HumanActivityRecognition

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

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

How data was recorded

By using the sensors(Gyroscope and accelerometer) in a smartphone, they have captured '3-axial linear acceleratior 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(slic 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 doma

   > 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 ***tGravityA*** 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* (***tBodyA*** ***tBodyGyroJerk-XYZ***).
5. The magnitude of these 3-dimensional signals were calculated using the Euclidian norm. This magnitudes are 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 Fourie obtained were labeled with ***prefix 'f'*** just like original signals with ***prefix 't'***. These signals are labeled as ***fBodyA***
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 c 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 use
    - 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

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 t
  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 an
- 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 calcu
- 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

```
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

```
# 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

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

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()
```

```
C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\pandas\io\parsers.py:678: UserWarnin
  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	tBody mac
2403	0.302769	-0.007871	-0.07725	-0.954795	-0.868161	-0.911811	-0.961271	-0.88

1 rows × 564 columns

```
train.shape
```

```
(7352, 564)
```

Obtain the test data

```
# 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=fe

# 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()
```

```
C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\pandas\io\parsers.py:678: UserWarnin
  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	tBody mac
2430	0.272074	-0.015423	-0.051673	-0.989788	-0.974372	-0.924292	-0.991696	-0

1 rows × 564 columns

```
test.shape
```

```
(2947, 564)
```

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())))
```

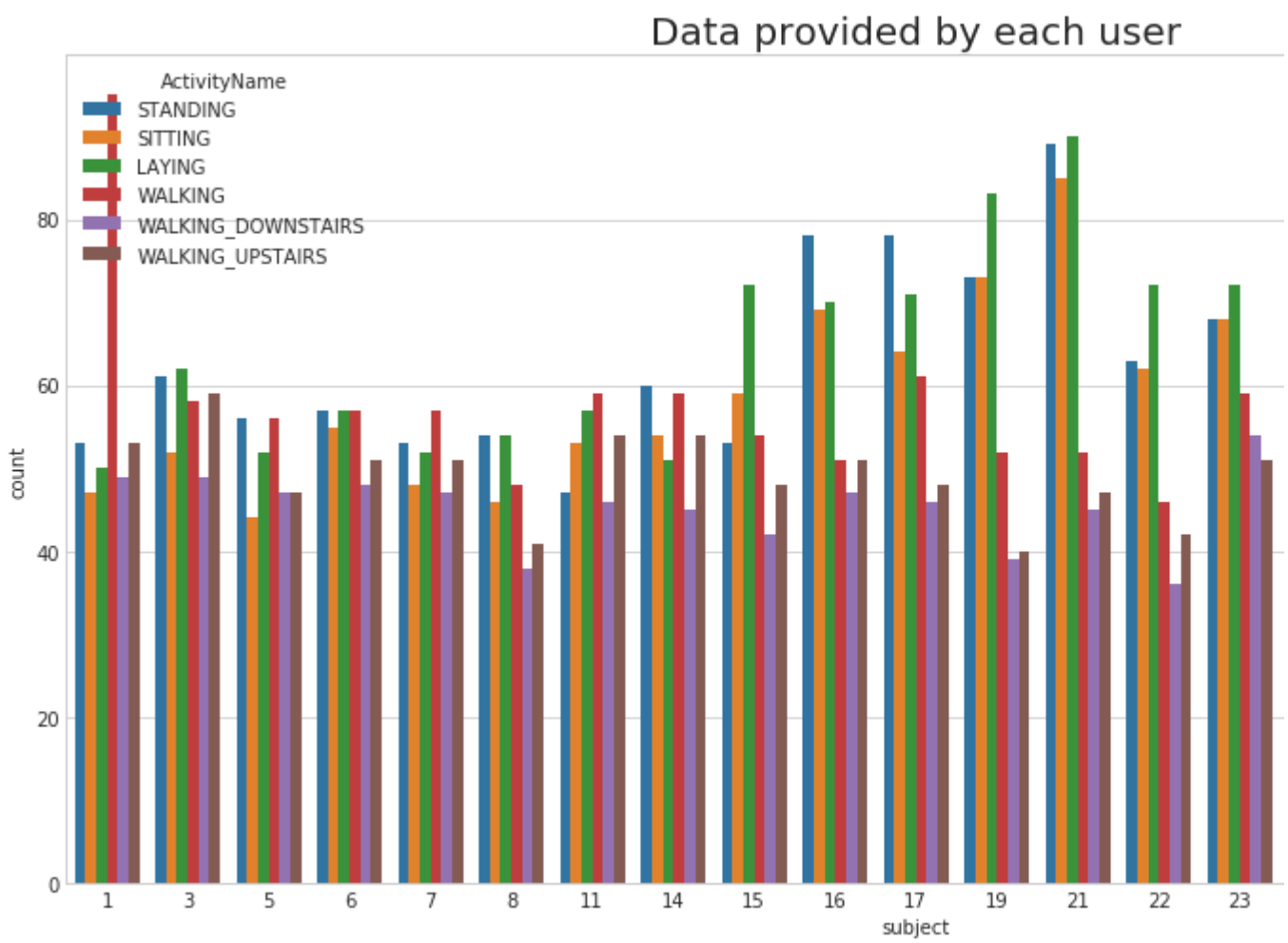
```
No of duplicates in train: 0
No of duplicates in test : 0
```

2. Checking for NaN/null values

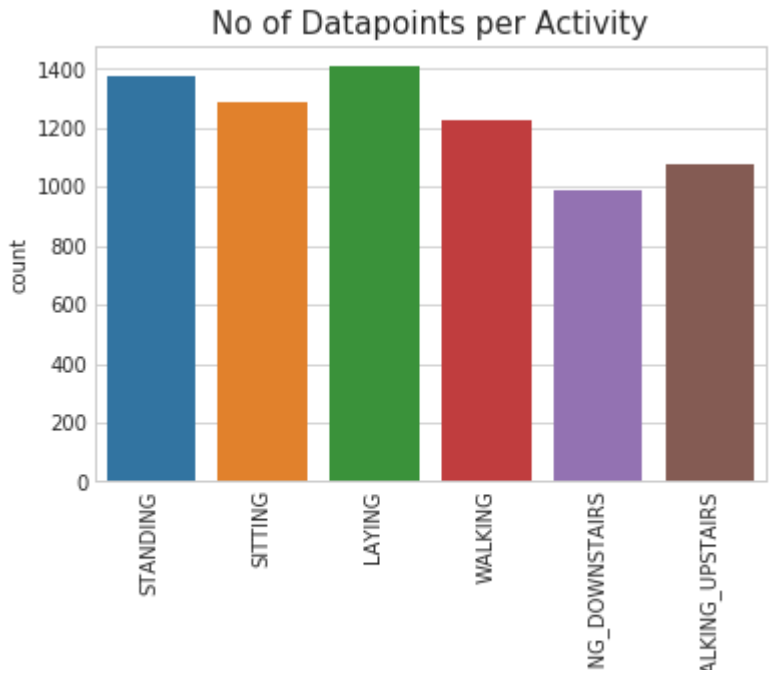
```
print('We have {} NaN/Null values in train'.format(train.isnull().values.sum()))
print('We have {} NaN/Null values in test'.format(test.isnull().values.sum()))
```

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()
```




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



Observation

1. 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',
       ...
       'angletBodyAccMeangravity', 'angletBodyAccJerkMeangravityMean',
       'angletBodyGyroMeangravityMean', 'angletBodyGyroJerkMeangravityMean',
       '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)
```

Exploratory Data Analysis

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

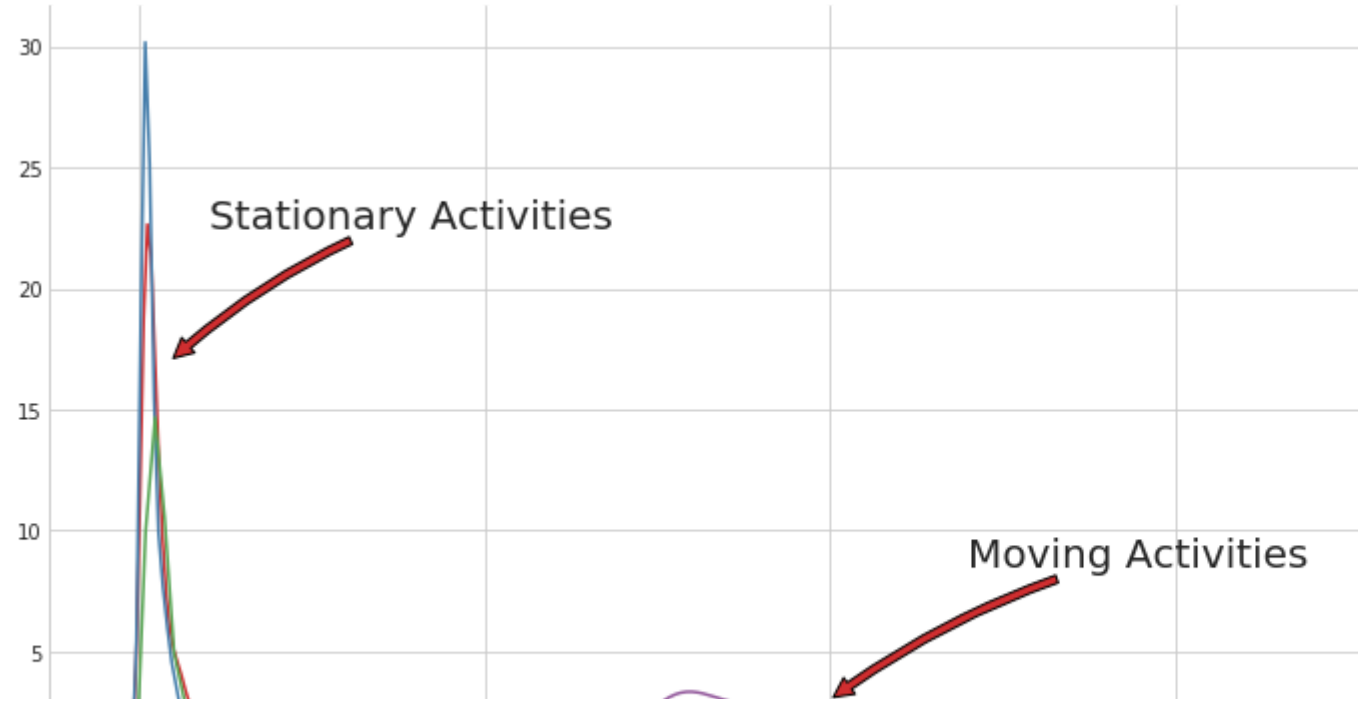
1. Featuring Engineering from Domain Knowledge

- **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

```
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()
```

```
C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWar
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



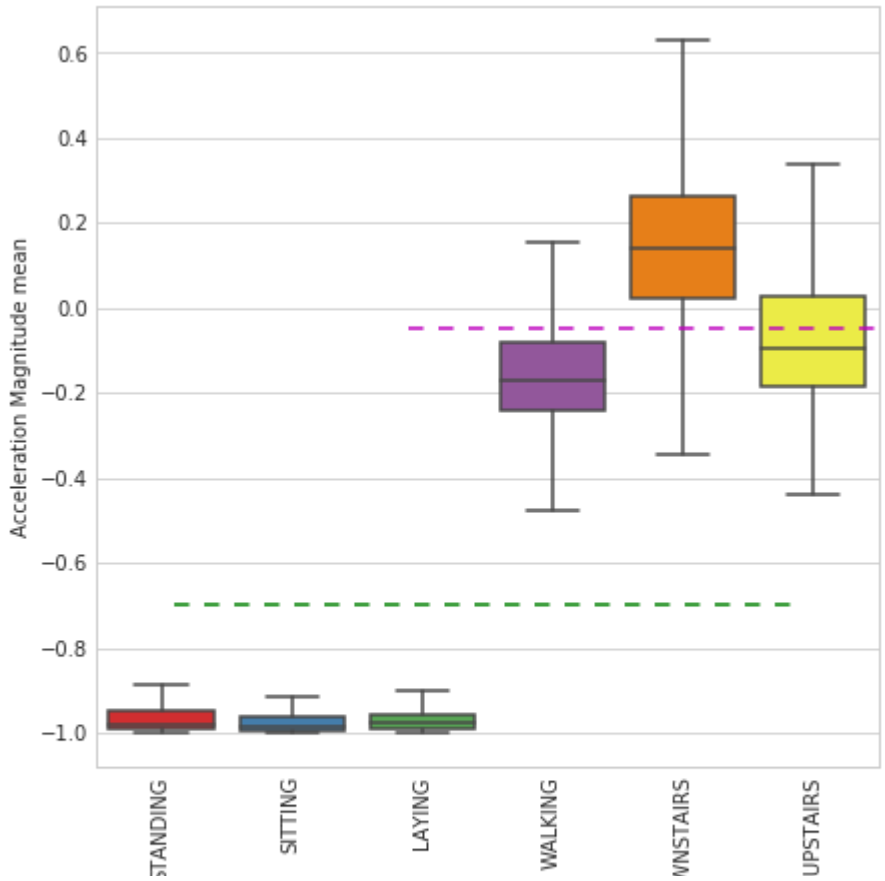

```
# for plotting purposes taking datapoints of each activity to a different dataframe
df1 = train[train['Activity']==1]
df2 = train[train['Activity']==2]
df3 = train[train['Activity']==3]
df4 = train[train['Activity']==4]
df5 = train[train['Activity']==5]
df6 = train[train['Activity']==6]

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()
```

```
C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWar
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

3. Magnitude of an acceleration can saperate it wel

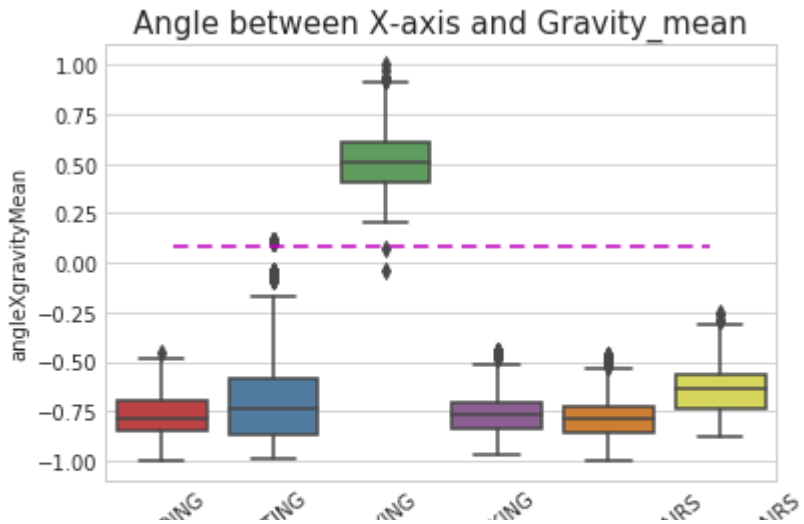
tBodyAccMagmean

```
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()
```

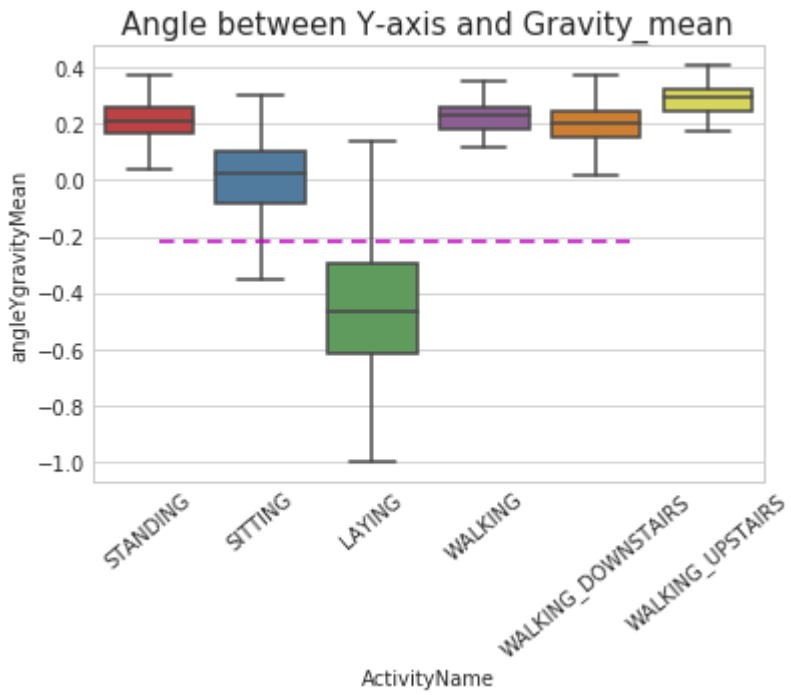


4. Position of GravityAccelerationComponants also matters

```
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()
```




```
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()
```

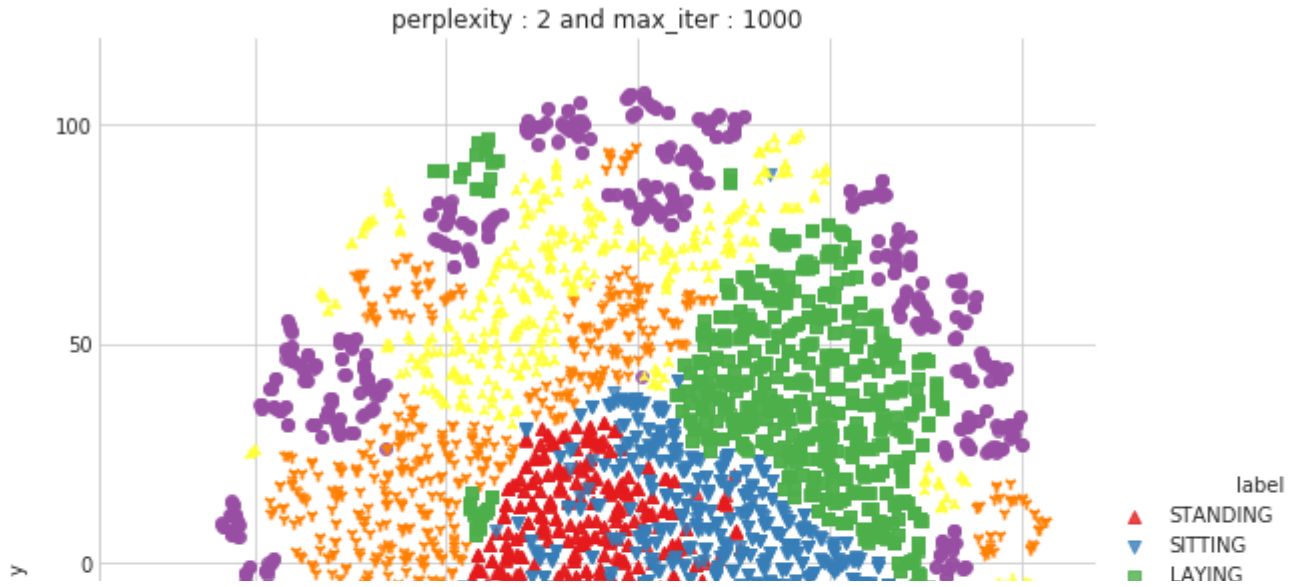


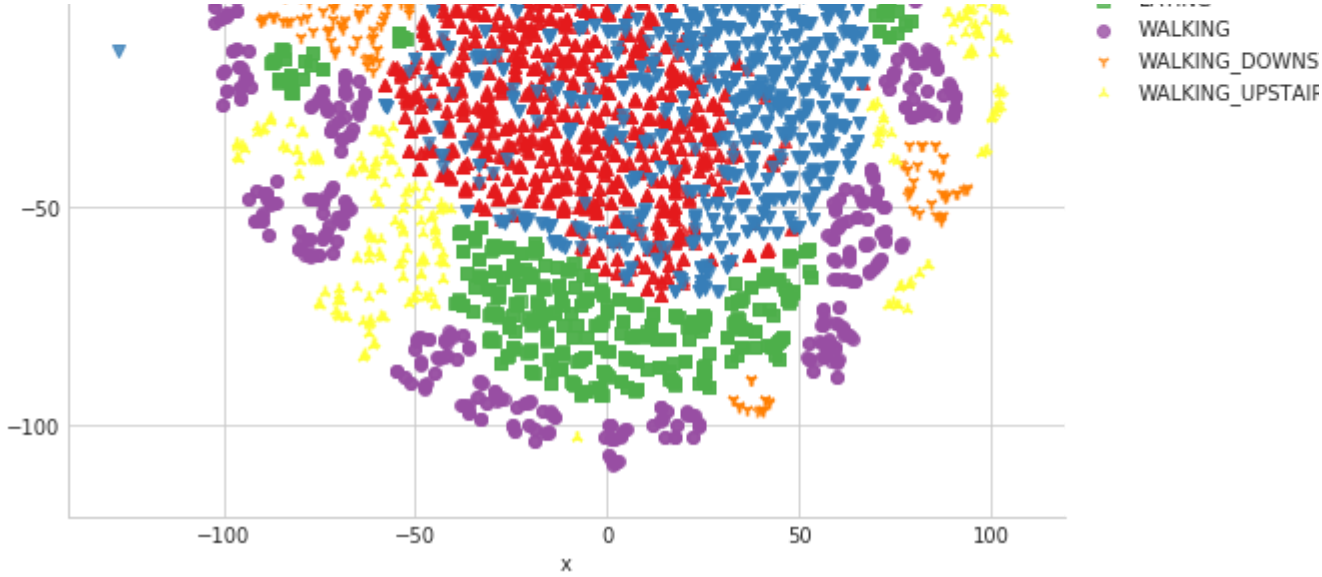
Apply t-sne on the data

```
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(perplexity
        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 0.838s...
[t-SNE] Computed neighbors for 7352 samples in 86.791s...
[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 0.142s
[t-SNE] Iteration 50: error = 124.7675247, gradient norm = 0.0255740 (50 iterations in 1
[t-SNE] Iteration 100: error = 107.6125488, gradient norm = 0.0274877 (50 iterations in
[t-SNE] Iteration 150: error = 101.0450897, gradient norm = 0.0184977 (50 iterations in
[t-SNE] Iteration 200: error = 97.5868759, gradient norm = 0.0151198 (50 iterations in 1
[t-SNE] Iteration 250: error = 95.2531052, gradient norm = 0.0108517 (50 iterations in 1
[t-SNE] KL divergence after 250 iterations with early exaggeration: 95.253105
[t-SNE] Iteration 300: error = 4.1179585, gradient norm = 0.0015632 (50 iterations in 13
[t-SNE] Iteration 350: error = 3.2094731, gradient norm = 0.0010008 (50 iterations in 11
[t-SNE] Iteration 400: error = 2.7811444, gradient norm = 0.0007083 (50 iterations in 10
[t-SNE] Iteration 450: error = 2.5168667, gradient norm = 0.0005688 (50 iterations in 11
[t-SNE] Iteration 500: error = 2.3340361, gradient norm = 0.0004776 (50 iterations in 11
[t-SNE] Iteration 550: error = 2.1960475, gradient norm = 0.0004157 (50 iterations in 11
[t-SNE] Iteration 600: error = 2.0868433, gradient norm = 0.0003702 (50 iterations in 11
[t-SNE] Iteration 650: error = 1.9973743, gradient norm = 0.0003322 (50 iterations in 11
[t-SNE] Iteration 700: error = 1.9220415, gradient norm = 0.0003040 (50 iterations in 11
[t-SNE] Iteration 750: error = 1.8569380, gradient norm = 0.0002783 (50 iterations in 12
[t-SNE] Iteration 800: error = 1.8002290, gradient norm = 0.0002570 (50 iterations in 11
[t-SNE] Iteration 850: error = 1.7502086, gradient norm = 0.0002419 (50 iterations in 11
[t-SNE] Iteration 900: error = 1.7056594, gradient norm = 0.0002252 (50 iterations in 11
[t-SNE] Iteration 950: error = 1.6655064, gradient norm = 0.0002115 (50 iterations in 11
[t-SNE] Iteration 1000: error = 1.6291345, gradient norm = 0.0001993 (50 iterations in 1
[t-SNE] Error after 1000 iterations: 1.629135
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```

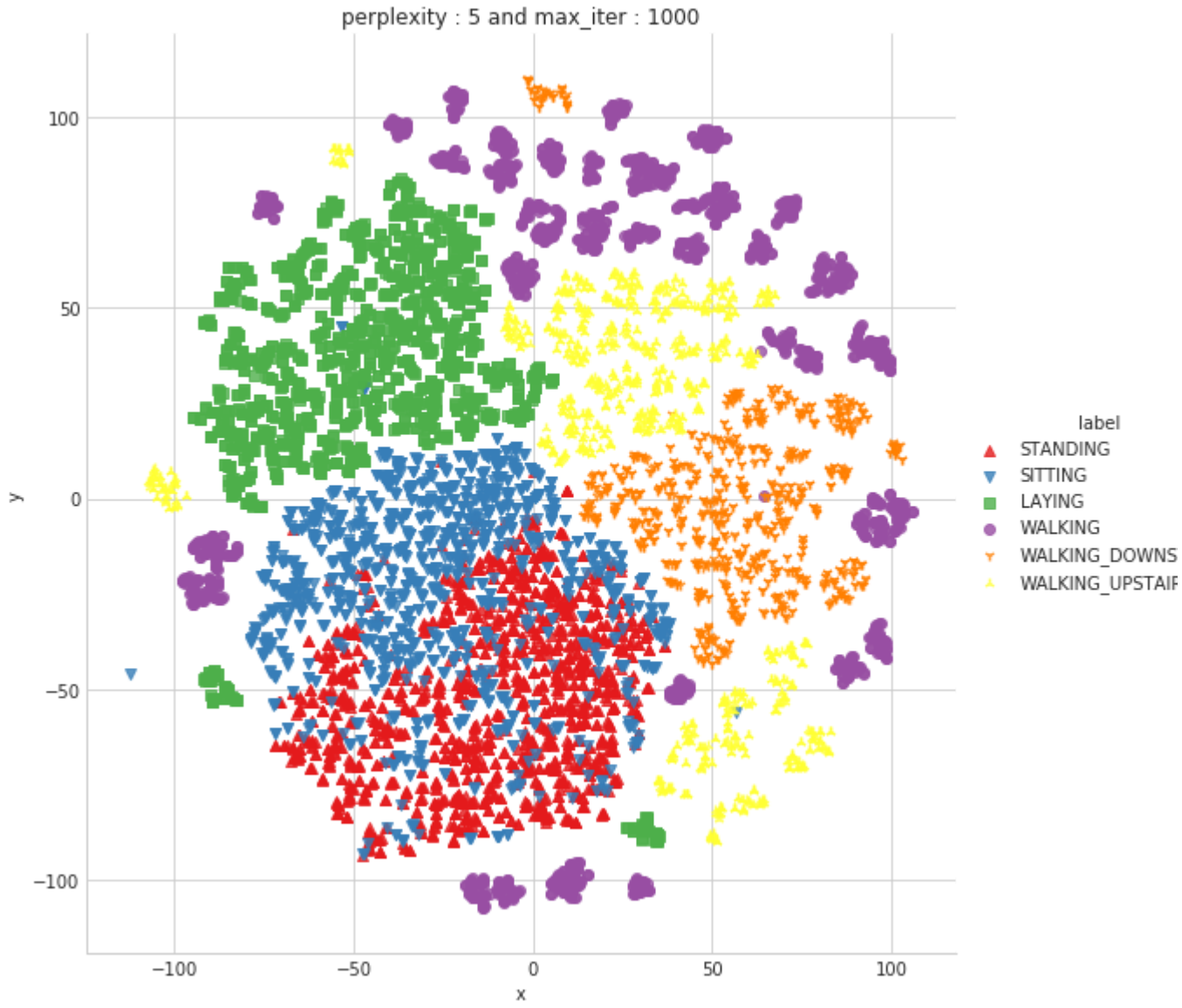




Done

```
performing tsne with perplexity 5 and with 1000 iterations at max
[t-SNE] Computing 16 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.558s...
[t-SNE] Computed neighbors for 7352 samples in 84.092s...
[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 0.109s
[t-SNE] Iteration 50: error = 114.1764755, gradient norm = 0.0184469 (50 iterations in 2
[t-SNE] Iteration 100: error = 97.7323837, gradient norm = 0.0147993 (50 iterations in 1
[t-SNE] Iteration 150: error = 93.2218399, gradient norm = 0.0100412 (50 iterations in 1
[t-SNE] Iteration 200: error = 91.2443314, gradient norm = 0.0075446 (50 iterations in 1
[t-SNE] Iteration 250: error = 90.0597992, gradient norm = 0.0052089 (50 iterations in 1
[t-SNE] KL divergence after 250 iterations with early exaggeration: 90.059799
[t-SNE] Iteration 300: error = 3.5737796, gradient norm = 0.0014619 (50 iterations in 11
[t-SNE] Iteration 350: error = 2.8149288, gradient norm = 0.0007498 (50 iterations in 12
[t-SNE] Iteration 400: error = 2.4327743, gradient norm = 0.0005248 (50 iterations in 12
[t-SNE] Iteration 450: error = 2.2149560, gradient norm = 0.0004035 (50 iterations in 11
[t-SNE] Iteration 500: error = 2.0701339, gradient norm = 0.0003292 (50 iterations in 11
[t-SNE] Iteration 550: error = 1.9648148, gradient norm = 0.0002835 (50 iterations in 11
[t-SNE] Iteration 600: error = 1.8835868, gradient norm = 0.0002445 (50 iterations in 12
[t-SNE] Iteration 650: error = 1.8185158, gradient norm = 0.0002177 (50 iterations in 11
[t-SNE] Iteration 700: error = 1.7645615, gradient norm = 0.0002004 (50 iterations in 14
[t-SNE] Iteration 750: error = 1.7192863, gradient norm = 0.0001799 (50 iterations in 12
[t-SNE] Iteration 800: error = 1.6804105, gradient norm = 0.0001663 (50 iterations in 12
[t-SNE] Iteration 850: error = 1.6466123, gradient norm = 0.0001545 (50 iterations in 12
[t-SNE] Iteration 900: error = 1.6171678, gradient norm = 0.0001443 (50 iterations in 16
[t-SNE] Iteration 950: error = 1.5909027, gradient norm = 0.0001352 (50 iterations in 11
[t-SNE] Iteration 1000: error = 1.5674709, gradient norm = 0.0001271 (50 iterations in 1
[t-SNE] Error after 1000 iterations: 1.567471
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory
```

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



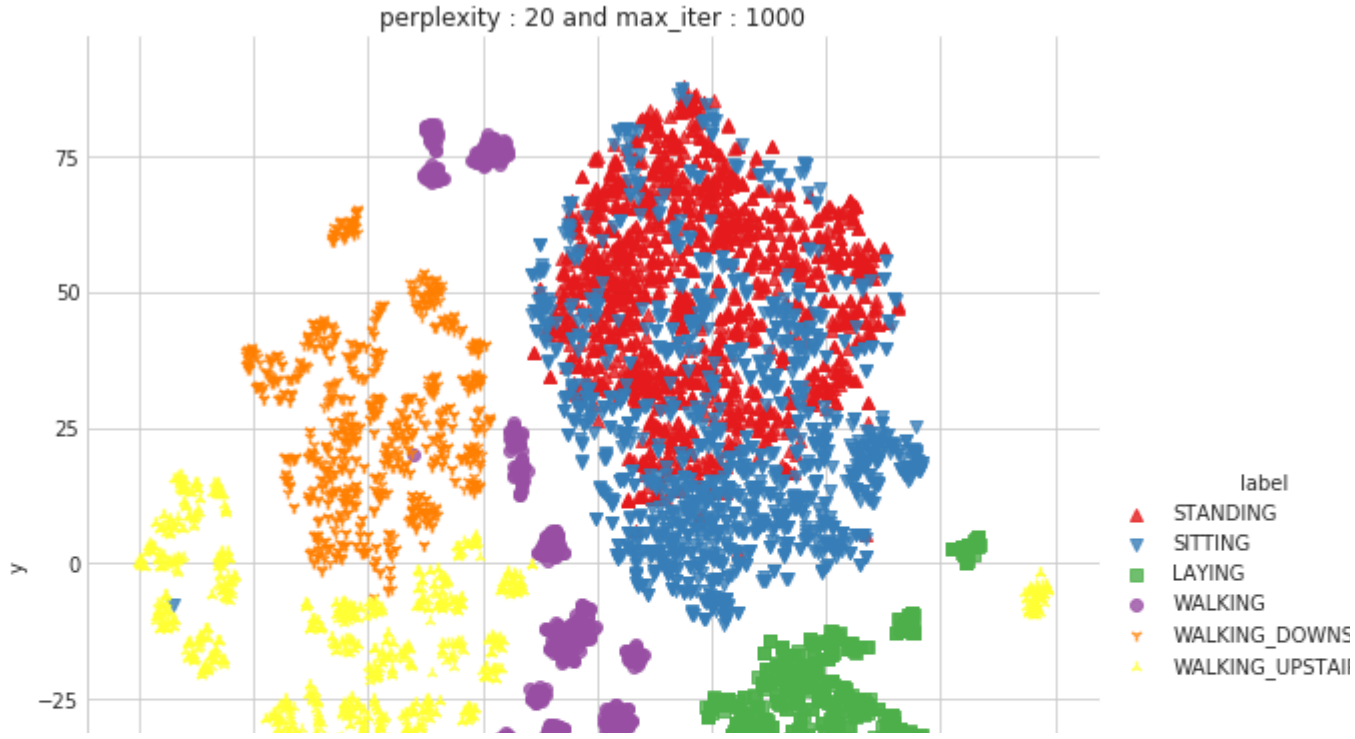
Done

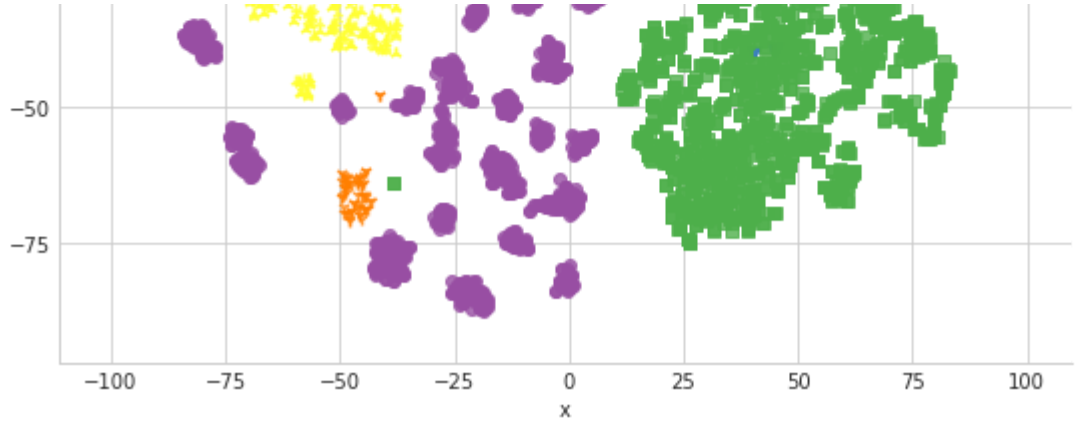
```
performing tsne with perplexity 10 and with 1000 iterations at max
[t-SNE] Computing 31 nearest neighbors...
[t-SNE] Indexed 7352 samples in 1.583s...
[t-SNE] Computed neighbors for 7352 samples in 102.330s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional 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 0.194s
[t-SNE] Iteration 50: error = 105.3449936, gradient norm = 0.0255474 (50 iterations in 2
[t-SNE] Iteration 100: error = 89.9901810, gradient norm = 0.0107842 (50 iterations in 1
[t-SNE] Iteration 150: error = 87.1829529, gradient norm = 0.0060275 (50 iterations in 1
[t-SNE] Iteration 200: error = 85.9958344, gradient norm = 0.0036191 (50 iterations in 1
[t-SNE] Iteration 250: error = 85.3113556, gradient norm = 0.0031416 (50 iterations in 1
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.311356
[t-SNE] Iteration 300: error = 3.1348600, gradient norm = 0.0013892 (50 iterations in 13
```

```
[t-SNE] Iteration 350: error = 2.4918995, gradient norm = 0.0006502 (50 iterations in 12
[t-SNE] Iteration 400: error = 2.1717200, gradient norm = 0.0004260 (50 iterations in 12
[t-SNE] Iteration 450: error = 1.9871796, gradient norm = 0.0003183 (50 iterations in 12
[t-SNE] Iteration 500: error = 1.8690444, gradient norm = 0.0002527 (50 iterations in 13
[t-SNE] Iteration 550: error = 1.7854657, gradient norm = 0.0002124 (50 iterations in 12
[t-SNE] Iteration 600: error = 1.7225573, gradient norm = 0.0001830 (50 iterations in 13
[t-SNE] Iteration 650: error = 1.6734198, gradient norm = 0.0001619 (50 iterations in 12
[t-SNE] Iteration 700: error = 1.6337041, gradient norm = 0.0001437 (50 iterations in 13
[t-SNE] Iteration 750: error = 1.6008332, gradient norm = 0.0001300 (50 iterations in 14
[t-SNE] Iteration 800: error = 1.5733235, gradient norm = 0.0001184 (50 iterations in 15
[t-SNE] Iteration 850: error = 1.5497011, gradient norm = 0.0001109 (50 iterations in 13
[t-SNE] Iteration 900: error = 1.5296534, gradient norm = 0.0001026 (50 iterations in 13
[t-SNE] Iteration 950: error = 1.5119159, gradient norm = 0.0000974 (50 iterations in 13
[t-SNE] Iteration 1000: error = 1.4965218, gradient norm = 0.0000896 (50 iterations in 1
[t-SNE] Error after 1000 iterations: 1.496522
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 0.538s...
```

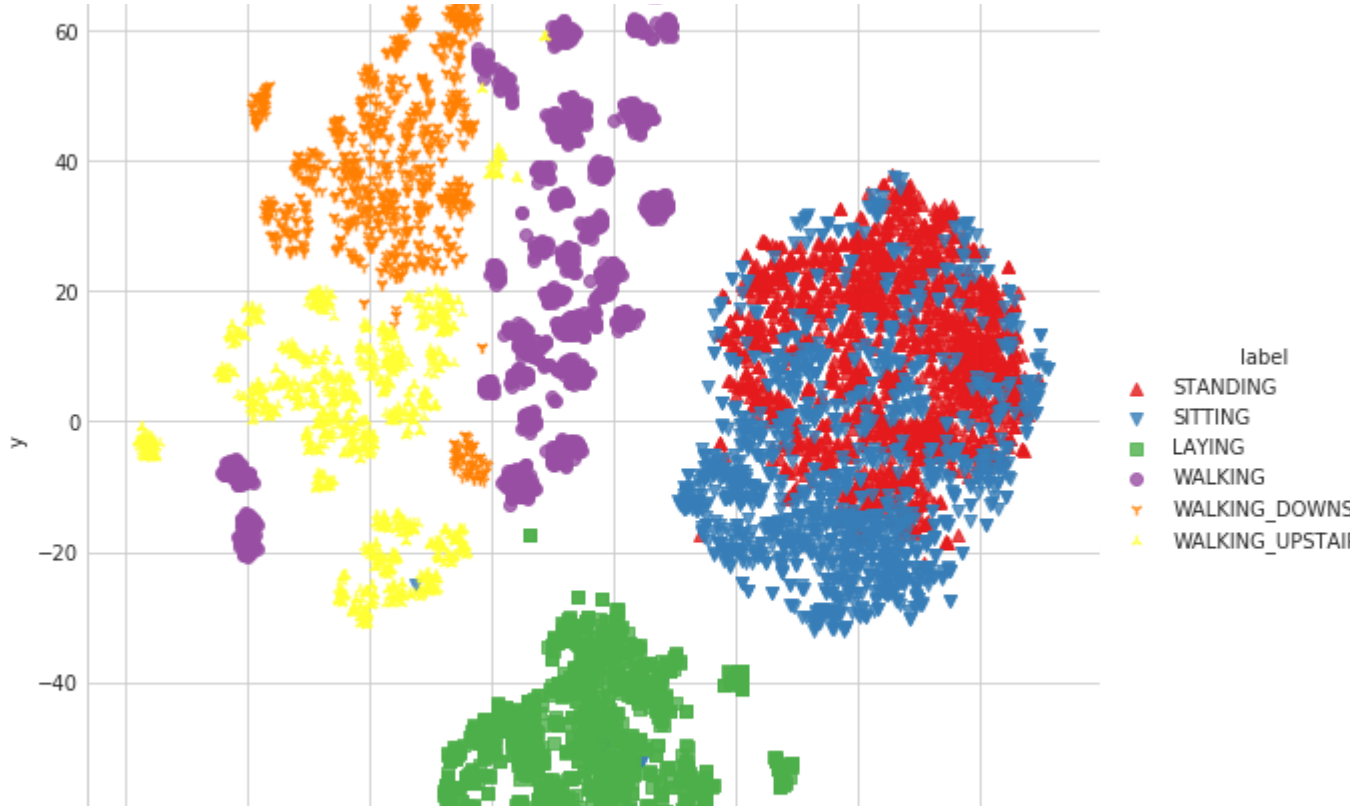
```
[t-SNE] Computed neighbors for 7352 samples in 85.907s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities 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 0.375s
[t-SNE] Iteration 50: error = 97.6168213, gradient norm = 0.0183816 (50 iterations in 22
[t-SNE] Iteration 100: error = 84.0879593, gradient norm = 0.0061127 (50 iterations in 1
[t-SNE] Iteration 150: error = 82.0260391, gradient norm = 0.0044832 (50 iterations in 1
[t-SNE] Iteration 200: error = 81.2621918, gradient norm = 0.0031052 (50 iterations in 1
[t-SNE] Iteration 250: error = 80.8613968, gradient norm = 0.0019238 (50 iterations in 2
[t-SNE] KL divergence after 250 iterations with early exaggeration: 80.861397
[t-SNE] Iteration 300: error = 2.7037404, gradient norm = 0.0013109 (50 iterations in 17
[t-SNE] Iteration 350: error = 2.1686161, gradient norm = 0.0005734 (50 iterations in 15
[t-SNE] Iteration 400: error = 1.9184941, gradient norm = 0.0003487 (50 iterations in 16
[t-SNE] Iteration 450: error = 1.7721525, gradient norm = 0.0002474 (50 iterations in 16
[t-SNE] Iteration 500: error = 1.6781281, gradient norm = 0.0001933 (50 iterations in 17
[t-SNE] Iteration 550: error = 1.6138120, gradient norm = 0.0001576 (50 iterations in 17
[t-SNE] Iteration 600: error = 1.5669305, gradient norm = 0.0001353 (50 iterations in 18
[t-SNE] Iteration 650: error = 1.5316076, gradient norm = 0.0001173 (50 iterations in 17
[t-SNE] Iteration 700: error = 1.5040718, gradient norm = 0.0001055 (50 iterations in 17
[t-SNE] Iteration 750: error = 1.4819882, gradient norm = 0.0000951 (50 iterations in 17
[t-SNE] Iteration 800: error = 1.4639986, gradient norm = 0.0000877 (50 iterations in 18
[t-SNE] Iteration 850: error = 1.4487678, gradient norm = 0.0000813 (50 iterations in 16
[t-SNE] Iteration 900: error = 1.4360871, gradient norm = 0.0000757 (50 iterations in 18
[t-SNE] Iteration 950: error = 1.4253293, gradient norm = 0.0000715 (50 iterations in 16
[t-SNE] Iteration 1000: error = 1.4159945, gradient norm = 0.0000676 (50 iterations in 1
[t-SNE] Error after 1000 iterations: 1.415995
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 0.595s...
[t-SNE] Computed neighbors for 7352 samples in 91.571s...
[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 0.919s
[t-SNE] Iteration 50: error = 86.9809418, gradient norm = 0.0147315 (50 iterations in 30
[t-SNE] Iteration 100: error = 75.5526047, gradient norm = 0.0040142 (50 iterations in 2
[t-SNE] Iteration 150: error = 74.5927505, gradient norm = 0.0021310 (50 iterations in 2
[t-SNE] Iteration 200: error = 74.2324142, gradient norm = 0.0015615 (50 iterations in 2
[t-SNE] Iteration 250: error = 74.0571442, gradient norm = 0.0015757 (50 iterations in 2
[t-SNE] KL divergence after 250 iterations with early exaggeration: 74.057144
[t-SNE] Iteration 300: error = 2.1537080, gradient norm = 0.0011831 (50 iterations in 25
[t-SNE] Iteration 350: error = 1.7572905, gradient norm = 0.0004833 (50 iterations in 24
[t-SNE] Iteration 400: error = 1.5880311, gradient norm = 0.0002797 (50 iterations in 24
[t-SNE] Iteration 450: error = 1.4938954, gradient norm = 0.0001910 (50 iterations in 25
[t-SNE] Iteration 500: error = 1.4340088, gradient norm = 0.0001419 (50 iterations in 25
[t-SNE] Iteration 550: error = 1.3921987, gradient norm = 0.0001183 (50 iterations in 25
[t-SNE] Iteration 600: error = 1.3633460, gradient norm = 0.0000964 (50 iterations in 24
[t-SNE] Iteration 650: error = 1.3417592, gradient norm = 0.0000830 (50 iterations in 24
[t-SNE] Iteration 700: error = 1.3259081, gradient norm = 0.0000755 (50 iterations in 24
[t-SNE] Iteration 750: error = 1.3140126, gradient norm = 0.0000717 (50 iterations in 24
[t-SNE] Iteration 800: error = 1.3053392, gradient norm = 0.0000650 (50 iterations in 24
[t-SNE] Iteration 850: error = 1.2983644, gradient norm = 0.0000614 (50 iterations in 24
[t-SNE] Iteration 900: error = 1.2925525, gradient norm = 0.0000600 (50 iterations in 24
[t-SNE] Iteration 950: error = 1.2876562, gradient norm = 0.0000559 (50 iterations in 24
[t-SNE] Iteration 1000: error = 1.2835616, gradient norm = 0.0000518 (50 iterations in 2
[t-SNE] Error after 1000 iterations: 1.283562
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```



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	-0.913526
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944

3 rows × 564 columns

```
# 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,))
```

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 specifie

```
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 matr
    plt.show()

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

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

    return results
```

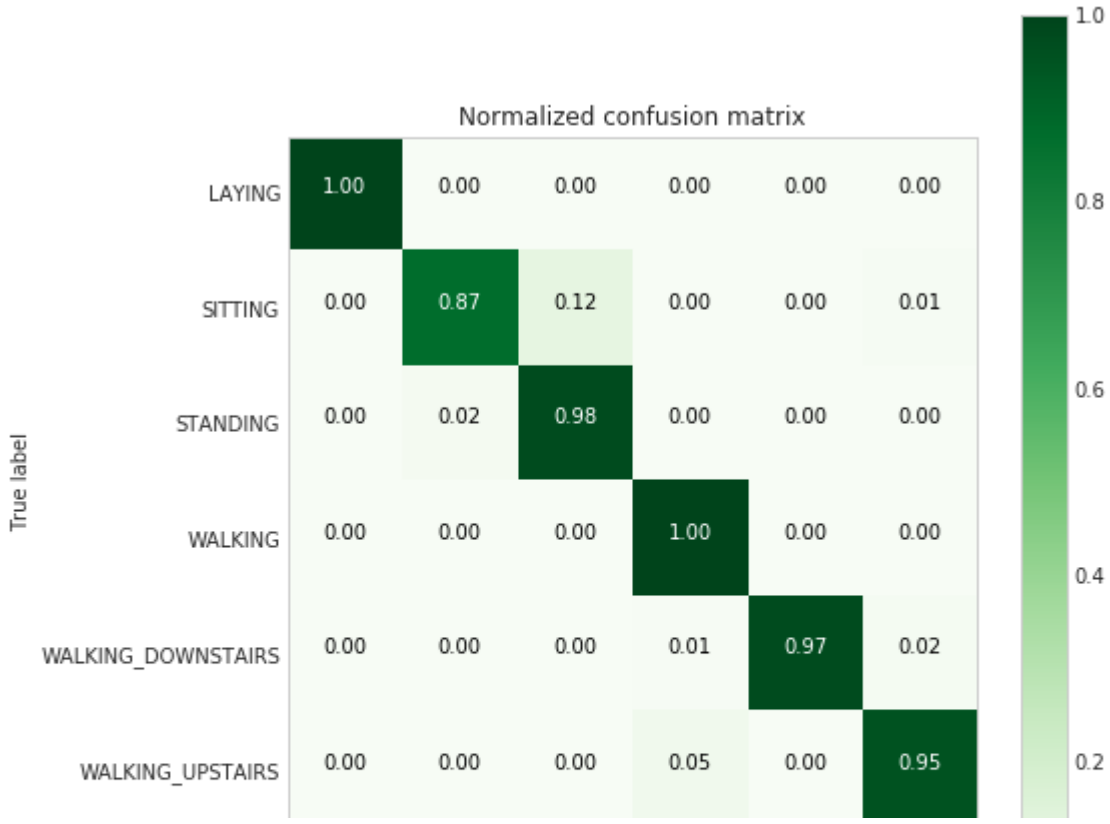
Method to print the gridsearch Attributes

```
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

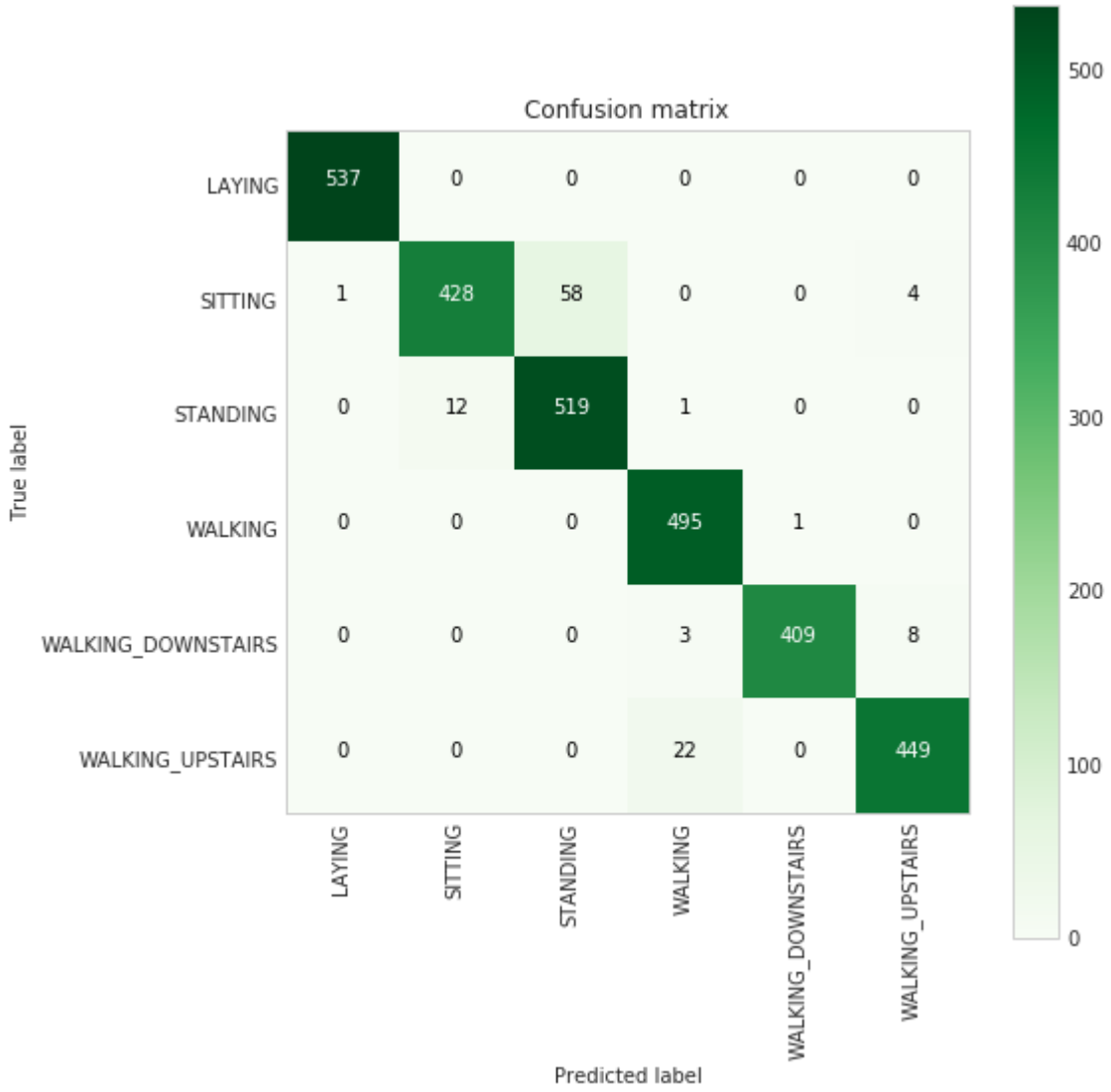
```
from sklearn import linear_model
from sklearn import metrics

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, class_labels=l
```

```
training the model..
Fitting 3 folds for each of 12 candidates, totalling 36 fits
[Parallel(n_jobs=-1)]: Done  36 out of  36 | elapsed:  2.5min finished
Done


training_time(HH:MM:SS.ms) - 0:02:44.602273


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.010975


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

```
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()
```



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

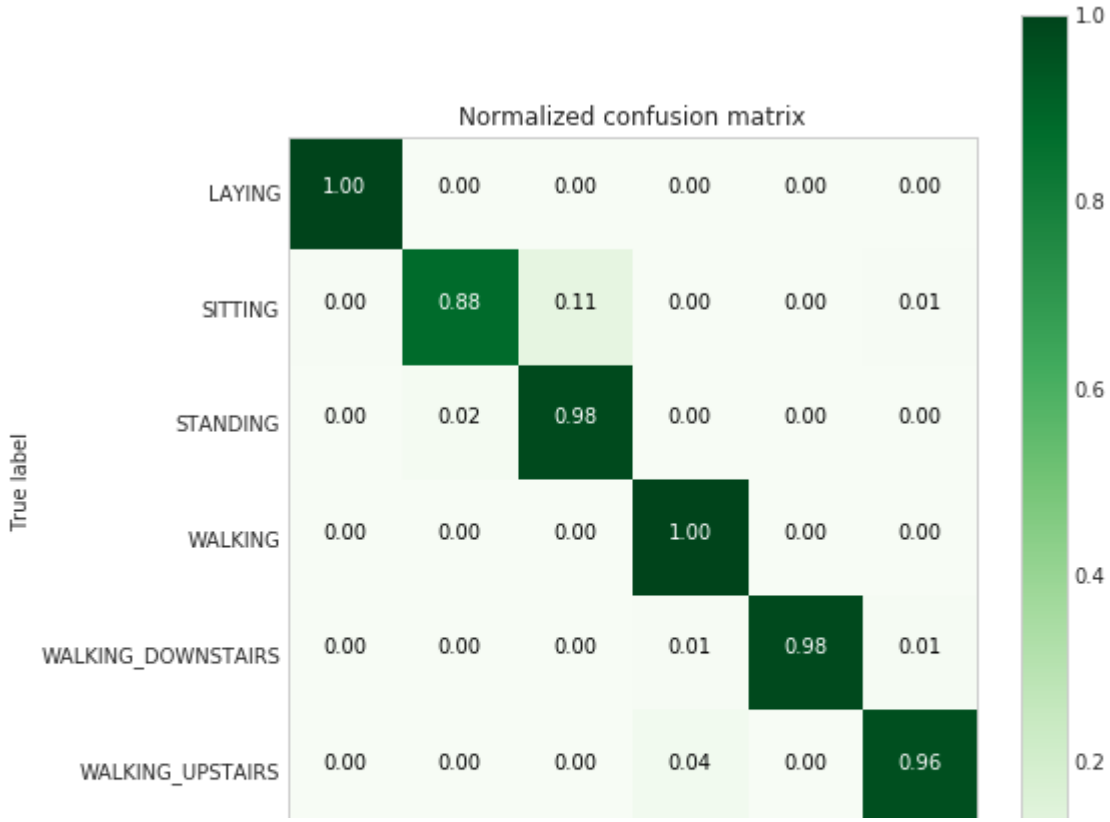
```
-------------------------
|      Best Estimator     |
-------------------------

        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       |
```

2. Linear SVC with GridSearch

```
            0.9481371055495104
```

```
from sklearn.svm import LinearSVC


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=labe
```

```
training the model..
Fitting 3 folds for each of 6 candidates, totalling 18 fits
[Parallel(n_jobs=-1)]: Done  18 out of  18 | elapsed:  1.1min finished
Done


training_time(HH:MM:SS.ms) - 0:01:19.510271


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.012008


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

    0.9670851713607058


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

 [[537   0   0   0   0   0]
 [  2 430  56   0   0   3]
 [  0  10 521   1   0   0]
 [  0   0   0 496   0   0]
 [  0   0   0   3 412   5]
 [  0   0   0  17   0 454]]
```




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

            LAYING       1.00      1.00      1.00       537
           SITTING       0.98      0.88      0.92       491
          STANDING       0.90      0.98      0.94       532
```

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

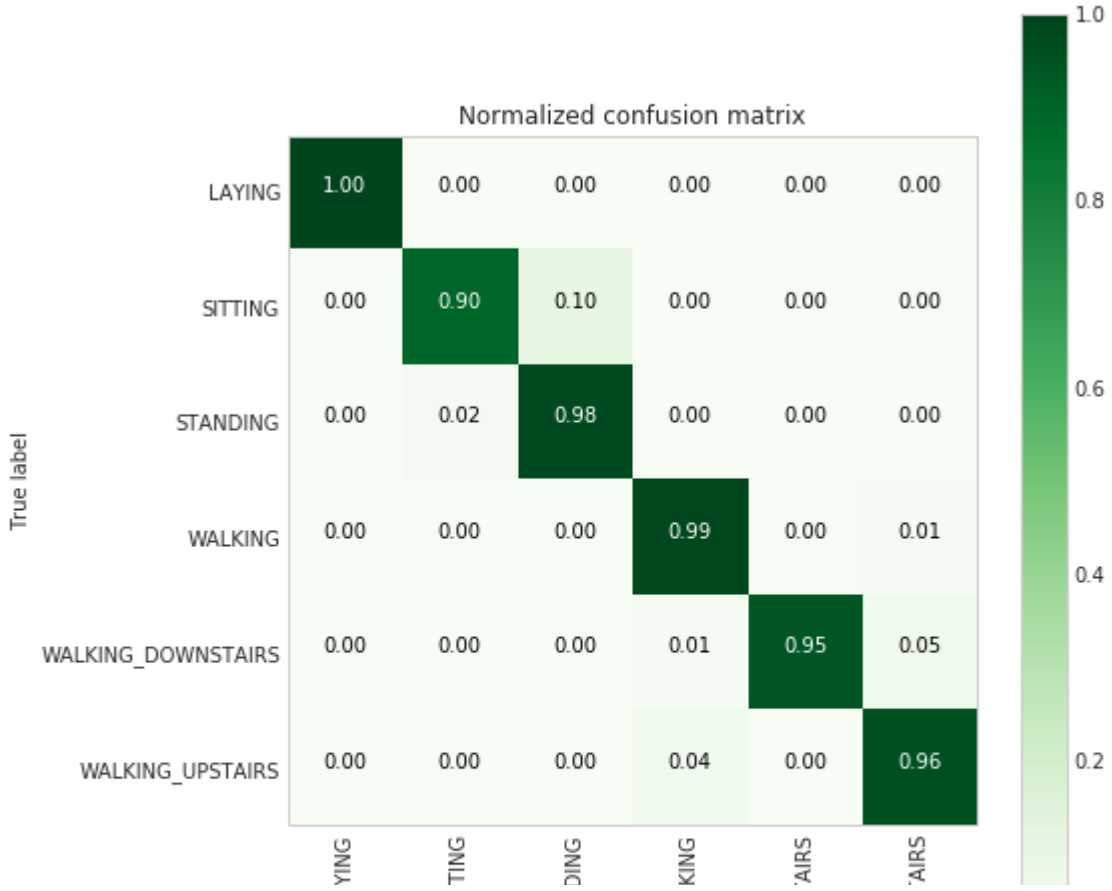
```
-------------------------
|      Best Estimator     |
-------------------------

        LinearSVC(C=0.5, 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': 0.5}

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

        Total numbre of cross validation sets: 3

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

        Average Cross Validate scores of best estimator :

        0.9458650707290533
```

3. Kernel SVM with GridSearch

```
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=la
```

```
training the model..
Done


training_time(HH:MM:SS.ms) - 0:11:40.677845


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:05.421593


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

    0.9626739056667798


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

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




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

            LAYING       1.00      1.00      1.00       537
           SITTING       0.97      0.90      0.93       491
          STANDING       0.92      0.98      0.95       532
```

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

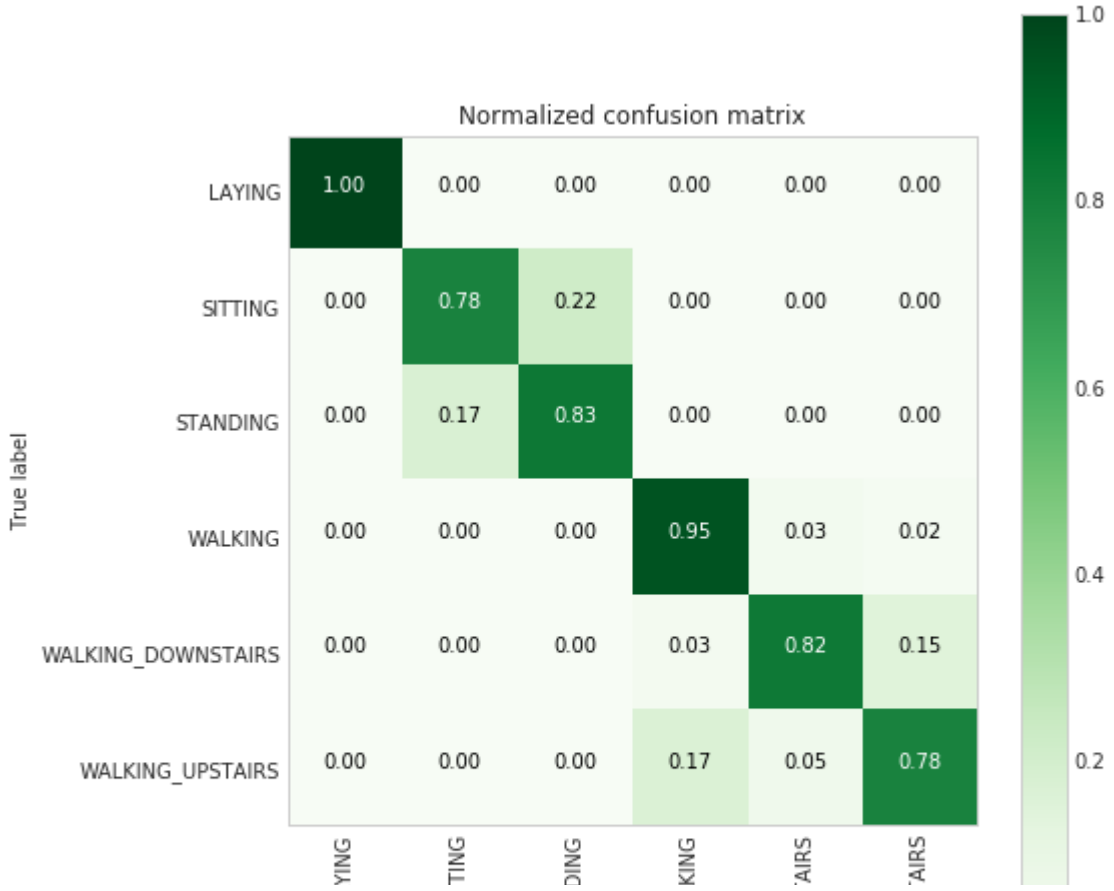
```
-------------------------
|      Best Estimator     |
-------------------------

        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

```
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) - 0:00:26.000829


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.012968


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

    0.8642687478791992


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

 [[537   0   0   0   0   0]
 [  0 385 106   0   0   0]
 [  0  93 439   0   0   0]
 [  0   0   0 471  17   8]
 [  0   0   0  13 346  61]
 [  0   0   0  78  24 369]]
```



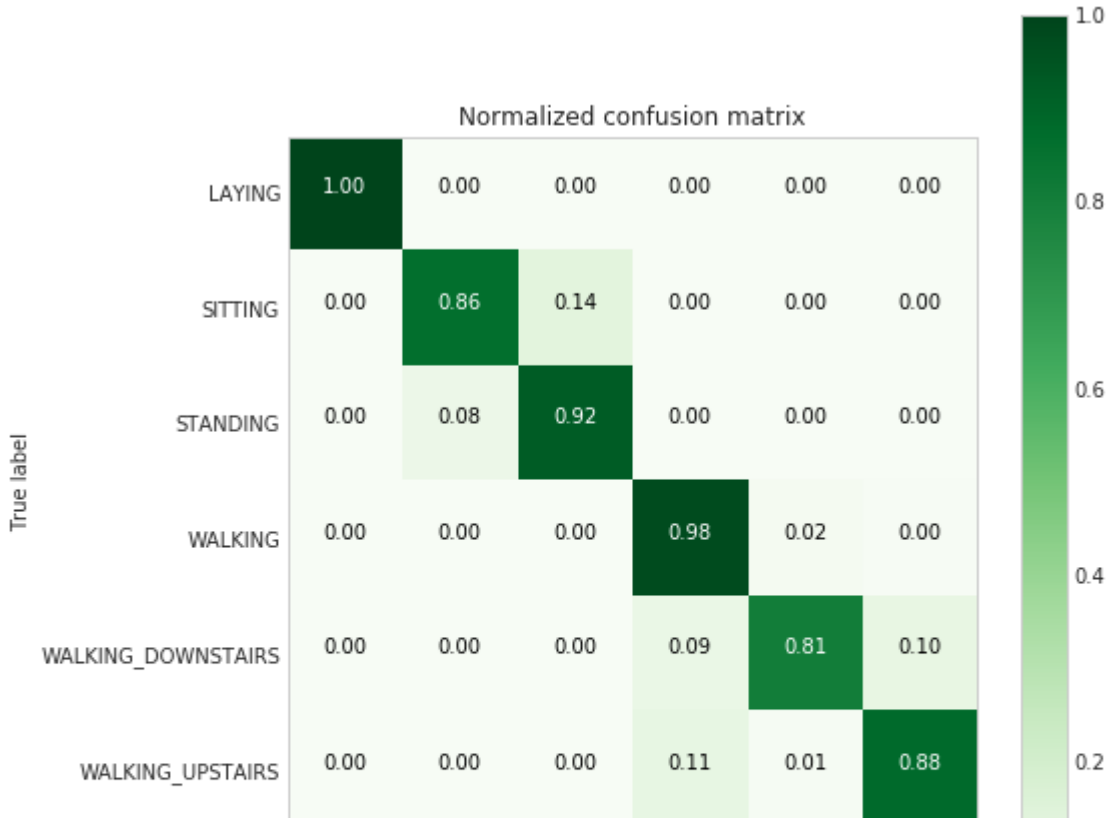

```
-------------------------
| Classifiction Report |
-------------------------
                    precision    recall  f1-score   support

           LAYING       1.00      1.00      1.00       537
          SITTING       0.81      0.78      0.79       491
         STANDING       0.81      0.83      0.82       532
          WALKING       0.84      0.95      0.89       496
WALKING_DOWNSTAIRS       0.89      0.82      0.86       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       |
```

5. Random Forest Classifier with GridSearch

```
0.9409564744427737
```

```
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'])
```

```
C:\Users\BALARAMI REDDY\Anaconda3\lib\site-packages\sklearn\ensemble\weight_boosting.py:
  from numpy.core.umath_tests import inner1d
training the model..
Done


training_time(HH:MM:SS.ms) - 0:12:14.172213


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.130223


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

    0.9127926705123854


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

 [[537   0   0   0   0   0]
 [  0 424  67   0   0   0]
 [  0  40 492   0   0   0]
 [  0   0   0 484  10   2]
 [  0   0   0  37 339  44]
 [  0   0   0  51   6 414]]
```




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

           LAYING       1.00      1.00      1.00       537
          SITTING       0.91      0.86      0.89       491
         STANDING       0.88      0.92      0.90       532
          WALKING       0.85      0.98      0.91       496
WALKING_DOWNSTAIRS       0.95      0.81      0.87       420
  WALKING_UPSTAIRS       0.90      0.88      0.89       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=150, 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': 150}

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

        Total numbre of cross validation sets: 3

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

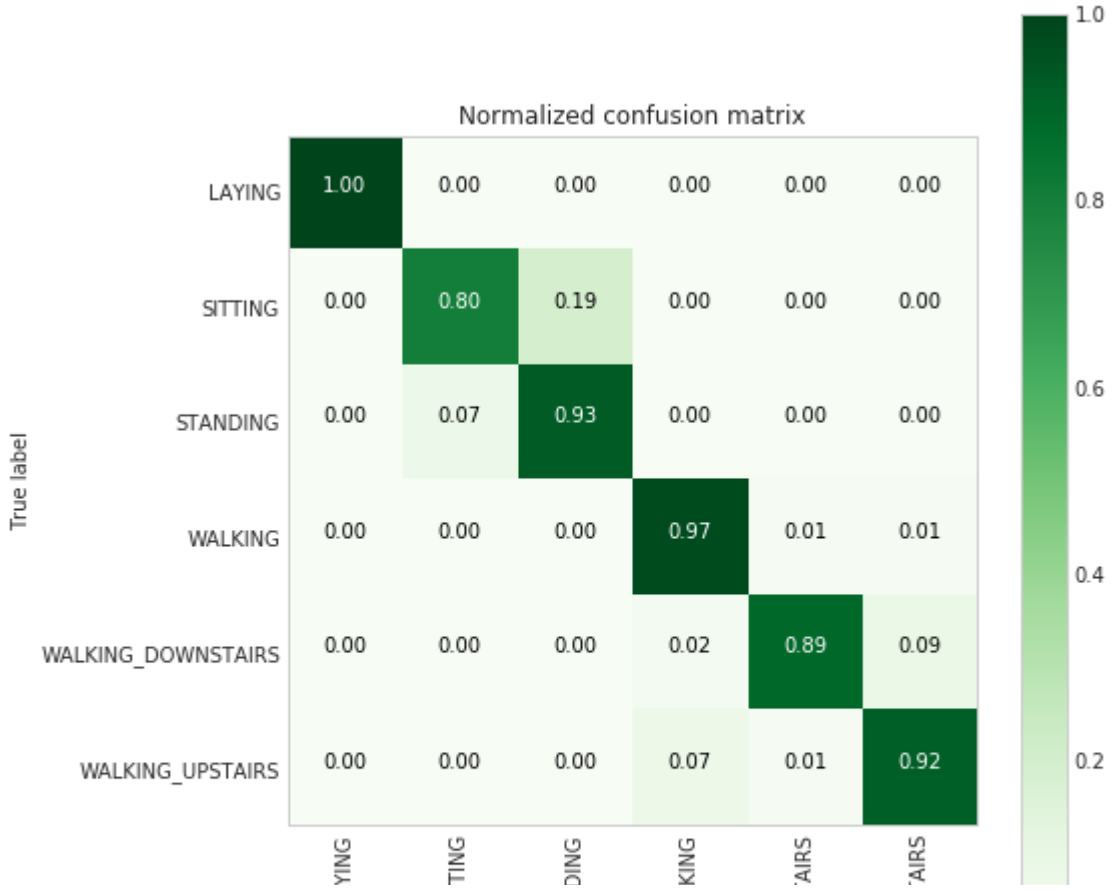
6. Gradient Boosted Decision Trees With GridSearch

```
0.9149891186071817
```

```
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) - 1:23:12.742865


Predicting test data
Done


testing time(HH:MM:SS:ms) - 0:00:00.168552


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

    0.9212758737699356


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

 [[537   0   0   0   0   0]
 [  0 395  95   0   0   1]
 [  0  38 494   0   0   0]
 [  0   0   0 483   7   6]
 [  0   0   0  10 374  36]
 [  0   1   0  32   6 432]]
```




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

           LAYING       1.00      1.00      1.00       537
          SITTING       0.91      0.80      0.85       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=130,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False)

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

        {'max_depth': 5, 'n_estimators': 130}

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

        Total numbre of cross validation sets: 3

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

7. Comparing all models

```
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'] * 10

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          : 96.71%       3.291%
     rbf SVM classifier  : 96.27%      3.733%
     DecisionTree        : 86.43%      13.57%
     Random Forest       : 91.28%      8.721%
     GradientBoosting DT : 91.28%      8.721%
```

Deep Learning Models

```
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'])
```

```
# Run this cell to mount your Google Drive.
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=9473189

Enter your authorization code:
··········
```

```
!ls /content/drive/My\ Drive/Colab\ Notebooks
```

```
 database.sqlite  'Logistic Regression.ipynb'   SGD.ipynb
 Db-IMDB.db        model.py                    'sql_question (1).pdf'
 driving_data.py   NB.ipynb                     SVM.ipynb
 driving_dataset  'Netflix (1).ipynb'           TruncatedSVD.ipynb
 DT.ipynb          Reviews.csv                  UCI_HAR_Dataset
 imdb.db           RF.ipynb                     Untitled2.ipynb
 KNN.ipynb         Self_Driving_Car.ipynb       Untitled.ipynb
```

```
# Data directory
#data=pd.read_csv('/content/drive/My Drive/Colab Notebooks/Reviews.csv')
DATADIR = '/content/drive/My Drive/Colab Notebooks/UCI_HAR_Dataset'
```

```
ls /content/drive/My\ Drive/Colab\ Notebooks \ UCI_HAR_Dataset
```

```
ls: cannot access ' UCI_HAR_Dataset': No such file or directory
'/content/drive/My Drive/Colab Notebooks':
 database.sqlite   'Logistic Regression.ipynb'   SGD.ipynb
 Db-IMDB.db         model.py                    'sql_question (1).pdf'
 driving_data.py    NB.ipynb                     SVM.ipynb
 driving_dataset/  'Netflix (1).ipynb'           TruncatedSVD.ipynb
 DT.ipynb           Reviews.csv                  UCI_HAR_Dataset/
 imdb.db            RF.ipynb                     Untitled2.ipynb
 KNN.ipynb          Self_Driving_Car.ipynb       Untitled.ipynb
```

```
# 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/Colab Notebooks/UCI_HAR_Dataset/{subset}/Inertial Signa
        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/Colab Notebooks/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


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


# 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
from keras.layers.normalization import BatchNormalization


# 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: Method .
  # This is added back by InteractiveShellApp.init_path()
```

```
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
```

LSTM

LSTM with single layer

```
epochs = 20
batch_size = 32
n_hidden = 128
pv = 0.5


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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_5 (LSTM)                (None, 128)               70656     
_________________________________________________________________
batch_normalization_3 (Batch (None, 128)               512       
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 774       
```

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

```
_________________________________________________________________
```

```
# Training the model
histiry_1 = 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 1/20
7352/7352 [==============================] - 39s 5ms/step - loss: 1.1075 - acc: 0.5030 -
Epoch 2/20
7352/7352 [==============================] - 37s 5ms/step - loss: 0.8218 - acc: 0.6306 -
Epoch 3/20
7352/7352 [==============================] - 37s 5ms/step - loss: 0.7112 - acc: 0.6447 -
Epoch 4/20
7352/7352 [==============================] - 37s 5ms/step - loss: 0.6794 - acc: 0.6351 -
Epoch 5/20
7352/7352 [==============================] - 37s 5ms/step - loss: 0.6395 - acc: 0.6635 -
Epoch 6/20
7352/7352 [==============================] - 38s 5ms/step - loss: 0.6385 - acc: 0.6608 -
Epoch 7/20
7352/7352 [==============================] - 37s 5ms/step - loss: 0.5359 - acc: 0.7417 -
Epoch 8/20
7352/7352 [==============================] - 36s 5ms/step - loss: 0.3366 - acc: 0.8855 -
Epoch 9/20
7352/7352 [==============================] - 36s 5ms/step - loss: 0.2234 - acc: 0.9211 -
Epoch 10/20
7352/7352 [==============================] - 37s 5ms/step - loss: 0.1852 - acc: 0.9346 -
Epoch 11/20
7352/7352 [==============================] - 36s 5ms/step - loss: 0.1590 - acc: 0.9354 -
Epoch 12/20
7352/7352 [==============================] - 37s 5ms/step - loss: 0.1547 - acc: 0.9391 -
Epoch 13/20
7352/7352 [==============================] - 37s 5ms/step - loss: 0.1752 - acc: 0.9320 -
Epoch 14/20
7352/7352 [==============================] - 36s 5ms/step - loss: 0.1544 - acc: 0.9399 -
Epoch 15/20
7352/7352 [==============================] - 37s 5ms/step - loss: 0.1489 - acc: 0.9418 -
Epoch 16/20
7352/7352 [==============================] - 37s 5ms/step - loss: 0.1518 - acc: 0.9406 -
Epoch 17/20
7352/7352 [==============================] - 36s 5ms/step - loss: 0.1443 - acc: 0.9372 -
```

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

```
Pred                LAYING  SITTING  ...  WALKING_DOWNSTAIRS  WALKING_UPSTAIRS
True                                 ...
LAYING                 537        0  ...                   0                 0
SITTING                  0      328  ...                   0                 2
STANDING                 0       38  ...                   0                 0
WALKING                  0        0  ...                  11                21
WALKING_DOWNSTAIRS       0        0  ...                 418                 1
WALKING_UPSTAIRS         0        0  ...                   0               465

[6 rows x 6 columns]
```

```
score = model.evaluate(X_test, Y_test)
print("Accuracy: %.2f%%" % (score[1]*100))
```

```
2947/2947 [==============================] - 3s 868us/step
Accuracy: 91.75%
```

```
epochs = 20
batch_size = 64
n_hidden = 128
pv = 0.1


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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_1 (LSTM)                (None, 128)               70656     
_________________________________________________________________
batch_normalization_10 (Batc (None, 128)               512       
_________________________________________________________________
dropout_21 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_19 (Dense)             (None, 6)                 774       
=================================================================
Total params: 71,942
Trainable params: 71,686
Non-trainable params: 256
_________________________________________________________________
```

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


# Training the model
histiry_1 = 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 1/20
7352/7352 [==============================] - 30s 4ms/step - loss: 1.0474 - acc: 0.5586 -
Epoch 2/20
7352/7352 [==============================] - 27s 4ms/step - loss: 0.7839 - acc: 0.6587 -
Epoch 3/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.7595 - acc: 0.6538 -
Epoch 4/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.6482 - acc: 0.6785 -
Epoch 5/20
7352/7352 [==============================] - 27s 4ms/step - loss: 0.5855 - acc: 0.6884 -
Epoch 6/20
7352/7352 [==============================] - 27s 4ms/step - loss: 0.5140 - acc: 0.7285 -
Epoch 7/20
7352/7352 [==============================] - 27s 4ms/step - loss: 0.3885 - acc: 0.8294 -
Epoch 8/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.2402 - acc: 0.9195 -
Epoch 9/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1842 - acc: 0.9323 -
Epoch 10/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1914 - acc: 0.9283 -
Epoch 11/20
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1613 - acc: 0.9377 -
Epoch 12/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1533 - acc: 0.9382 -
Epoch 13/20
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1503 - acc: 0.9399 -
Epoch 14/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1468 - acc: 0.9416 -
Epoch 15/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1464 - acc: 0.9459 -
Epoch 16/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1340 - acc: 0.9470 -
Epoch 17/20
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1363 - acc: 0.9455 -
```

```
score = model.evaluate(X_test, Y_test)
print("Accuracy: %.2f%%" % (score[1]*100))
```

```
2947/2947 [==============================] - 3s 914us/step
Accuracy: 93.21%
```

LSTM with 2 Layers

```
epochs1 = 30
batch_size1= 32
n_hidden1 = 128
n_hidden2 =64
pv1 = 0.2
pv2 = 0.5
```

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

```
model1.add(LSTM(n_hidden1, return_sequences=True, input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model1.add(Dropout(pv1))

model1.add(LSTM(n_hidden2))
# Adding a dropout layer
model1.add(Dropout(pv2))
# Adding a dense output layer with sigmoid activation
model1.add(Dense(n_classes, activation='sigmoid'))
model1.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 128, 128)          70656
_________________________________________________________________
dropout_3 (Dropout)          (None, 128, 128)          0
_________________________________________________________________
lstm_4 (LSTM)                (None, 64)                49408
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 390
=================================================================
Total params: 120,454
Trainable params: 120,454
Non-trainable params: 0
_________________________________________________________________
```

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


# Training the model
history = model1.fit(X_train,
                     Y_train,
                     batch_size=batch_size,
                     validation_data=(X_test, Y_test),
                     epochs=epochs1)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 108s 15ms/step - loss: 1.0750 - acc: 0.5275
Epoch 2/30
7352/7352 [==============================] - 105s 14ms/step - loss: 0.7786 - acc: 0.6357
Epoch 3/30
7352/7352 [==============================] - 105s 14ms/step - loss: 0.7163 - acc: 0.6604
Epoch 4/30
7352/7352 [==============================] - 105s 14ms/step - loss: 0.9102 - acc: 0.5891
Epoch 5/30
7352/7352 [==============================] - 107s 15ms/step - loss: 0.6548 - acc: 0.6824
Epoch 6/30
7352/7352 [==============================] - 107s 15ms/step - loss: 0.4596 - acc: 0.8320
Epoch 7/30
7352/7352 [==============================] - 106s 14ms/step - loss: 0.2535 - acc: 0.9173
Epoch 8/30
7352/7352 [==============================] - 105s 14ms/step - loss: 0.2112 - acc: 0.9279
Epoch 9/30
7352/7352 [==============================] - 105s 14ms/step - loss: 0.1675 - acc: 0.9372
Epoch 10/30
7352/7352 [==============================] - 105s 14ms/step - loss: 0.1673 - acc: 0.9449
Epoch 11/30
7352/7352 [==============================] - 104s 14ms/step - loss: 0.1564 - acc: 0.9434
Epoch 12/30
7352/7352 [==============================] - 104s 14ms/step - loss: 0.1624 - acc: 0.9461
Epoch 13/30
7352/7352 [==============================] - 105s 14ms/step - loss: 0.1322 - acc: 0.9487
Epoch 14/30
7352/7352 [==============================] - 105s 14ms/step - loss: 0.1465 - acc: 0.9479
Epoch 15/30
7352/7352 [==============================] - 105s 14ms/step - loss: 0.1391 - acc: 0.9460
Epoch 16/30
7352/7352 [==============================] - 105s 14ms/step - loss: 0.1292 - acc: 0.9513
Epoch 17/30
7352/7352 [==============================] - 106s 14ms/step - loss: 0.1358 - acc: 0.9480
Epoch 18/30
7352/7352 [==============================] - 106s 14ms/step - loss: 0.1322 - acc: 0.9497
Epoch 19/30
7352/7352 [==============================] - 107s 15ms/step - loss: 0.1268 - acc: 0.9512
Epoch 20/30
7352/7352 [==============================] - 106s 14ms/step - loss: 0.1143 - acc: 0.9510
Epoch 21/30
7352/7352 [==============================] - 106s 14ms/step - loss: 0.1680 - acc: 0.9482
Epoch 22/30
7352/7352 [==============================] - 106s 14ms/step - loss: 0.1438 - acc: 0.9464
Epoch 23/30
7352/7352 [==============================] - 106s 14ms/step - loss: 0.1210 - acc: 0.9539
Epoch 24/30
7352/7352 [==============================] - 106s 14ms/step - loss: 0.1498 - acc: 0.9497
Epoch 25/30
7352/7352 [==============================] - 105s 14ms/step - loss: 0.1264 - acc: 0.9538
Epoch 26/30
7352/7352 [==============================] - 105s 14ms/step - loss: 0.1384 - acc: 0.9525
Epoch 27/30
7352/7352 [==============================] - 106s 14ms/step - loss: 0.1419 - acc: 0.9516
```

```
# Confusion Matrix
```

```
print(confusion_matrix(Y_test, model1.predict(X_test)))
```

```
Pred                LAYING  SITTING  ...  WALKING_DOWNSTAIRS  WALKING_UPSTAIRS
True                                 ...
LAYING                 537        0  ...                   0                 0
SITTING                  0      412  ...                   0                 0
STANDING                 0       67  ...                   0                 0
WALKING                  0        0  ...                  20                29
WALKING_DOWNSTAIRS       0        0  ...                 397                17
WALKING_UPSTAIRS         0        0  ...                   0               452

[6 rows x 6 columns]
```

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

```
2947/2947 [==============================] - 4s 1ms/step
```

```
print("Accuracy: %.2f%%" % (score1[1]*100))
```

```
Accuracy: 91.96%
```

```
epochs = 20
batch_size =64
n_hidden = 128

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


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

# Training the model
histiry_1 = model.fit(X_train,
                      Y_train,
                      batch_size=batch_size,
                      validation_data=(X_test, Y_test),
                      epochs=epochs)


print("==============================")
score = model.evaluate(X_test, Y_test)
print("Accuracy: %.2f%%" % (score[1]*100))
```

```
WARNING: Logging before flag parsing goes to stderr.
W0815 11:55:18.505002 139678289241984 deprecation_wrapper.py:119] From /usr/local/lib/py

W0815 11:55:18.508882 139678289241984 deprecation_wrapper.py:119] From /usr/local/lib/py

W0815 11:55:18.520177 139678289241984 deprecation_wrapper.py:119] From /usr/local/lib/py

W0815 11:55:18.813213 139678289241984 deprecation_wrapper.py:119] From /usr/local/lib/py

W0815 11:55:18.835502 139678289241984 deprecation_wrapper.py:119] From /usr/local/lib/py


_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 128)               70656
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 774
=================================================================
Total params: 71,430
Trainable params: 71,430
Non-trainable params: 0
_________________________________________________________________
W0815 11:55:19.133373 139678289241984 deprecation.py:323] From /usr/local/lib/python3.6/
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
W0815 11:55:19.752544 139678289241984 deprecation_wrapper.py:119] From /usr/local/lib/py

Train on 7352 samples, validate on 2947 samples
Epoch 1/20
7352/7352 [==============================] - 28s 4ms/step - loss: 1.3094 - acc: 0.4195 -
Epoch 2/20
7352/7352 [==============================] - 26s 4ms/step - loss: 1.2410 - acc: 0.4531 -
Epoch 3/20
7352/7352 [==============================] - 26s 4ms/step - loss: 1.1656 - acc: 0.4976 -
Epoch 4/20
7352/7352 [==============================] - 26s 4ms/step - loss: 1.0044 - acc: 0.5832 -
Epoch 5/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.7302 - acc: 0.7175 -
Epoch 6/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.5062 - acc: 0.8240 -
Epoch 7/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.3289 - acc: 0.8829 -
Epoch 8/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.2540 - acc: 0.9101 -
Epoch 9/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.2403 - acc: 0.9113 -
Epoch 10/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.2179 - acc: 0.9131 -
Epoch 11/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.2469 - acc: 0.9040 -
Epoch 12/20
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1936 - acc: 0.9267 -
Epoch 13/20
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1708 - acc: 0.9317 -
Epoch 14/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1699 - acc: 0.9306 -
Epoch 15/20
```

```
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1648 - acc: 0.9407 -
Epoch 16/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1712 - acc: 0.9403 -
Epoch 17/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1521 - acc: 0.9430 -
Epoch 18/20
7352/7352 [==============================] - 27s 4ms/step - loss: 0.1689 - acc: 0.9353 -
Epoch 19/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1639 - acc: 0.9393 -
Epoch 20/20
7352/7352 [==============================] - 26s 4ms/step - loss: 0.1521 - acc: 0.9373 -
==============================
2947/2947 [==============================] - 3s 870us/step
Accuracy: 90.16%
```

```
epochs1 = 30
batch_size1= 64
n_hidden1 = 128
n_hidden2 =128
pv1 = 0.1
pv2 = 0.1

# Initiliazing the sequential model
model1 = Sequential()
# Configuring the parameters
model1.add(LSTM(n_hidden1, return_sequences=True, input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model1.add(Dropout(pv1))

model1.add(LSTM(n_hidden2))
# Adding a dropout layer
model1.add(Dropout(pv2))
# Adding a dense output layer with sigmoid activation
model1.add(Dense(n_classes, activation='sigmoid'))
model1.summary()


# Compiling the model
model1.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
# Training the model
history = model1.fit(X_train,
                     Y_train,
                     batch_size=batch_size,
                     validation_data=(X_test, Y_test),
                     epochs=epochs1)

print("==============================")
score = model.evaluate(X_test, Y_test)
print("Accuracy: %.2f%%" % (score[1]*100))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_6 (LSTM)                (None, 128, 128)          70656     
_________________________________________________________________
dropout_5 (Dropout)          (None, 128, 128)          0         
_________________________________________________________________
lstm_7 (LSTM)                (None, 128)               131584    
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 774       
=================================================================
Total params: 203,014
Trainable params: 203,014
Non-trainable params: 0
_________________________________________________________________
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 71s 10ms/step - loss: 1.1304 - acc: 0.5045
Epoch 2/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.7577 - acc: 0.6594 -
Epoch 3/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.6582 - acc: 0.7169 -
Epoch 4/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.5248 - acc: 0.7923 -
Epoch 5/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.4366 - acc: 0.8372 -
Epoch 6/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.3094 - acc: 0.8862 -
Epoch 7/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.2439 - acc: 0.9138 -
Epoch 8/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1832 - acc: 0.9328 -
Epoch 9/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1789 - acc: 0.9353 -
Epoch 10/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1585 - acc: 0.9402 -
Epoch 11/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1393 - acc: 0.9444 -
Epoch 12/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1408 - acc: 0.9404 -
Epoch 13/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1486 - acc: 0.9438 -
Epoch 14/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1360 - acc: 0.9471 -
Epoch 15/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1383 - acc: 0.9475 -
Epoch 16/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1315 - acc: 0.9429 -
Epoch 17/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1290 - acc: 0.9516 -
Epoch 18/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1228 - acc: 0.9493 -
Epoch 19/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1188 - acc: 0.9517 -
Epoch 20/30
```

```
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1227 - acc: 0.9478 -
Epoch 21/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1156 - acc: 0.9531 -
Epoch 22/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1257 - acc: 0.9524 -
Epoch 23/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1241 - acc: 0.9457 -
Epoch 24/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1187 - acc: 0.9493 -
Epoch 25/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1190 - acc: 0.9499 -
Epoch 26/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1082 - acc: 0.9553 -
Epoch 27/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1182 - acc: 0.9532 -
Epoch 28/30
7352/7352 [==============================] - 69s 9ms/step - loss: 0.1145 - acc: 0.9528 -
Epoch 29/30
```

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

print("Accuracy: %.2f%%" % (score1[1]*100))
```

```
2947/2947 [==============================] - 6s 2ms/step
Accuracy: 92.43%
```

CNN

CNN Single Layer

```
model = Sequential()
model.add(Conv1D(32, kernel_size=3,
                 activation='relu',
                 input_shape=(timesteps, input_dim)))
model.add(Conv1D(64, 3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(n_classes, activation='softmax'))

batch_size = 128
epochs = 30

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


histiry_1 = 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 [==============================] - 7s 888us/step - loss: 0.7618 - acc: 0.6793
Epoch 2/30
7352/7352 [==============================] - 5s 676us/step - loss: 0.4210 - acc: 0.8341
Epoch 3/30
7352/7352 [==============================] - 5s 683us/step - loss: 0.2678 - acc: 0.9048
Epoch 4/30
7352/7352 [==============================] - 5s 677us/step - loss: 0.1826 - acc: 0.9327
Epoch 5/30
7352/7352 [==============================] - 5s 679us/step - loss: 0.1508 - acc: 0.9402
Epoch 6/30
7352/7352 [==============================] - 5s 679us/step - loss: 0.1332 - acc: 0.9467
Epoch 7/30
7352/7352 [==============================] - 5s 681us/step - loss: 0.1289 - acc: 0.9489
Epoch 8/30
7352/7352 [==============================] - 5s 683us/step - loss: 0.1228 - acc: 0.9501
Epoch 9/30
7352/7352 [==============================] - 5s 669us/step - loss: 0.1124 - acc: 0.9506
Epoch 10/30
7352/7352 [==============================] - 5s 692us/step - loss: 0.1102 - acc: 0.9544
Epoch 11/30
7352/7352 [==============================] - 5s 691us/step - loss: 0.1089 - acc: 0.9524
Epoch 12/30
7352/7352 [==============================] - 5s 682us/step - loss: 0.1030 - acc: 0.9561
Epoch 13/30
7352/7352 [==============================] - 5s 680us/step - loss: 0.1036 - acc: 0.9566
Epoch 14/30
7352/7352 [==============================] - 5s 682us/step - loss: 0.1016 - acc: 0.9555
Epoch 15/30
7352/7352 [==============================] - 5s 682us/step - loss: 0.0979 - acc: 0.9563
Epoch 16/30
7352/7352 [==============================] - 5s 679us/step - loss: 0.0926 - acc: 0.9567
Epoch 17/30
7352/7352 [==============================] - 5s 687us/step - loss: 0.0934 - acc: 0.9585
Epoch 18/30
7352/7352 [==============================] - 5s 690us/step - loss: 0.0878 - acc: 0.9577
Epoch 19/30
7352/7352 [==============================] - 5s 680us/step - loss: 0.0855 - acc: 0.9610
Epoch 20/30
7352/7352 [==============================] - 5s 686us/step - loss: 0.0848 - acc: 0.9608
Epoch 21/30
7352/7352 [==============================] - 5s 678us/step - loss: 0.0814 - acc: 0.9616
Epoch 22/30
7352/7352 [==============================] - 5s 671us/step - loss: 0.0808 - acc: 0.9625
Epoch 23/30
7352/7352 [==============================] - 5s 694us/step - loss: 0.0740 - acc: 0.9638
Epoch 24/30
7352/7352 [==============================] - 5s 703us/step - loss: 0.0752 - acc: 0.9652
Epoch 25/30
7352/7352 [==============================] - 5s 682us/step - loss: 0.0747 - acc: 0.9634
Epoch 26/30
7352/7352 [==============================] - 5s 691us/step - loss: 0.0686 - acc: 0.9660
Epoch 27/30
7352/7352 [==============================] - 5s 676us/step - loss: 0.0708 - acc: 0.9646
Epoch 28/30
7352/7352 [==============================] - 5s 692us/step - loss: 0.0666 - acc: 0.9709
```

```
Epoch 29/30
7352/7352 [==============================] - 5s 678us/step - loss: 0.0691 - acc: 0.9665
Epoch 30/30
7352/7352 [==============================] - 5s 688us/step - loss: 0.0678 - acc: 0.9695
```

```
score = model.evaluate(X_test, Y_test, verbose=0)
```

```
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.6347342224354259
Test accuracy: 0.9049881235154394
```

```
model = Sequential()
model.add(Conv1D(64, kernel_size=3,
                 activation='relu',
                 input_shape=(timesteps, input_dim)))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(n_classes, activation='softmax'))

batch_size = 128
epochs = 30

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


histiry_1 = model.fit(X_train,
                      Y_train,
                      batch_size=batch_size,
                      validation_data=(X_test, Y_test),
                      epochs=epochs)
```

```
W0813 16:18:47.958996 140532760672128 deprecation_wrapper.py:119] From /usr/local/lib/py

W0813 16:18:47.983665 140532760672128 deprecation_wrapper.py:119] From /usr/local/lib/py

W0813 16:18:48.129502 140532760672128 deprecation.py:323] From /usr/local/lib/python3.6/
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 4s 601us/step - loss: 0.7612 - acc: 0.6915
Epoch 2/30
7352/7352 [==============================] - 4s 484us/step - loss: 0.4243 - acc: 0.8337
Epoch 3/30
7352/7352 [==============================] - 4s 483us/step - loss: 0.3007 - acc: 0.8863
Epoch 4/30
7352/7352 [==============================] - 4s 490us/step - loss: 0.2313 - acc: 0.9136
Epoch 5/30
7352/7352 [==============================] - 3s 463us/step - loss: 0.1959 - acc: 0.9260
Epoch 6/30
7352/7352 [==============================] - 3s 465us/step - loss: 0.1714 - acc: 0.9340
Epoch 7/30
7352/7352 [==============================] - 3s 465us/step - loss: 0.1472 - acc: 0.9418
Epoch 8/30
7352/7352 [==============================] - 3s 467us/step - loss: 0.1428 - acc: 0.9407
Epoch 9/30
7352/7352 [==============================] - 3s 457us/step - loss: 0.1359 - acc: 0.9463
Epoch 10/30
7352/7352 [==============================] - 3s 458us/step - loss: 0.1296 - acc: 0.9486
Epoch 11/30
7352/7352 [==============================] - 3s 460us/step - loss: 0.1216 - acc: 0.9476
Epoch 12/30
7352/7352 [==============================] - 3s 461us/step - loss: 0.1195 - acc: 0.9486
Epoch 13/30
7352/7352 [==============================] - 3s 457us/step - loss: 0.1164 - acc: 0.9524
Epoch 14/30
7352/7352 [==============================] - 3s 464us/step - loss: 0.1116 - acc: 0.9532
Epoch 15/30
7352/7352 [==============================] - 3s 454us/step - loss: 0.1079 - acc: 0.9542
Epoch 16/30
7352/7352 [==============================] - 3s 471us/step - loss: 0.1050 - acc: 0.9563
Epoch 17/30
7352/7352 [==============================] - 3s 460us/step - loss: 0.1053 - acc: 0.9566
Epoch 18/30
7352/7352 [==============================] - 3s 460us/step - loss: 0.1016 - acc: 0.9562
Epoch 19/30
7352/7352 [==============================] - 3s 457us/step - loss: 0.0985 - acc: 0.9591
Epoch 20/30
7352/7352 [==============================] - 3s 462us/step - loss: 0.0982 - acc: 0.9569
Epoch 21/30
7352/7352 [==============================] - 3s 451us/step - loss: 0.0957 - acc: 0.9581
Epoch 22/30
7352/7352 [==============================] - 3s 468us/step - loss: 0.0912 - acc: 0.9601
Epoch 23/30
7352/7352 [==============================] - 3s 457us/step - loss: 0.0898 - acc: 0.9581
Epoch 24/30
7352/7352 [==============================] - 3s 463us/step - loss: 0.0855 - acc: 0.9619
Epoch 25/30
```

```
7352/7352 [==============================] - 3s 456us/step - loss: 0.0872 - acc: 0.9608
Epoch 26/30
7352/7352 [==============================] - 3s 451us/step - loss: 0.0859 - acc: 0.9640
Epoch 27/30
7352/7352 [==============================] - 3s 460us/step - loss: 0.0826 - acc: 0.9645
Epoch 28/30
```

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

```
Test loss: 0.3605819721979393
Test accuracy: 0.9172039362063115
```

```
model = Sequential()
model.add(Conv1D(64, kernel_size=3,
                 activation='relu',
                 input_shape=(timesteps, input_dim)))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(n_classes, activation='softmax'))

batch_size = 128
epochs = 25

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


histiry_1 = 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 1/25
7352/7352 [==============================] - 4s 529us/step - loss: 0.7192 - acc: 0.6968
Epoch 2/25
7352/7352 [==============================] - 3s 442us/step - loss: 0.3579 - acc: 0.8604
Epoch 3/25
7352/7352 [==============================] - 3s 434us/step - loss: 0.2416 - acc: 0.9075
Epoch 4/25
7352/7352 [==============================] - 3s 448us/step - loss: 0.1793 - acc: 0.9359
Epoch 5/25
7352/7352 [==============================] - 3s 448us/step - loss: 0.1567 - acc: 0.9402
Epoch 6/25
7352/7352 [==============================] - 3s 445us/step - loss: 0.1387 - acc: 0.9453
Epoch 7/25
7352/7352 [==============================] - 3s 445us/step - loss: 0.1233 - acc: 0.9506
Epoch 8/25
7352/7352 [==============================] - 3s 443us/step - loss: 0.1177 - acc: 0.9498
Epoch 9/25
7352/7352 [==============================] - 3s 445us/step - loss: 0.1087 - acc: 0.9563
Epoch 10/25
7352/7352 [==============================] - 3s 447us/step - loss: 0.1049 - acc: 0.9566
Epoch 11/25
7352/7352 [==============================] - 3s 448us/step - loss: 0.0989 - acc: 0.9550
Epoch 12/25
7352/7352 [==============================] - 3s 444us/step - loss: 0.0981 - acc: 0.9570
Epoch 13/25
7352/7352 [==============================] - 3s 438us/step - loss: 0.0931 - acc: 0.9576
Epoch 14/25
7352/7352 [==============================] - 3s 436us/step - loss: 0.0874 - acc: 0.9625
Epoch 15/25
7352/7352 [==============================] - 3s 447us/step - loss: 0.0856 - acc: 0.9623
Epoch 16/25
7352/7352 [==============================] - 3s 445us/step - loss: 0.0833 - acc: 0.9638
Epoch 17/25
7352/7352 [==============================] - 3s 451us/step - loss: 0.0814 - acc: 0.9653
Epoch 18/25
7352/7352 [==============================] - 3s 439us/step - loss: 0.0773 - acc: 0.9669
Epoch 19/25
7352/7352 [==============================] - 3s 442us/step - loss: 0.0724 - acc: 0.9674
Epoch 20/25
7352/7352 [==============================] - 3s 449us/step - loss: 0.0714 - acc: 0.9679
Epoch 21/25
7352/7352 [==============================] - 3s 458us/step - loss: 0.0699 - acc: 0.9697
Epoch 22/25
7352/7352 [==============================] - 3s 456us/step - loss: 0.0646 - acc: 0.9733
```

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

```
Test loss: 0.4078495118270636
Test accuracy: 0.9195792331184255
```

CNN with 2 Layers

```
batch_size = 128
epochs = 30

model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(timesteps, input_dim)))
model.add(MaxPooling1D(pool_size=2))


model.add(Conv1D(64,kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.1))

#model.add(Conv1D(64,kernel_size=3, activation='relu',padding = 'same'))
#model.add(MaxPooling1D(pool_size=2))
#model.add(BatchNormalization())
#model.add(Dropout(0.5))



model.add(Flatten())
model.add(Dense(128, activation='softmax'))
model.add(Dropout(0.1))
model.add(Dense(n_classes, activation='softmax'))

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


histiry_1 = 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 [==============================] - 7s 935us/step - loss: 1.5689 - acc: 0.7195
Epoch 2/30
7352/7352 [==============================] - 5s 703us/step - loss: 1.4746 - acc: 0.7727
Epoch 3/30
7352/7352 [==============================] - 5s 708us/step - loss: 1.3924 - acc: 0.8200
Epoch 4/30
7352/7352 [==============================] - 5s 721us/step - loss: 1.3055 - acc: 0.8666
Epoch 5/30
7352/7352 [==============================] - 5s 708us/step - loss: 1.2313 - acc: 0.8681
Epoch 6/30
7352/7352 [==============================] - 5s 719us/step - loss: 1.1592 - acc: 0.8719
Epoch 7/30
7352/7352 [==============================] - 5s 711us/step - loss: 1.0928 - acc: 0.8708
Epoch 8/30
7352/7352 [==============================] - 5s 712us/step - loss: 1.0344 - acc: 0.8708
Epoch 9/30
7352/7352 [==============================] - 5s 716us/step - loss: 0.9620 - acc: 0.8814
Epoch 10/30
7352/7352 [==============================] - 5s 717us/step - loss: 0.9045 - acc: 0.8830
Epoch 11/30
7352/7352 [==============================] - 5s 727us/step - loss: 0.8491 - acc: 0.8817
Epoch 12/30
7352/7352 [==============================] - 5s 723us/step - loss: 0.8050 - acc: 0.8760
Epoch 13/30
7352/7352 [==============================] - 5s 721us/step - loss: 0.7610 - acc: 0.8776
Epoch 14/30
7352/7352 [==============================] - 5s 722us/step - loss: 0.7205 - acc: 0.8792
Epoch 15/30
7352/7352 [==============================] - 5s 711us/step - loss: 0.6765 - acc: 0.8810
Epoch 16/30
7352/7352 [==============================] - 5s 720us/step - loss: 0.6462 - acc: 0.8751
Epoch 17/30
7352/7352 [==============================] - 5s 724us/step - loss: 0.6107 - acc: 0.8761
Epoch 18/30
7352/7352 [==============================] - 5s 714us/step - loss: 0.5871 - acc: 0.8732
Epoch 19/30
7352/7352 [==============================] - 5s 709us/step - loss: 0.5422 - acc: 0.8823
Epoch 20/30
7352/7352 [==============================] - 5s 714us/step - loss: 0.5161 - acc: 0.8841
Epoch 21/30
7352/7352 [==============================] - 5s 730us/step - loss: 0.5012 - acc: 0.8785
Epoch 22/30
7352/7352 [==============================] - 5s 725us/step - loss: 0.4862 - acc: 0.8774
Epoch 23/30
7352/7352 [==============================] - 5s 717us/step - loss: 0.4503 - acc: 0.8879
Epoch 24/30
7352/7352 [==============================] - 5s 718us/step - loss: 0.4425 - acc: 0.8806
Epoch 25/30
7352/7352 [==============================] - 5s 716us/step - loss: 0.4288 - acc: 0.8818
Epoch 26/30
7352/7352 [==============================] - 5s 729us/step - loss: 0.4132 - acc: 0.8823
Epoch 27/30
7352/7352 [==============================] - 5s 721us/step - loss: 0.4059 - acc: 0.8755
```

```
score = model.evaluate(X_test, Y_test, verbose=0)
```

```
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.43721131847436284
Test accuracy: 0.8958262639972854
```

CNN with 3 Layers

```
batch_size = 128
epochs = 30

model = Sequential()
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(timesteps, input_dim)))
model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
model.add(Dropout(0.5))
model.add(MaxPooling1D(pool_size=2))


model.add(Conv1D(64,kernel_size=3, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Conv1D(64,kernel_size=3, activation='relu',padding = 'same'))
model.add(MaxPooling1D(pool_size=2))
model.add(BatchNormalization())
model.add(Dropout(0.5))



model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(n_classes, activation='softmax'))

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


histiry_1 = 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 [==============================] - 13s 2ms/step - loss: 1.0519 - acc: 0.6640 -
Epoch 2/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.4149 - acc: 0.8410 -
Epoch 3/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.2296 - acc: 0.9157 -
Epoch 4/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.1925 - acc: 0.9278 -
Epoch 5/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.1749 - acc: 0.9343 -
Epoch 6/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.1639 - acc: 0.9380 -
Epoch 7/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.1587 - acc: 0.9354 -
Epoch 8/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.1618 - acc: 0.9380 -
Epoch 9/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.1537 - acc: 0.9410 -
Epoch 10/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.1373 - acc: 0.9448 -
Epoch 11/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.1529 - acc: 0.9418 -
Epoch 12/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.1412 - acc: 0.9452 -
Epoch 13/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.1671 - acc: 0.9399 -
Epoch 14/30
7352/7352 [==============================] - 10s 1ms/step - loss: 0.1573 - acc: 0.9385 -
Epoch 15/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.1684 - acc: 0.9396 -
Epoch 16/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.1907 - acc: 0.9374 -
Epoch 17/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.2303 - acc: 0.9358 -
Epoch 18/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.3119 - acc: 0.9305 -
Epoch 19/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.3697 - acc: 0.9306 -
Epoch 20/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.4910 - acc: 0.9309 -
Epoch 21/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.5950 - acc: 0.9374 -
Epoch 22/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.6405 - acc: 0.9368 -
Epoch 23/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.7187 - acc: 0.9415 -
Epoch 24/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.7384 - acc: 0.9442 -
Epoch 25/30
7352/7352 [==============================] - 10s 1ms/step - loss: 0.7416 - acc: 0.9453 -
Epoch 26/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.7733 - acc: 0.9455 -
Epoch 27/30
7352/7352 [==============================] - 11s 1ms/step - loss: 0.7626 - acc: 0.9475 -
```

```
score = model.evaluate(X_test, Y_test, verbose=0)
```

```
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 1.065255669042026
Test accuracy: 0.8971835765184933
```

CNN with 5 layers

```
model = Sequential()

model.add(Conv1D(64, kernel_size=2,activation='relu',input_shape=(timesteps, input_dim)))
model.add(MaxPooling1D(pool_size=1))

model.add(Conv1D(64,kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Dropout(0.5))

#third layer
model.add(Conv1D(64,kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=1))
model.add(BatchNormalization())
model.add(Dropout(0.5))

#Fourth Layer
model.add(Conv1D(64,kernel_size=2, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(BatchNormalization())
model.add(Dropout(0.5))

#Fifth Layer
model.add(Conv1D(128,kernel_size=5, activation='relu',padding = 'same'))
model.add(MaxPooling1D(pool_size=3))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(n_classes, activation='softmax'))
```

```
batch_size = 128
epochs = 30

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


histiry_1 = 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 [==============================] - 16s 2ms/step - loss: 1.2523 - acc: 0.6171 -
Epoch 2/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.7354 - acc: 0.7499 -
Epoch 3/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.4343 - acc: 0.8508 -
Epoch 4/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.2973 - acc: 0.8924 -
Epoch 5/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.2328 - acc: 0.9125 -
Epoch 6/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.2319 - acc: 0.9203 -
Epoch 7/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.2012 - acc: 0.9293 -
Epoch 8/30
7352/7352 [==============================] - 12s 2ms/step - loss: 0.2052 - acc: 0.9248 -
Epoch 9/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.1862 - acc: 0.9348 -
Epoch 10/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.1980 - acc: 0.9264 -
Epoch 11/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.2326 - acc: 0.9242 -
Epoch 12/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.3221 - acc: 0.9229 -
Epoch 13/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.3943 - acc: 0.9229 -
Epoch 14/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.4820 - acc: 0.9212 -
Epoch 15/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.5948 - acc: 0.9246 -
Epoch 16/30
7352/7352 [==============================] - 12s 2ms/step - loss: 0.5923 - acc: 0.9321 -
Epoch 17/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.7082 - acc: 0.9336 -
Epoch 18/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.7904 - acc: 0.9335 -
Epoch 19/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.7842 - acc: 0.9378 -
Epoch 20/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.8843 - acc: 0.9351 -
Epoch 21/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.9562 - acc: 0.9282 -
Epoch 22/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.8606 - acc: 0.9355 -
Epoch 23/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.8159 - acc: 0.9393 -
Epoch 24/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.8531 - acc: 0.9370 -
Epoch 25/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.8397 - acc: 0.9404 -
Epoch 26/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.8095 - acc: 0.9430 -
Epoch 27/30
7352/7352 [==============================] - 13s 2ms/step - loss: 0.8464 - acc: 0.9397 -
```

```
score = model.evaluate(X_test, Y_test, verbose=0)
```

```
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 4.999360807364286
Test accuracy: 0.675941635561588
```

Performance Table

```
print("                              ACC")
print("                             ======")
print(" LSTM with single layer    " , 93.21 )
print(" LSTM with 2 lyers         "  ,92.43 )
print("CNN with single layer       ",91.95 )
print("CNN with 2 layers           ",89.58 )
print("CNN with 3 layers           ",89.71 )
print("CNN with 5 layers           ",67.59 )
```

```
                                ACC
                               ======
     LSTM with single layer     93.21
     LSTM with 2 lyers          92.43
    CNN with single layer         91.95
    CNN with 2 layers             89.58
    CNN with 3 layers             89.71
    CNN with 5 layers             67.59
```

Conclusion:

1. We split the our data 70% as train and 30% as test
2. After that we perform some EDA techniques
3. We get a feature vector of 561 features and these features are build by Domain Experts
4. Finding a best features is very expensive
5. After that use these features we build some models like Logistic Regression, Linear SVC, RBF SVM Classifier, GBDT with hyperperameter tuning
6. In these model Linear SVC gave best results with Accuracy = 96.71%
7. After that we perform simple LSTM model with single layer it gave Accuracy = 93.21 %
8. We buid simple CNN It gave Accuracy = 91.95%
9. Without any domain knowledge and any features simple LSTM gave 93.21% accuracy

```
1
```