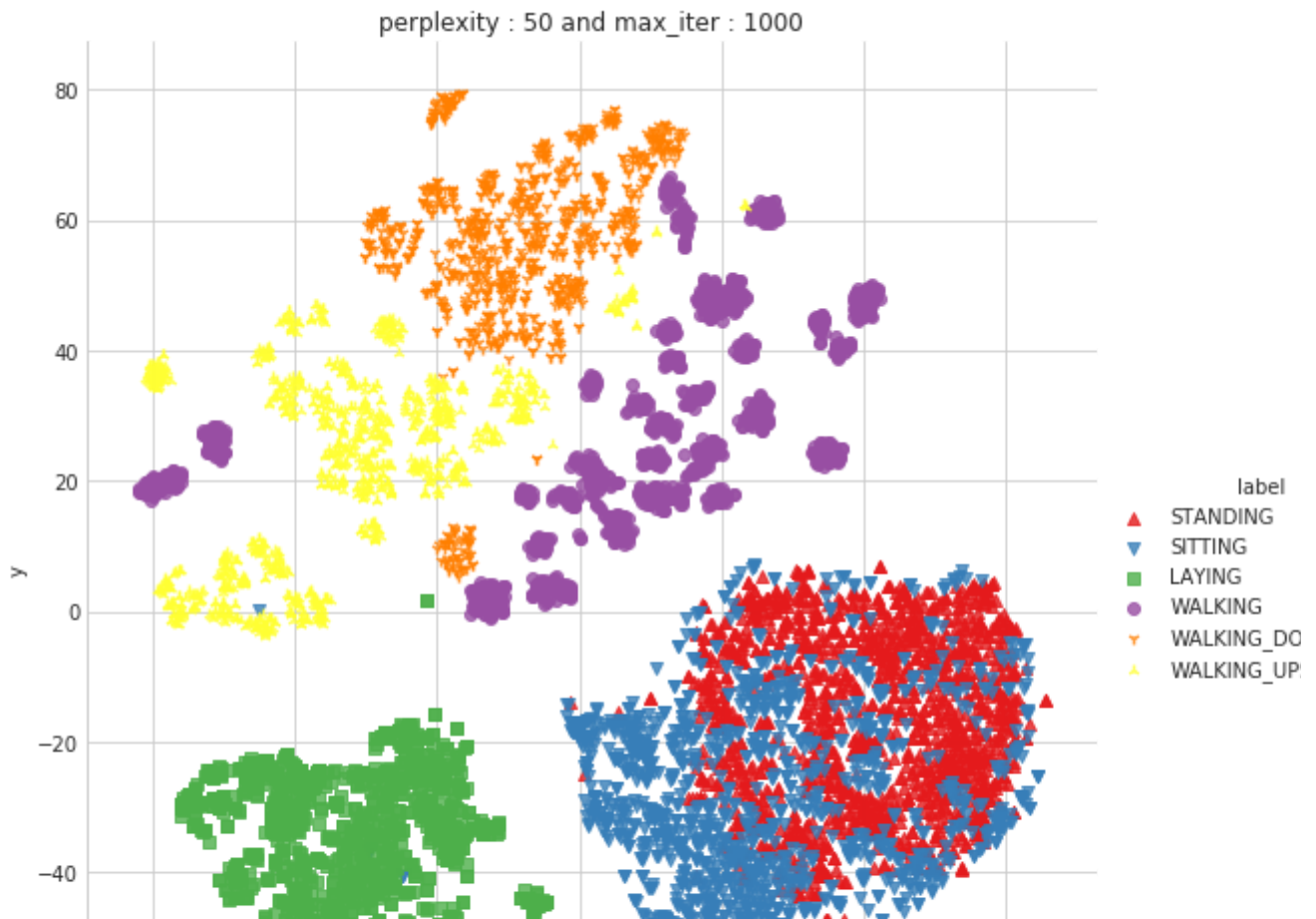
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities 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
[t-SNE] Iteration 100: error = 75.9874649, gradient norm = 0.0061005 (50 iterations i
[t-SNE] Iteration 150: error = 74.7072296, gradient norm = 0.0024708 (50 iterations i
[t-SNE] Iteration 200: error = 74.2736282, gradient norm = 0.0018644 (50 iterations i
[t-SNE] Iteration 250: error = 74.0722427, gradient norm = 0.0014078 (50 iterations i
[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
[t-SNE] Iteration 350: error = 1.7567128, gradient norm = 0.0004845 (50 iterations in
[t-SNE] Iteration 400: error = 1.5888531, gradient norm = 0.0002798 (50 iterations in
[t-SNE] Iteration 450: error = 1.4956820, gradient norm = 0.0001894 (50 iterations in
[t-SNE] Iteration 500: error = 1.4359720, gradient norm = 0.0001420 (50 iterations in
[t-SNE] Iteration 550: error = 1.3947564, gradient norm = 0.0001117 (50 iterations in
[t-SNE] Iteration 600: error = 1.3653858, gradient norm = 0.0000949 (50 iterations in
[t-SNE] Iteration 650: error = 1.3441534, gradient norm = 0.0000814 (50 iterations in
[t-SNE] Iteration 700: error = 1.3284039, gradient norm = 0.0000742 (50 iterations in
[t-SNE] Iteration 750: error = 1.3171139, gradient norm = 0.0000700 (50 iterations in
[t-SNE] Iteration 800: error = 1.3085558, gradient norm = 0.0000657 (50 iterations in
[t-SNE] Iteration 850: error = 1.3017821, gradient norm = 0.0000603 (50 iterations in
[t-SNE] Iteration 900: error = 1.2962619, gradient norm = 0.0000586 (50 iterations in
[t-SNE] Iteration 950: error = 1.2914882, gradient norm = 0.0000573 (50 iterations in
[t-SNE] Iteration 1000: error = 1.2874244, gradient norm = 0.0000546 (50 iterations i
[t-SNE] Error after 1000 iterations: 1.287424

```

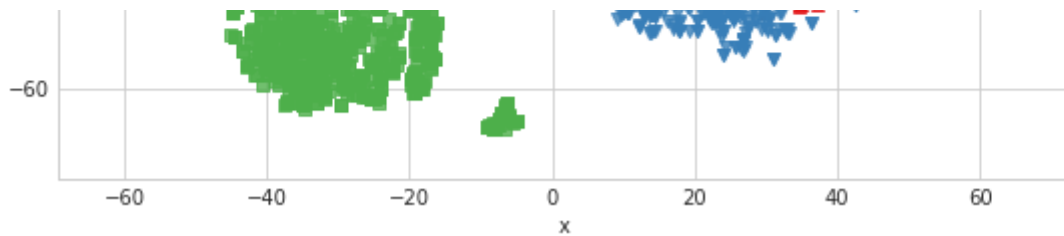
Done..

Creating plot for this t-sne visualization..

saving this plot as image in present working directory...








Done


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

## ▼ Obtain the train and test data

```
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)

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




	tBodyAccmeanX	tBodyAccmeanY	tBodyAccmeanZ	tBodyAccstdX	tBodyAccstdY	tBodyAccstdZ
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913111
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960300
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978187

3 rows × 564 columns

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

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

```
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,))

Double-click (or enter) to edit

Double-click (or enter) to edit

## ▼ Let's model with our data

### ▼ Labels that are useful in plotting confusion matrix

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

### ▼ Function to plot the confusion matrix

```
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

```
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) - {}'.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) - {}'.format(results['testing_time']))
results['predicted'] = y_pred

# calculate overall accuracy 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      {}'.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 confusion matrix
plt.figure(figsize=(8,8))
plt.grid(b=False)
plot_confusion_matrix(cm, classes=class_labels, normalize=True, title='Normalized conf
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

```

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 numbere 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.

```

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

## ▼ 1. Logistic Regression with Grid Search

```
from sklearn import linear_model
from sklearn import metrics

from sklearn.model_selection import GridSearchCV

# 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, clas
```



```
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

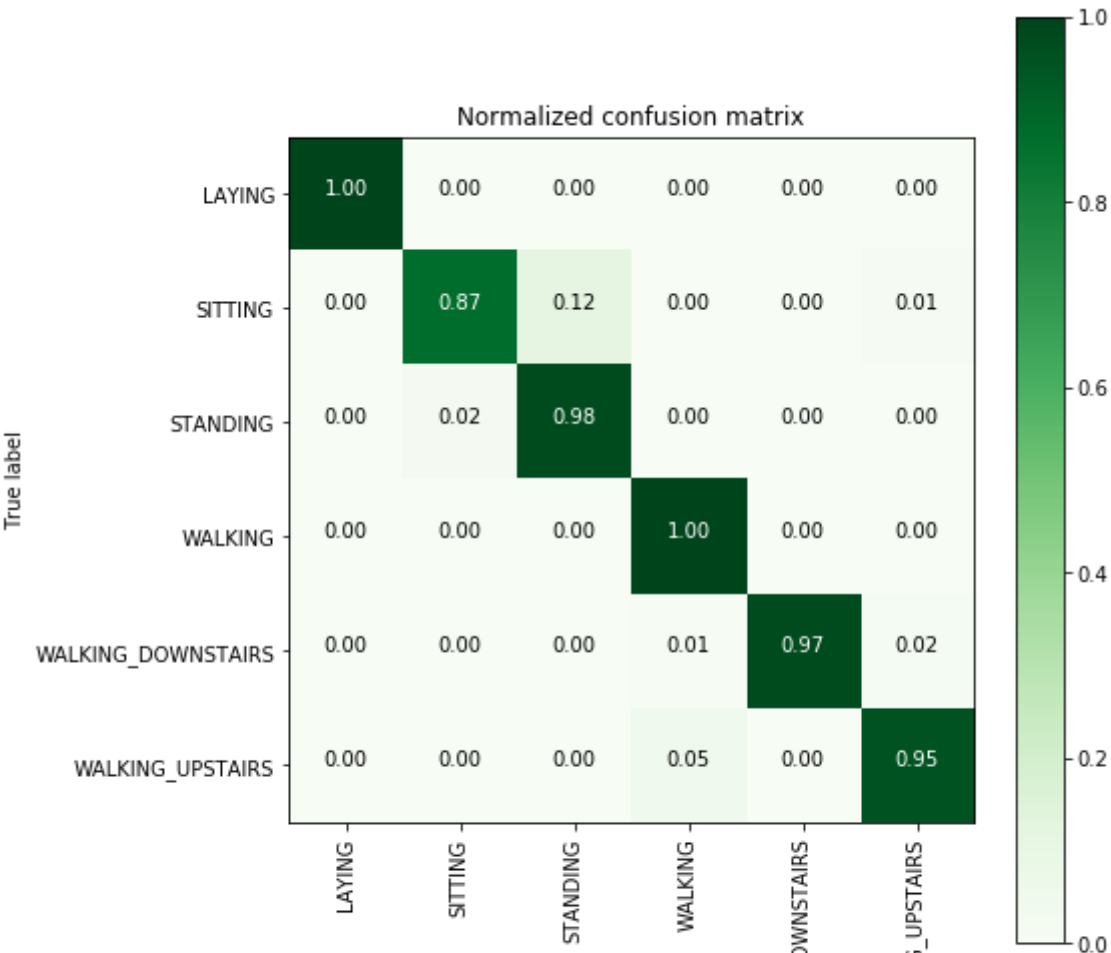
testing time(HH:MM:SS.ms) - 0:00:00.009192

-----  
Accuracy

0.9626739056667798

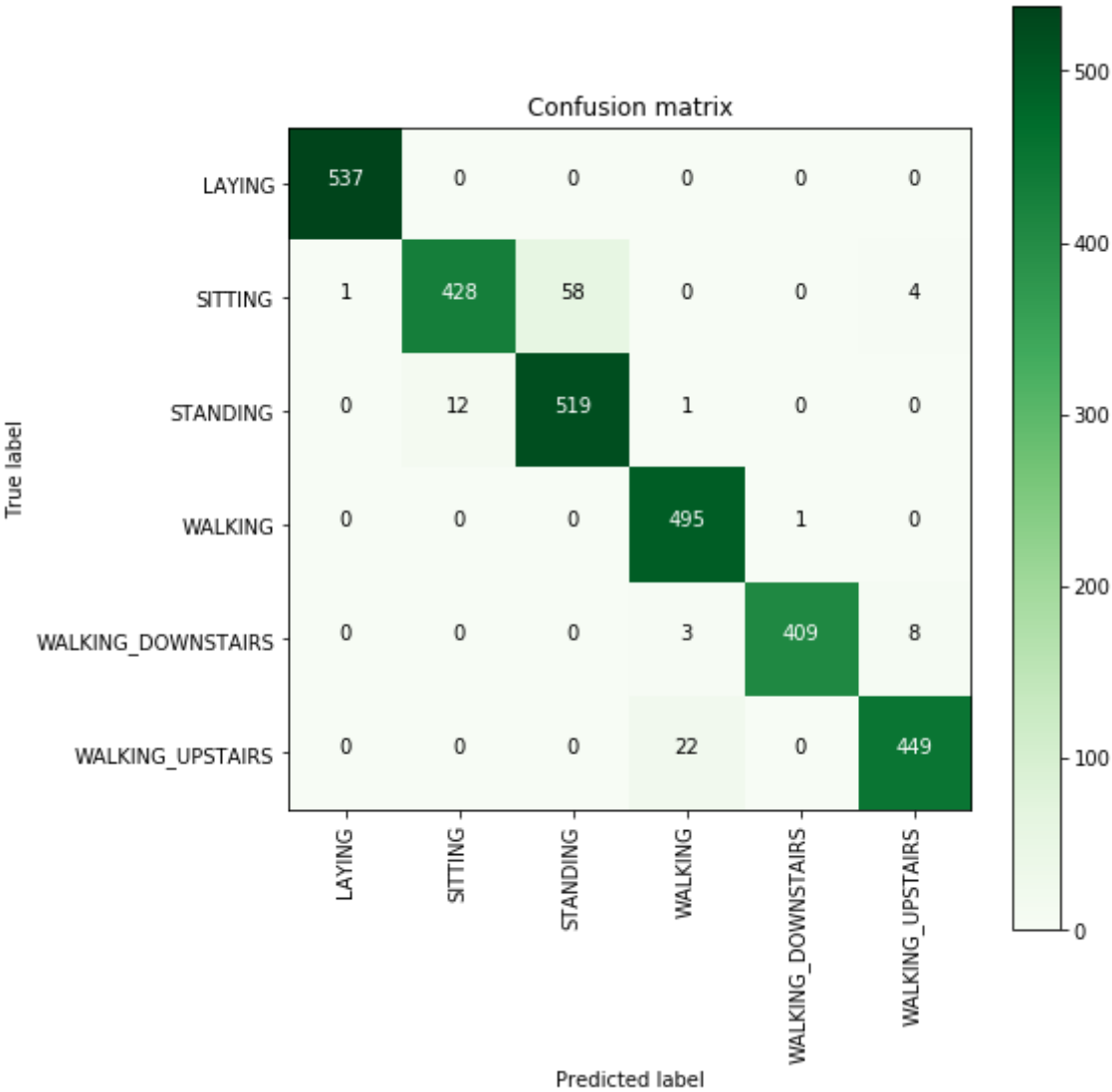
-----  
Confusion Matrix

```
[[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]]
```



		Predicted label			
		WALKING_DC	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
-----					
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	0.98	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

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



```
# 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 nombre of cross validation sets: 3
```

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

```
Average Cross Validate scores of best estimator :
```

```
0.9461371055495104
```

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

## ▼ 2. Linear SVC with GridSearch

```
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_1
```





```
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
```

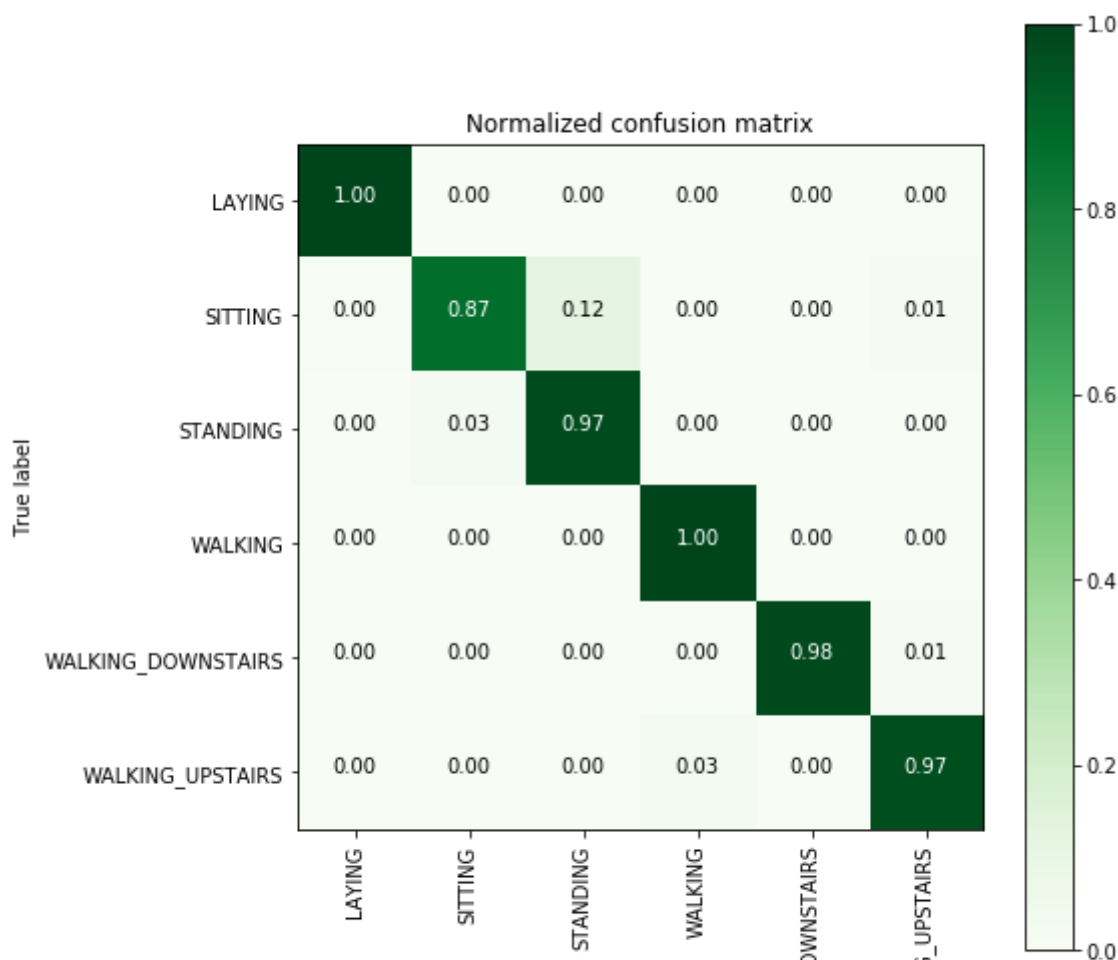
training\_time(HH:MM:SS.ms) - 0:00:32.951942

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]]
```



Predicted label					
-----					
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	

```
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

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



training the model..  
Done

training\_time(HH:MM:SS.ms) - 0:05:46.182889

Predicting test data  
Done

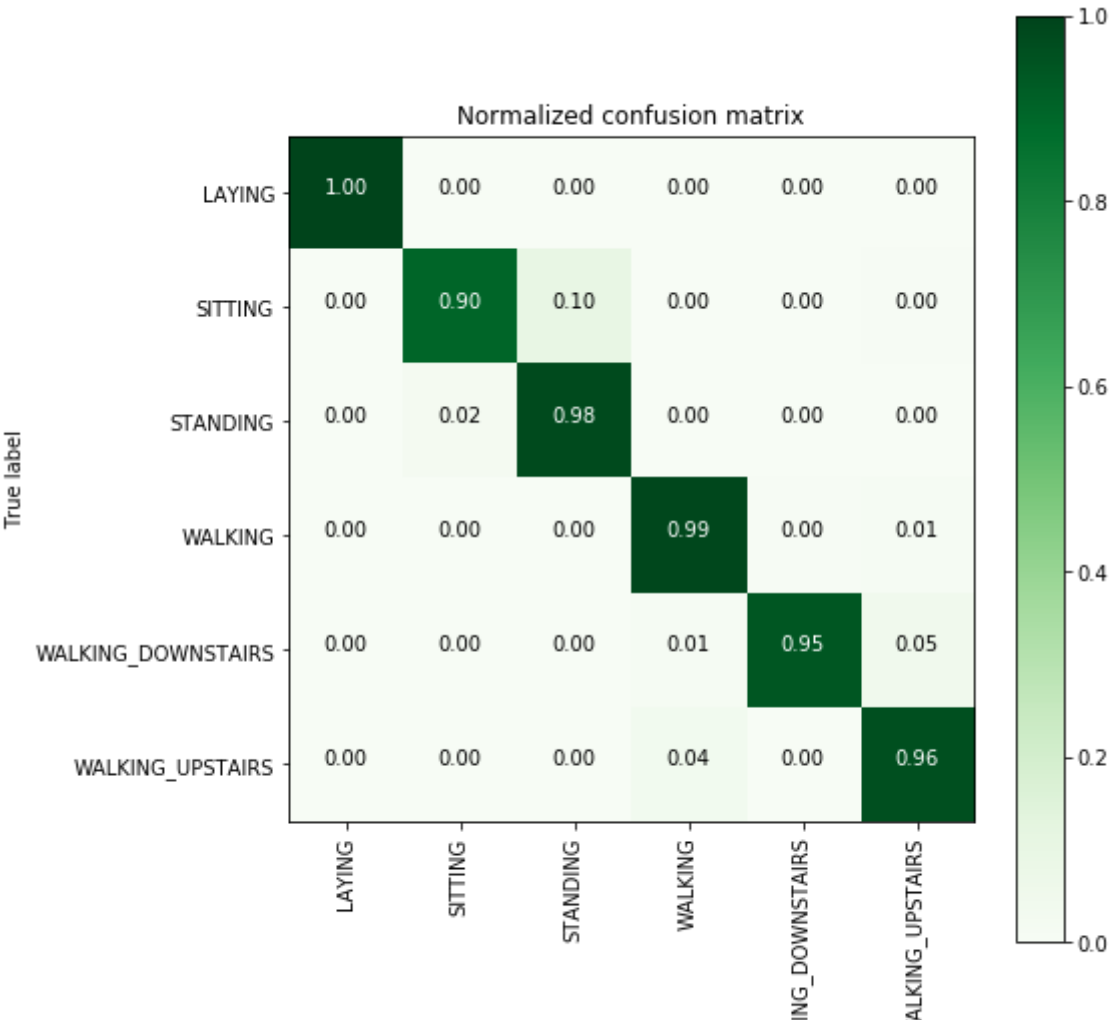
testing time(HH:MM:SS.ms) - 0:00:05.221285

-----  
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]]
```



WALK

W

Predicted label

```
-----
| 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

```
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 numbere of cross validation sets: 3
```

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

```
Average Cross Validate scores of best estimator :
```

```
0.9440968443960827
```

## ➤ 4. Decision Trees with GridSearchCV

```
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=la
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

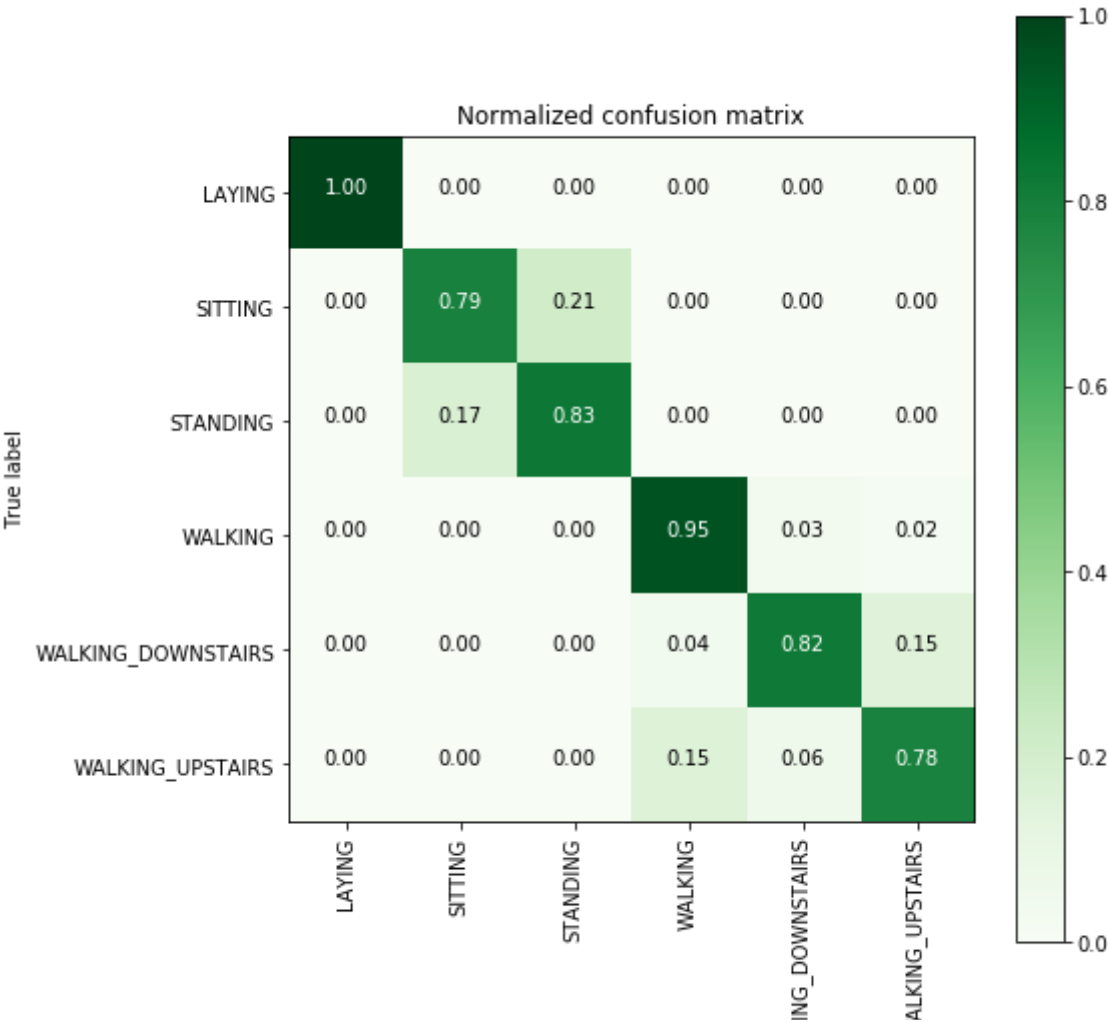
testing time(HH:MM:SS.ms) - 0:00:00.012858

-----  
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]]
```



Predicted label

```
-----
| Classification Report |
-----
```

	precision	recall	f1-score	support
LAYING	1.00	1.00	1.00	537
SITTING	0.81	0.79	0.80	491
STANDING	0.81	0.83	0.82	532
WALKING	0.84	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

```
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=
https://colab.research.google.com/drive/1-RY01hV1Dr8FEDDyu53iOnjYUhtlDy-G#scrollTo=Ko580w4gUx9d&printMode=true
```



```
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

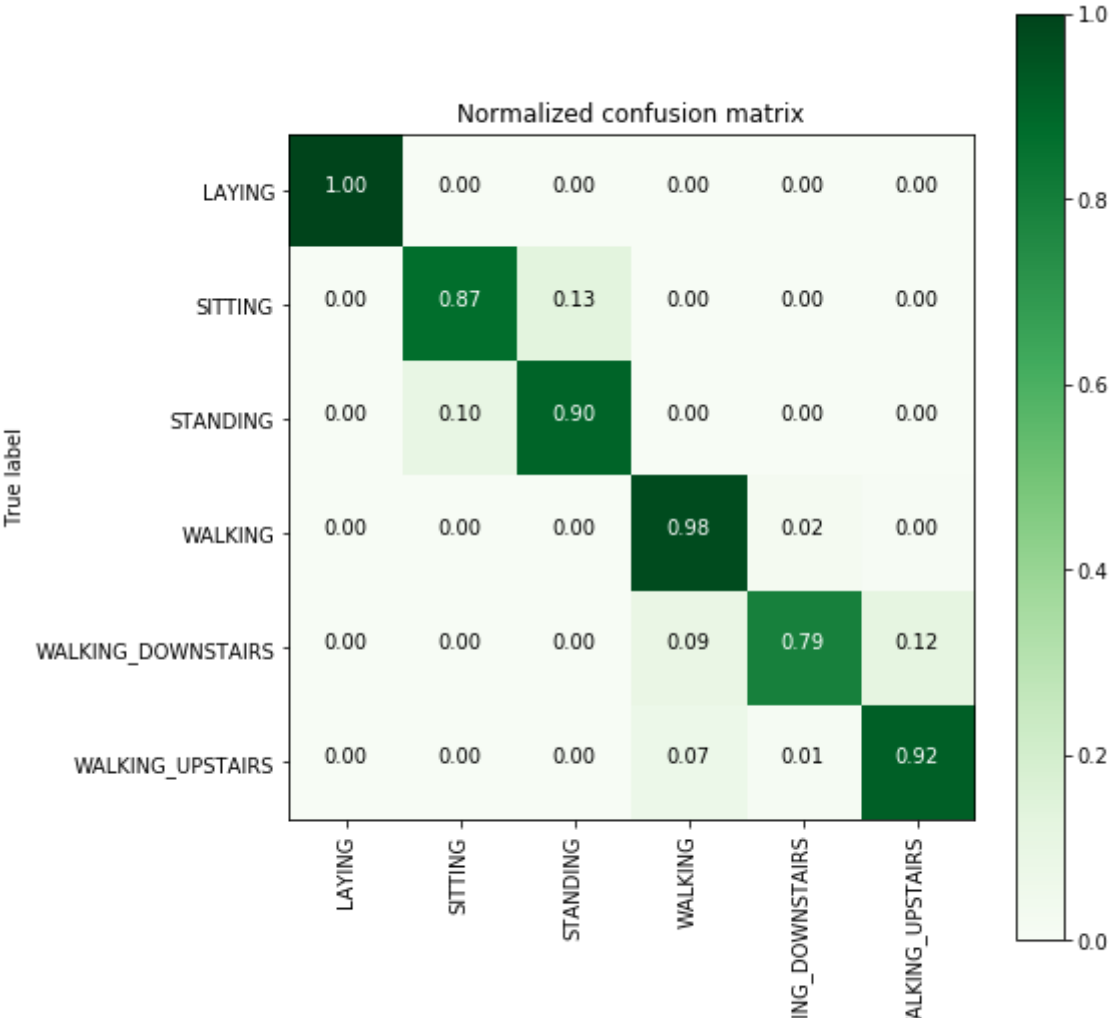
testing time(HH:MM:SS.ms) - 0:00:00.025937

-----  
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]]
```



WALK

W

Predicted label

```
-----
| Classification 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

```
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_label)
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

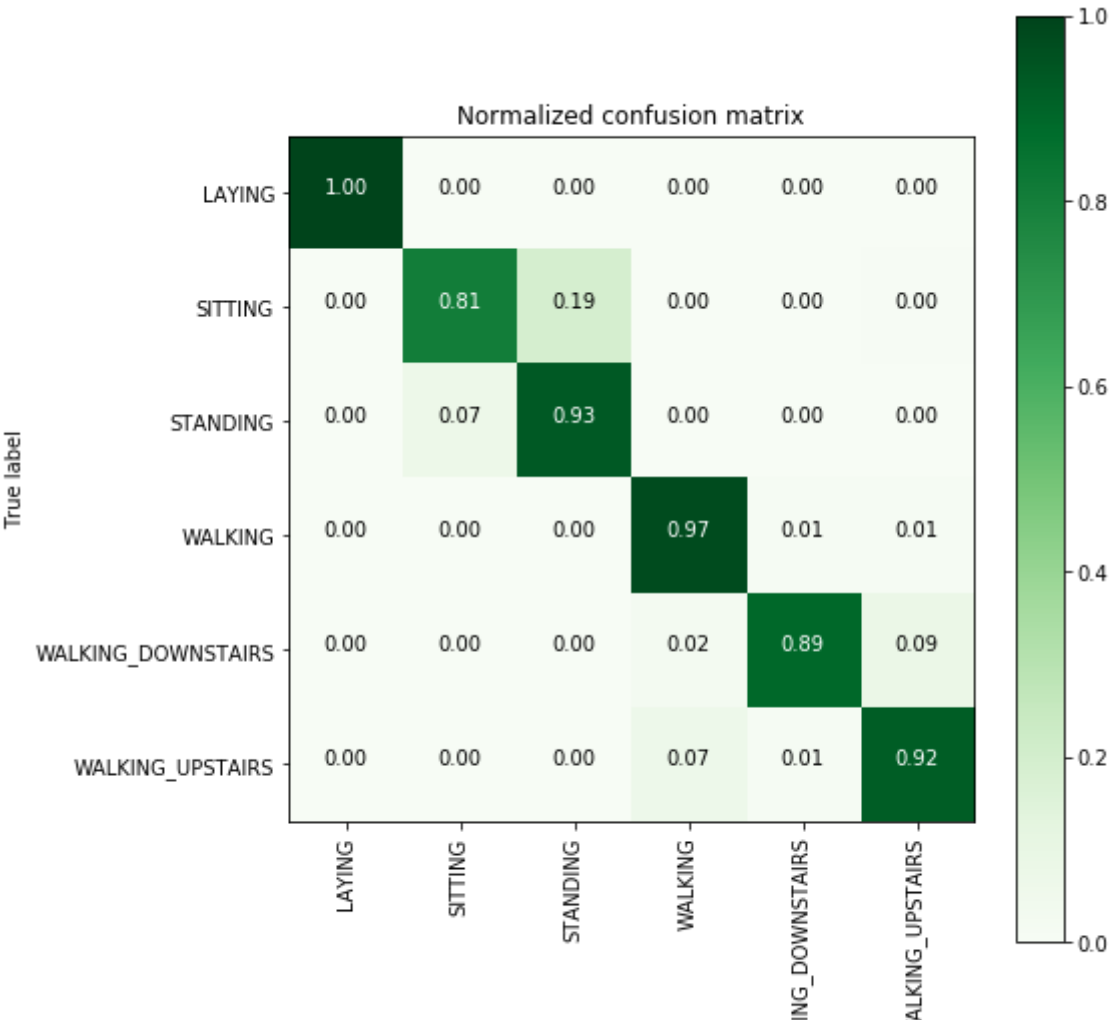
testing time(HH:MM:SS.ms) - 0:00:00.058843

-----  
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]]
```



WALK

W

Predicted label

```
-----
| Classification 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 nombre of cross validation sets: 3
```

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

```
Average Cross Validate scores of best estimator :

0.904379760609358
```

## ▼ 7. Comparing all models

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

```

100-(log_reg_grid_results['accuracy'] *
print('Linear SVC          : {:.04}%          {:.04}% '.format(lr_svc_grid_results['accuracy']
100-(lr_svc_grid_results['accuracy']

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

print('DecisionTree       : {:.04}%          {:.04}% '.format(dt_grid_results['accuracy'] * 1
100-(dt_grid_results['accuracy'] *

print('Random Forest      : {:.04}%          {:.04}% '.format(rfc_grid_results['accuracy'] *
100-(rfc_grid_results['accuracy']

print('GradientBoosting DT : {:.04}%          {:.04}% '.format(rfc_grid_results['accuracy'] *
100-(rfc_grid_results['accuracy']

```



	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.

```

import pandas as pd
import numpy as np

```

```

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

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

```

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

## ▼ Data

```
# Data directory
DATADIR = '/content/drive/My Drive/HAR/UCI_HAR_Dataset'

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

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

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

    for signal in SIGNALS:
        filename = f'/content/drive/My Drive/HAR/UCI_HAR_Dataset/{subset}/Inertial Signals
        signals_data.append(
            _read_csv(filename).as_matrix()
        )

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

def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get\_dummies.html)
    """
```



```

filename = f'/content/drive/My Drive/HAR/UCI_HAR_Dataset/{subset}/y_{subset}.txt'
y = _read_csv(filename)[0]

return pd.get_dummies(y).as_matrix()

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

    return X_train, X_test, y_train, y_test

```

## ▼ Model 1-Single hidden layer(LSTM)

```

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

```

🔗 The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.  
We recommend you [upgrade](#) now or ensure your notebook will continue to use TensorFlow 1.x via the %te

```

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

```

```

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

```

🔗 Using TensorFlow backend.

```

# Importing libraries
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout

```

```

# Initializing parameters

```

```

epochs = 30
batch_size = 32
n_hidden = 128

```

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

```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

```
↳ /usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: FutureWarning: Metho
    # This is added back by InteractiveShellApp.init_path()
    /usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:12: FutureWarning: Metho
        if sys.path[0] == '':
```

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

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

```
↳ 128
    9
    7352
```

- Defining the Architecture of LSTM

```
# Initiliazing the sequential model
model1 = Sequential()
# Configuring the parameters
model1.add(LSTM(n_hidden, input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model1.add(Dropout(0.25))
# Adding a dense output layer with sigmoid activation
model1.add(Dense(n_classes, activation='sigmoid'))
model1.summary()
```

```
↳
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 128)	70656
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 6)	774
Total params: 71,430		
Trainable params: 71,430		
Non-trainable params: 0		

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

```
⏏ WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:79
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
```

```
# Training the model
model1.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 [=====] - 56s 8ms/step - loss: 0.1188 - acc: 0.954

Epoch 2/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1115 - acc: 0.956

Epoch 3/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1355 - acc: 0.949

Epoch 4/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1243 - acc: 0.951

Epoch 5/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1191 - acc: 0.952

Epoch 6/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1191 - acc: 0.951

Epoch 7/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1165 - acc: 0.954

Epoch 8/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1244 - acc: 0.952

Epoch 9/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1250 - acc: 0.951

Epoch 10/30

7352/7352 [=====] - 55s 8ms/step - loss: 0.1109 - acc: 0.955

Epoch 11/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1114 - acc: 0.955

Epoch 12/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1073 - acc: 0.954

Epoch 13/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1043 - acc: 0.956

Epoch 14/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1114 - acc: 0.956

Epoch 15/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1070 - acc: 0.955

Epoch 16/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1001 - acc: 0.956

Epoch 17/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1215 - acc: 0.954

Epoch 18/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1065 - acc: 0.957

Epoch 19/30

7352/7352 [=====] - 57s 8ms/step - loss: 0.1185 - acc: 0.951

Epoch 20/30

7352/7352 [=====] - 57s 8ms/step - loss: 0.1072 - acc: 0.955

Epoch 21/30

7352/7352 [=====] - 57s 8ms/step - loss: 0.1011 - acc: 0.956

Epoch 22/30

7352/7352 [=====] - 57s 8ms/step - loss: 0.1138 - acc: 0.955

Epoch 23/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.0990 - acc: 0.958

Epoch 24/30

7352/7352 [=====] - 57s 8ms/step - loss: 0.1053 - acc: 0.958

Epoch 25/30

7352/7352 [=====] - 57s 8ms/step - loss: 0.0982 - acc: 0.956

Epoch 26/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1262 - acc: 0.955

Epoch 27/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1062 - acc: 0.957

Epoch 28/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1034 - acc: 0.958

Epoch 29/30

7352/7352 [=====] - 56s 8ms/step - loss: 0.1023 - acc: 0.954

Epoch 30/30

7352/7352 [=====] - 57s 8ms/step - loss: 0.0984 - acc: 0.957

<keras.callbacks.History at 0x7f718de9a6d8>

```
history1=model1.history
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
from sklearn.metrics import confusion_matrix
```

```
import itertools
```

```
#Utility function to plot the confusion matrices
```

```
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion\_matrix.html
```

```
def plot_confusion_matrix(cm_df, classes, normalize, title):
```

```
    if normalize:
```

```
        cm = cm_df.values
```

```
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
```

```
        plt.figure(figsize = (7,7))
```

```
        plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Greens)
```

```
        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"
```

```
        plt.tight_layout()
```

```
        plt.xlabel('Predicted labels')
```

```
        plt.ylabel('True labels')
```

```
    else:
```

```
        import seaborn as sn
```

```
        plt.figure(figsize = (6,5))
```

```
        ax = sn.heatmap(cm_df, annot=True, fmt='d', cmap=plt.cm.Blues) #fmt='d' for deci
```

```
        ax.set_xlabel("Predicted Labels")
```

```
        ax.set_ylabel("True Labels")
```

```
        ax.set_title(title)
```

```
#Utility function to design the confusion matrix DF
```

```
def get_confusion_matrix(Y_true, Y_pred):
```

```
    Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_true, axis=1)])
```

```
    Y_pred = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_pred, axis=1)])
```

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

```
    return cm_df
```

```
y_pred=model1.predict(X_test)
```

```
cm_df=get_confusion_matrix(Y_test, y_pred) #Prepare the confusion matrix by using get_conf
```

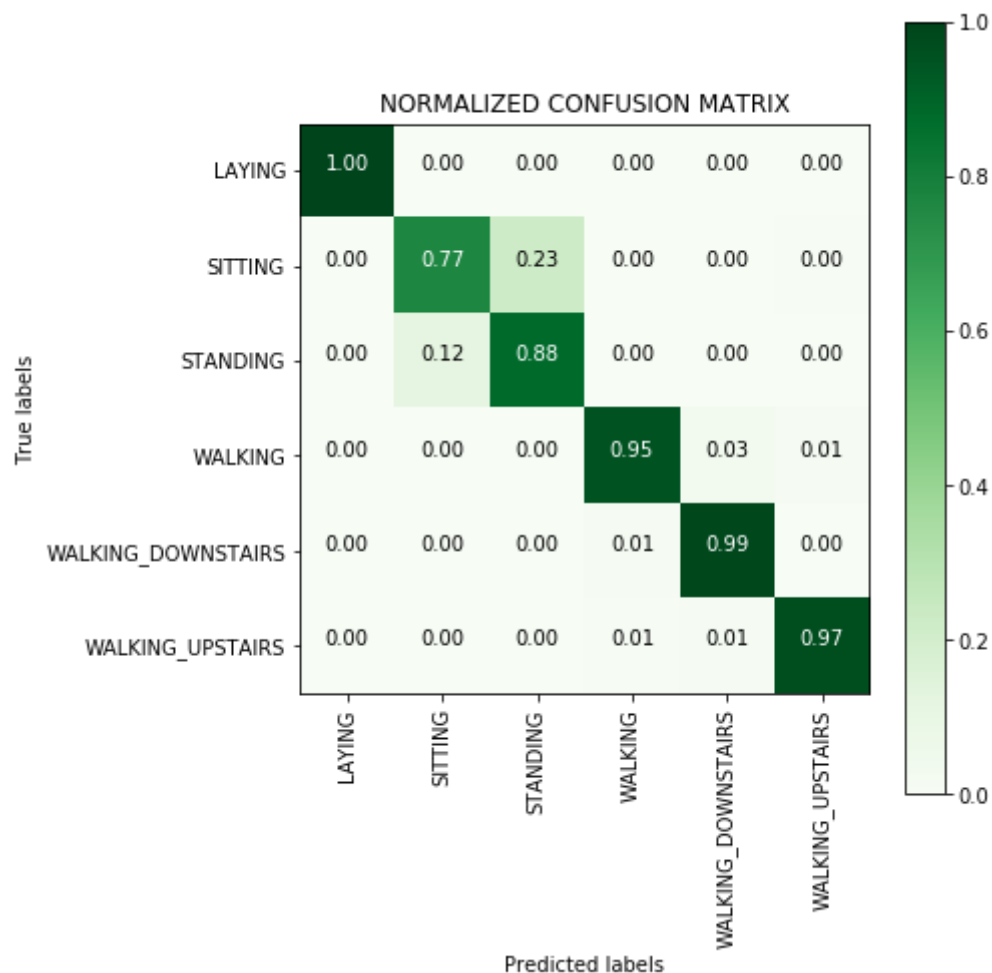
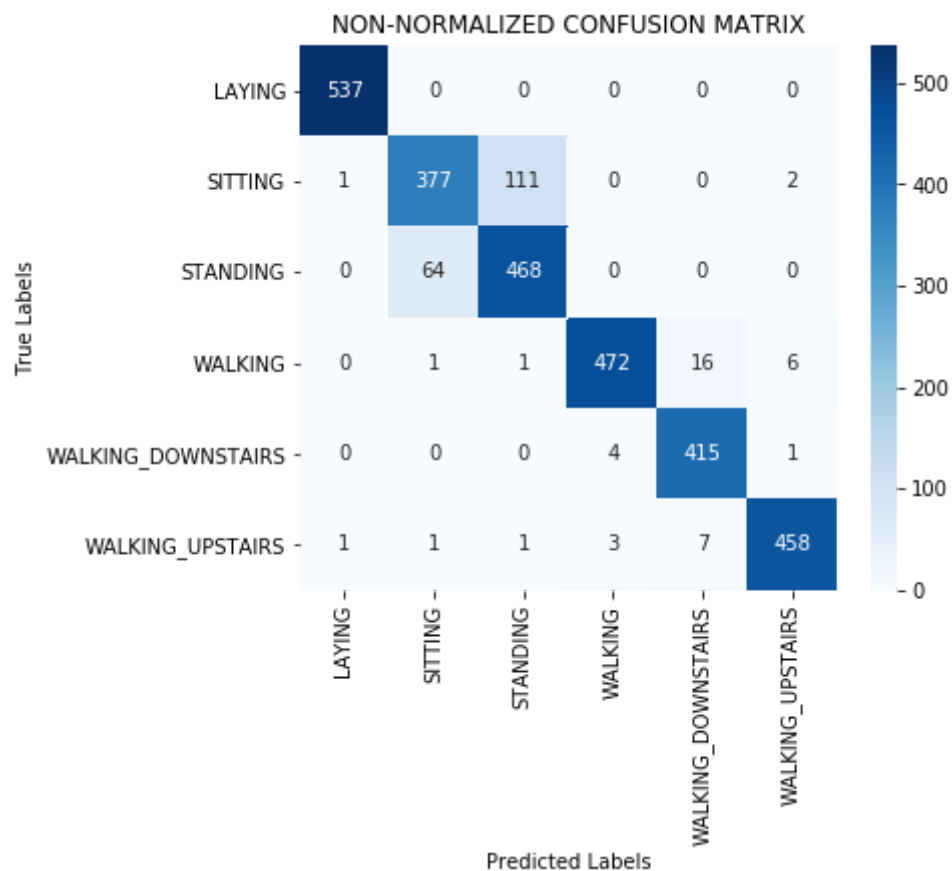
```
classes=list(cm_df.index) #Class names = Index Names or Column Names in cm_df
```

```
#Plot a Non-Normalized confusion matrix
```

```
plot_confusion_matrix(cm_df, classes, normalize=False, title="NON-NORMALIZED CONFUSION MAT
```

```
#Plot a Normalized confusion matrix
```

```
plot_confusion_matrix(cm_df, classes, normalize=True, title="NORMALIZED CONFUSION MATRIX")
```



```
score1 = model1.evaluate(X_test, Y_test)
```

```
↳ 2947/2947 [=====] - 8s 3ms/step
```

```
print('The Test loss is',score1[0],'and Test accuracy is',score1[1])
```

```
↳ The Test loss is 0.36151349165593255 and Test accuracy is 0.9253478113335596
```

## ▼ Model 2

Layer 1-128 lstm ,dropout=0.2

Layer 2-64 lstm ,dropout=0.5

```
epochs_1 = 30
```

```
batch_size_1= 32
```

```
n_hidden_1 = 128
```

```
n_hidden_2 =64
```

```
model1 = Sequential()
```

```
# Configuring the parameters
```

```
model1.add(LSTM(n_hidden_1, return_sequences=True, input_shape=(timesteps, input_dim)))
```

```
# Adding a dropout layer
```

```
model1.add(Dropout(0.2))
```

```
model1.add(LSTM(n_hidden_2))
```

```
# Adding a dropout layer
```

```
model1.add(Dropout(0.5))
```

```
# Adding a dense output layer with sigmoid activation
```

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

```
model1.summary()
```

```
↳ Model: "sequential_4"
```

Layer (type)	Output Shape	Param #
=====		
lstm_4 (LSTM)	(None, 128, 128)	70656
=====		
dropout_4 (Dropout)	(None, 128, 128)	0
=====		
lstm_5 (LSTM)	(None, 64)	49408
=====		
dropout_5 (Dropout)	(None, 64)	0
=====		
dense_4 (Dense)	(None, 6)	390
=====		
Total params: 120,454		
Trainable params: 120,454		
Non-trainable params: 0		
=====		

```
model1.compile(loss='categorical_crossentropy',
               optimizer='rmsprop',
               metrics=['accuracy'])
```

```
model1.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 [=====] - 118s 16ms/step - loss: 1.3030 - acc: 0.4

Epoch 2/30

7352/7352 [=====] - 117s 16ms/step - loss: 0.9816 - acc: 0.5

Epoch 3/30

7352/7352 [=====] - 115s 16ms/step - loss: 0.7327 - acc: 0.6

Epoch 4/30

7352/7352 [=====] - 114s 16ms/step - loss: 0.5335 - acc: 0.8

Epoch 5/30

7352/7352 [=====] - 114s 16ms/step - loss: 0.3052 - acc: 0.9

Epoch 6/30

7352/7352 [=====] - 116s 16ms/step - loss: 0.2487 - acc: 0.9

Epoch 7/30

7352/7352 [=====] - 116s 16ms/step - loss: 0.2270 - acc: 0.9

Epoch 8/30

7352/7352 [=====] - 114s 15ms/step - loss: 0.1966 - acc: 0.9

Epoch 9/30

7352/7352 [=====] - 113s 15ms/step - loss: 0.1915 - acc: 0.9

Epoch 10/30

7352/7352 [=====] - 113s 15ms/step - loss: 0.1797 - acc: 0.9

Epoch 11/30

7352/7352 [=====] - 113s 15ms/step - loss: 0.1745 - acc: 0.9

Epoch 12/30

7352/7352 [=====] - 114s 15ms/step - loss: 0.1483 - acc: 0.9

Epoch 13/30

7352/7352 [=====] - 113s 15ms/step - loss: 0.1488 - acc: 0.9

Epoch 14/30

7352/7352 [=====] - 113s 15ms/step - loss: 0.1426 - acc: 0.9

Epoch 15/30

7352/7352 [=====] - 113s 15ms/step - loss: 0.1537 - acc: 0.9

Epoch 16/30

7352/7352 [=====] - 114s 15ms/step - loss: 0.1371 - acc: 0.9

Epoch 17/30

7352/7352 [=====] - 112s 15ms/step - loss: 0.1366 - acc: 0.9

Epoch 18/30

7352/7352 [=====] - 112s 15ms/step - loss: 0.1283 - acc: 0.9

Epoch 19/30

7352/7352 [=====] - 112s 15ms/step - loss: 0.1413 - acc: 0.9

Epoch 20/30

7352/7352 [=====] - 111s 15ms/step - loss: 0.1255 - acc: 0.9

Epoch 21/30

7352/7352 [=====] - 112s 15ms/step - loss: 0.1188 - acc: 0.9

Epoch 22/30

7352/7352 [=====] - 112s 15ms/step - loss: 0.1244 - acc: 0.9

Epoch 23/30

7352/7352 [=====] - 112s 15ms/step - loss: 0.1243 - acc: 0.9

Epoch 24/30

7352/7352 [=====] - 112s 15ms/step - loss: 0.1398 - acc: 0.9

Epoch 25/30

7352/7352 [=====] - 113s 15ms/step - loss: 0.1187 - acc: 0.9

Epoch 26/30

7352/7352 [=====] - 113s 15ms/step - loss: 0.1150 - acc: 0.9

Epoch 27/30

7352/7352 [=====] - 112s 15ms/step - loss: 0.1132 - acc: 0.9

Epoch 28/30

7352/7352 [=====] - 115s 16ms/step - loss: 0.1163 - acc: 0.9

Epoch 29/30

7352/7352 [=====] - 114s 15ms/step - loss: 0.1163 - acc: 0.9

Epoch 30/30

7352/7352 [=====] - 113s 15ms/step - loss: 0.1232 - acc: 0.9

<keras.callbacks.History at 0x7f71808f7c50>

```
history1=model1.history
```

```
y_pred=model2.predict(X_test)
```

```
cm_df=get_confusion_matrix(Y_test, y_pred) #Prepare the confusion matrix by using get_conf  
classes=list(cm_df.index) #Class names = Index Names or Column Names in cm_df
```

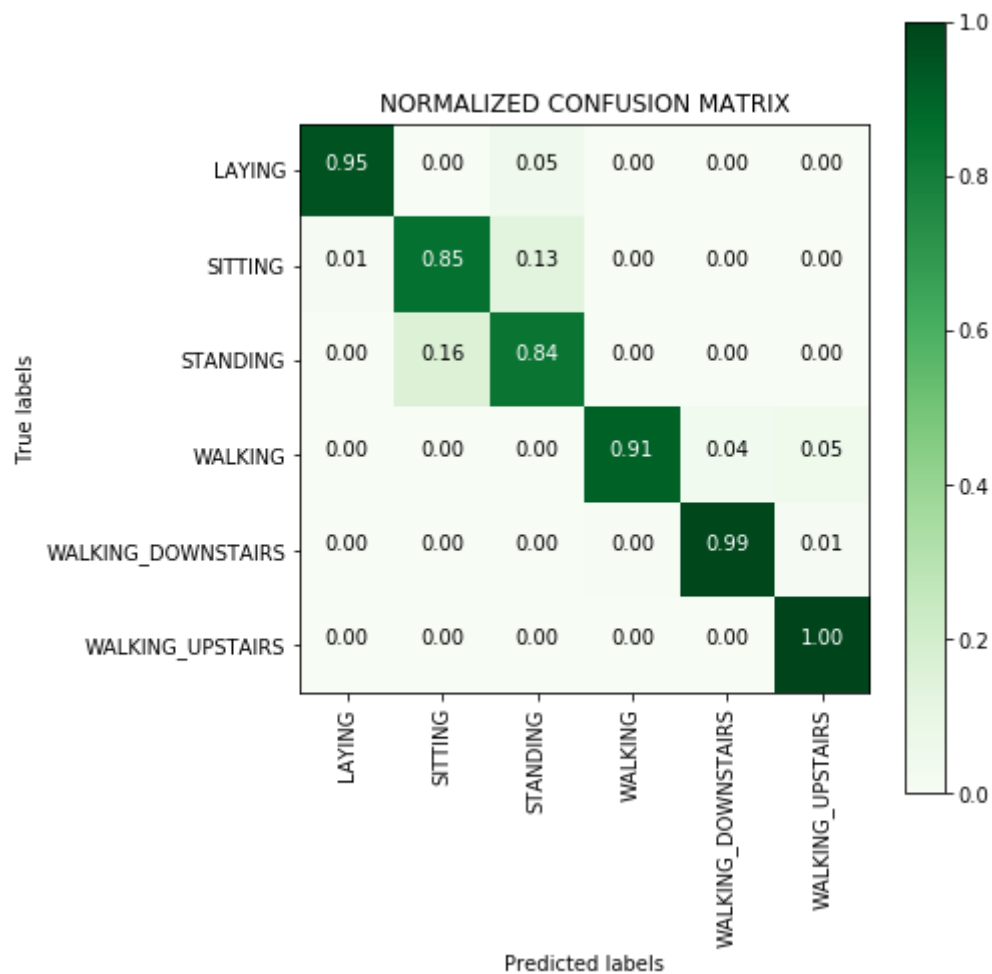
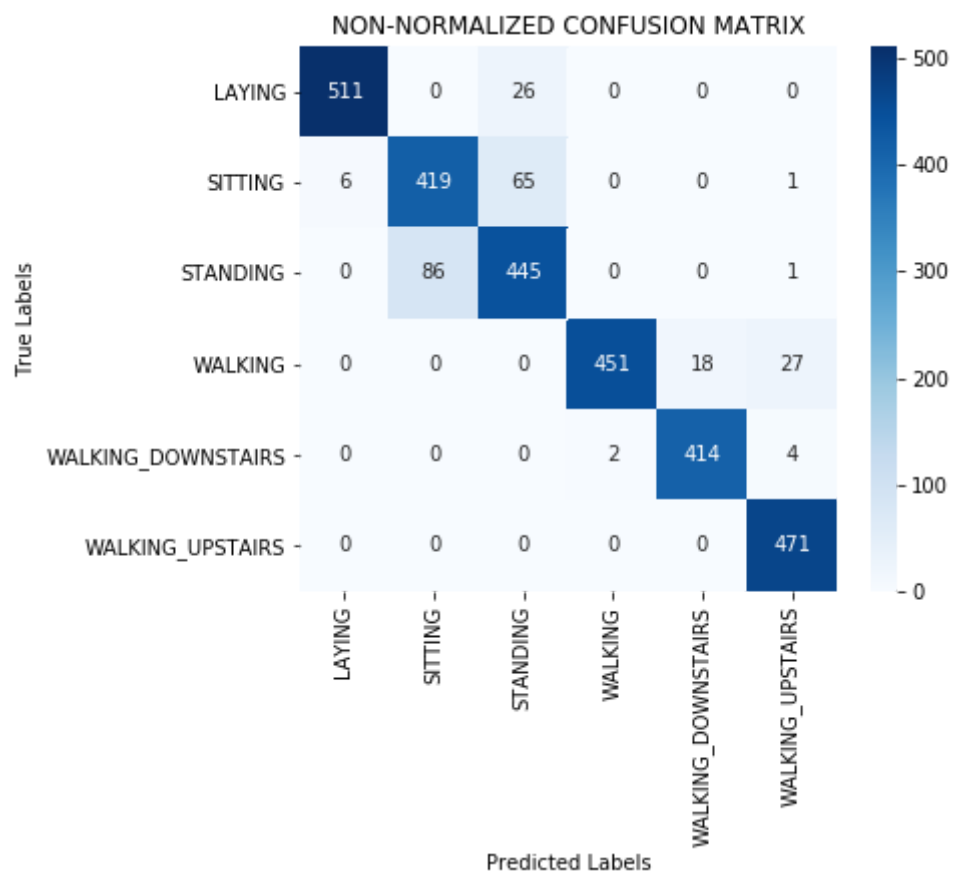
```
#Plot a Non-Normalized confusion matrix
```

```
plot_confusion_matrix(cm_df, classes, normalize=False, title="NON-NORMALIZED CONFUSION MAT
```

```
#Plot a Normalized confusion matrix
```

```
plot_confusion_matrix(cm_df, classes, normalize=True, title="NORMALIZED CONFUSION MATRIX")
```





```
score2 = model2.evaluate(X_test, Y_test)
```

```

    3047/3047 [-----] 16s 5ms/step
print('The Test loss is',score2[0],'and Test accuracy is',score2[1])

```

↳ The Test loss is 0.3247978840465356 and Test accuracy is 0.9199185612487275

## ▼ Classification using Conv1D

```

import pandas as pd
from matplotlib import pyplot
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Dropout
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers.core import Dense, Dropout

model3 = Sequential()
model3.add(Conv1D(filters=128, kernel_size=5, activation='relu',kernel_initializer='glorot_
model3.add(Conv1D(filters=64, kernel_size=5, activation='relu',kernel_initializer='glorot_
model3.add(Dropout(0.2))
model3.add(MaxPooling1D(pool_size=2))
model3.add(Flatten())
model3.add(Dense(50, activation='relu'))
model3.add(Dense(6, activation='softmax'))
model3.summary()

```

↳ Model: "sequential\_21"

Layer (type)	Output Shape	Param #
=====		
conv1d_33 (Conv1D)	(None, 124, 128)	5888
conv1d_34 (Conv1D)	(None, 120, 64)	41024
dropout_22 (Dropout)	(None, 120, 64)	0
max_pooling1d_17 (MaxPooling	(None, 60, 64)	0
flatten_17 (Flatten)	(None, 3840)	0
dense_35 (Dense)	(None, 50)	192050
dense_36 (Dense)	(None, 6)	306
=====		
Total params: 239,268		
Trainable params: 239,268		
Non-trainable params: 0		

```
model3.compile(loss='categorical_crossentropy',  
               optimizer='rmsprop',  
               metrics=['accuracy'])
```

```
model3.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 [=====] - 4s 563us/step - loss: 0.3941 - acc: 0.84

Epoch 2/30

7352/7352 [=====] - 2s 222us/step - loss: 0.1601 - acc: 0.94

Epoch 3/30

7352/7352 [=====] - 2s 216us/step - loss: 0.1292 - acc: 0.95

Epoch 4/30

7352/7352 [=====] - 2s 212us/step - loss: 0.1124 - acc: 0.95

Epoch 5/30

7352/7352 [=====] - 2s 221us/step - loss: 0.0988 - acc: 0.95

Epoch 6/30

7352/7352 [=====] - 2s 219us/step - loss: 0.0931 - acc: 0.96

Epoch 7/30

7352/7352 [=====] - 2s 216us/step - loss: 0.0859 - acc: 0.96

Epoch 8/30

7352/7352 [=====] - 2s 220us/step - loss: 0.1113 - acc: 0.96

Epoch 9/30

7352/7352 [=====] - 2s 213us/step - loss: 0.0921 - acc: 0.96

Epoch 10/30

7352/7352 [=====] - 2s 212us/step - loss: 0.0788 - acc: 0.96

Epoch 11/30

7352/7352 [=====] - 2s 212us/step - loss: 0.0772 - acc: 0.97

Epoch 12/30

7352/7352 [=====] - 2s 212us/step - loss: 0.0614 - acc: 0.97

Epoch 13/30

7352/7352 [=====] - 2s 216us/step - loss: 0.1023 - acc: 0.97

Epoch 14/30

7352/7352 [=====] - 2s 222us/step - loss: 0.0790 - acc: 0.97

Epoch 15/30

7352/7352 [=====] - 2s 220us/step - loss: 0.0916 - acc: 0.97

Epoch 16/30

7352/7352 [=====] - 2s 217us/step - loss: 0.0654 - acc: 0.97

Epoch 17/30

7352/7352 [=====] - 2s 220us/step - loss: 0.0785 - acc: 0.97

Epoch 18/30

7352/7352 [=====] - 2s 229us/step - loss: 0.0648 - acc: 0.97

Epoch 19/30

7352/7352 [=====] - 2s 219us/step - loss: 0.0597 - acc: 0.97

Epoch 20/30

7352/7352 [=====] - 2s 219us/step - loss: 0.0815 - acc: 0.97

Epoch 21/30

7352/7352 [=====] - 2s 215us/step - loss: 0.0619 - acc: 0.97

Epoch 22/30

7352/7352 [=====] - 2s 219us/step - loss: 0.0584 - acc: 0.98

Epoch 23/30

7352/7352 [=====] - 2s 217us/step - loss: 0.0476 - acc: 0.98

Epoch 24/30

7352/7352 [=====] - 2s 213us/step - loss: 0.0501 - acc: 0.98

Epoch 25/30

7352/7352 [=====] - 2s 229us/step - loss: 0.0656 - acc: 0.98

Epoch 26/30

7352/7352 [=====] - 2s 218us/step - loss: 0.0473 - acc: 0.98

Epoch 27/30

7352/7352 [=====] - 2s 211us/step - loss: 0.0749 - acc: 0.98

Epoch 28/30

7352/7352 [=====] - 2s 213us/step - loss: 0.0604 - acc: 0.98

Epoch 29/30

7352/7352 [=====] - 2s 219us/step - loss: 0.0450 - acc: 0.98

Epoch 30/30

7352/7352 [=====] - 2s 225us/step - loss: 0.0806 - acc: 0.98

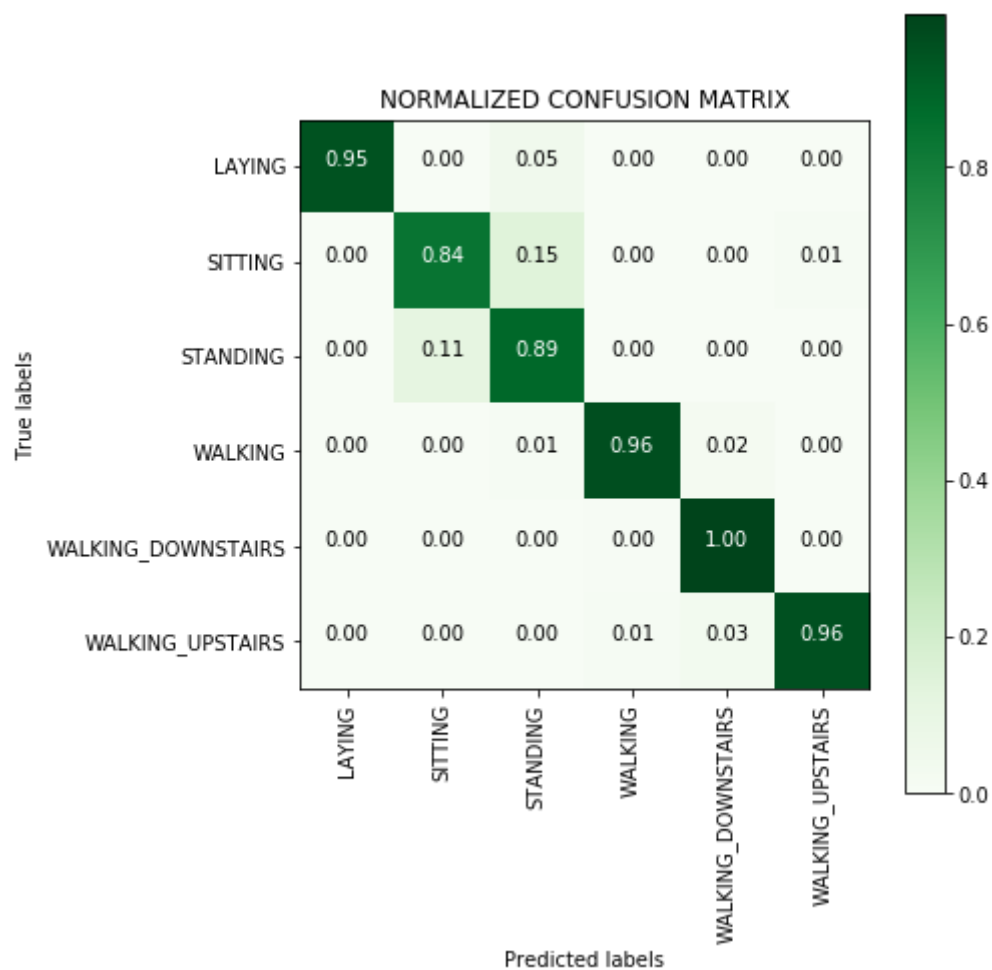
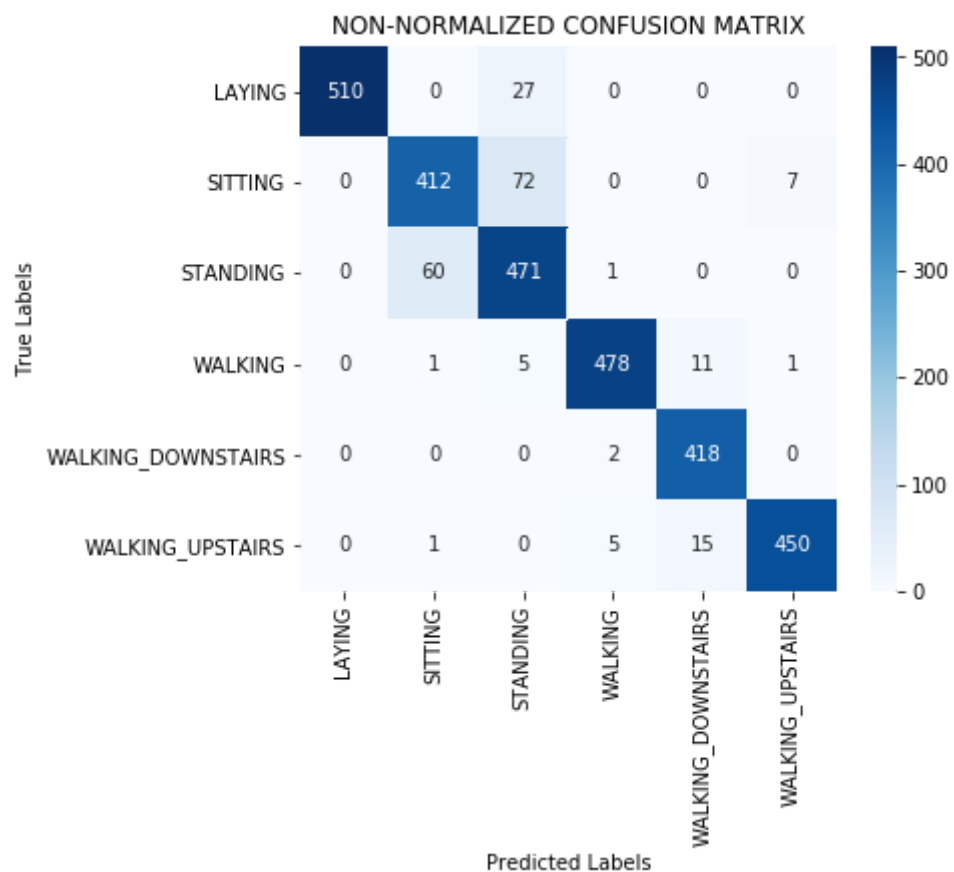
<keras.callbacks.History at 0x7f6f6f43be10>

```
y_pred=model3.predict(X_test)
cm_df=get_confusion_matrix(Y_test, y_pred) #Prepare the confusion matrix by using get_conf
classes=list(cm_df.index) #Class names = Index Names or Column Names in cm_df

#Plot a Non-Normalized confusion matrix
plot_confusion_matrix(cm_df, classes, normalize=False, title="NON-NORMALIZED CONFUSION MAT

#Plot a Normalized confusion matrix
plot_confusion_matrix(cm_df, classes, normalize=True, title="NORMALIZED CONFUSION MATRIX")
```





```
score3 = model3.evaluate(X_test, Y_test)
```





```
print('The Test loss is',score3[0],'and Test accuracy is',score3[1])
```

➞ The Test loss is 0.6447151533074668 and Test accuracy is 0.9294197488971836

```
from keras.layers import Bidirectional
model4 = Sequential()
model4.add(Bidirectional(LSTM(128, activation='relu'),input_shape=(timesteps, input_dim)))
model4.add(Dropout(0.2))
# Adding a dense output layer with sigmoid activation
model4.add(Dense(n_classes, activation='sigmoid'))
model4.summary()
```

➞ Model: "sequential\_24"

Layer (type)	Output Shape	Param #
=====		
bidirectional_3 (Bidirection	(None, 256)	141312
=====		
dropout_25 (Dropout)	(None, 256)	0
=====		
dense_39 (Dense)	(None, 6)	1542
=====		
Total params: 142,854		
Trainable params: 142,854		
Non-trainable params: 0		
=====		

```
model4.compile(loss='categorical_crossentropy',
               optimizer='adam',
               metrics=['accuracy'])
```

```
model4.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test,Y_test),
          epochs=5)
```

➞ Train on 7352 samples, validate on 2947 samples

```
Epoch 1/5
7352/7352 [=====] - 107s 15ms/step - loss: 1.7918 - acc: 0.1
Epoch 2/5
7352/7352 [=====] - 106s 14ms/step - loss: 1.7918 - acc: 0.1
Epoch 3/5
7352/7352 [=====] - 106s 14ms/step - loss: 1.7918 - acc: 0.1
Epoch 4/5
7352/7352 [=====] - 106s 14ms/step - loss: 1.7918 - acc: 0.1
Epoch 5/5
7352/7352 [=====] - 106s 14ms/step - loss: 1.7918 - acc: 0.1
<keras.callbacks.History at 0x7f6f6df81358>
```

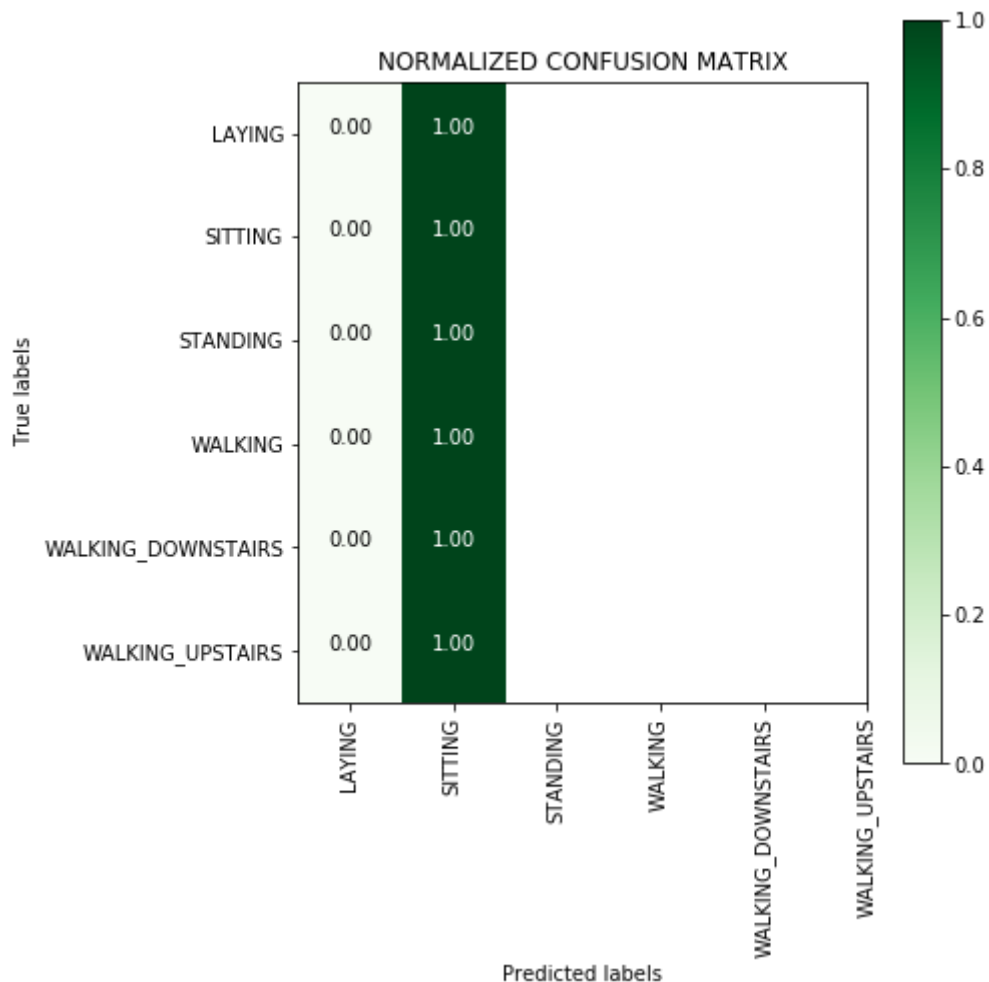
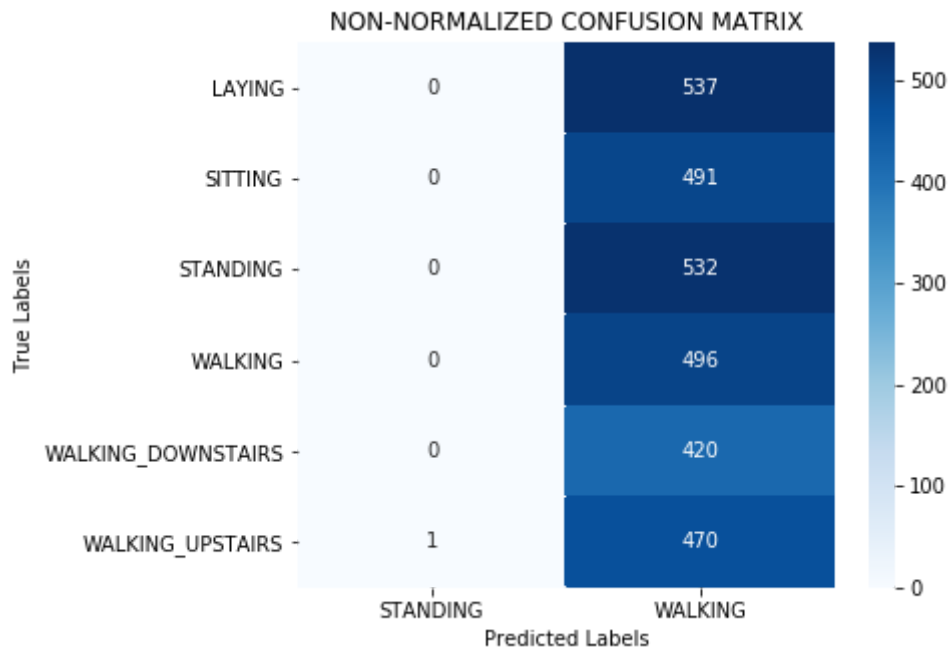
```
y_pred=model4.predict(X_test)
cm_df=get_confusion_matrix(Y_test, y_pred) #Prepare the confusion matrix by using get_conf
classes=list(cm_df.index) #Class names = Index Names or Column Names in cm_df
```

```
#Plot a Non-Normalized confusion matrix
```

```
plot_confusion_matrix(cm_df, classes, normalize=False, title="NON-NORMALIZED CONFUSION MAT
```

```
#Plot a Normalized confusion matrix
```

```
plot_confusion_matrix(cm_df, classes, normalize=True, title="NORMALIZED CONFUSION MATRIX")
```



```
score4 = model4.evaluate(X_test, Y_test)
```



```
2947/2947 [=====] - 15s 5ms/step
```

```
print('The Test Loss is: ', score4[0], 'and Test accuracy is: ', score4[1])
```

```
print( "The test loss is ",score4[0], and test accuracy is ",score4[1])
```

→ The Test loss is 1.7911514966496784 and Test accuracy is 0.168306752629793

```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "Test Accuracy(%)", "Test error(%)"]
x.add_row(["Lstm-single layer",92.53 ,7.47])
x.add_row(["Lstm-double layer",91.99, 8.01])
x.add_row(["CNN 1D", 92.94,7.06])
x.add_row(["Bidirectional Lstm", 16.83,83.17])
print(x)
```

→

Model	Test Accuracy(%)	Test error(%)
Lstm-single layer	92.53	7.47
Lstm-double layer	91.99	8.01
CNN 1D	92.94	7.06
Bidirectional Lstm	16.83	83.17

### Conclusion:

In video the accuracy was 90.09% and by doing some tuning in parameters it has been leveraged t  
Single layer Lstm performed better than double layer

To improve test accuracy we used a 1D CNN model which performed better than 2 layer Lstm and  
I also tried Bidirectional LSTM but it performed very poorly, test accuracy and train accuracy is less  
majority activities as sitting.