HumanActivityRecognition

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

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

How data was recorded

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

   > 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***
pass filter with corner frequecy of 0.3Hz.
4. After that, the body linear acceleration and angular velocity were derived in time to obtian *jerk si*
tBodyGyroJerk-XYZ).
5. The magnitude of these 3-dimensional signals were calculated using the Euclidian norm. This n
with names like *tBodyAccMag*, *tGravityAccMag*, *tBodyAccJerkMag*, *tBodyGyroMag* and *tBodyGyr*
6. Finally, We've got frequency domain signals from some of the available signals by applying a FF
obtained were labeled with ***prefix 'f'*** just like original signals with ***prefix 't'***. These signals are labe
etc.,.
7. These are the signals that we got so far.
   - tBodyAcc-XYZ
   - tGravityAcc-XYZ

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

8. We can esitmate some set of variables from the above signals. ie., We will estimate the followin 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 windo
   - ***angle()***: Angle between to vectors.
9. We can obtain some other vectors by taking the average of signals in a single window sample. T
   - 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%*** subjec

Data

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

Data Size :

> 27 MB

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Quick overview of the dataset :

- Accelerometer and Gyroscope readings are taken from 30 volunteers(referred as subjects) whil
    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, wh
- 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
- 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
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

List out feature names

```
# 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 =

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

y_train = pd.read_csv('UCI_HAR_dataset/train/y_train.txt', names = ['Activity'], squeeze = Tr
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.head()
```

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc mad()-
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.98318
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.97491
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.96366
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.98275
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.97967

5 rows × 564 columns

```
print("Information of train data")
print('*'*26, '\n')
train.info()
```

```
Information of train data
**************************

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7352 entries, 0 to 7351
Columns: 564 entries, tBodyAcc-mean()-X to ActivityName
dtypes: float64(561), int64(2), object(1)
memory usage: 31.6+ MB
```

```
print("Shape of train data:", train.shape)
print("Number of rows in train data:", train.shape[0])
print("Number of columns in train data:", train.shape[1])
```

```
Shape of train data: (7352, 564)
Number of rows in train data: 7352
Number of columns in train data: 564
```

```
# 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 = Non

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

# 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.head()
```

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc mad()-
0	0.257178	-0.023285	-0.014654	-0.938404	-0.920091	-0.667683	-0.952501	-0.92524
1	0.286027	-0.013163	-0.119083	-0.975415	-0.967458	-0.944958	-0.986799	-0.96840
2	0.275485	-0.026050	-0.118152	-0.993819	-0.969926	-0.962748	-0.994403	-0.97073
3	0.270298	-0.032614	-0.117520	-0.994743	-0.973268	-0.967091	-0.995274	-0.97447
4	0.274833	-0.027848	-0.129527	-0.993852	-0.967445	-0.978295	-0.994111	-0.96595

5 rows × 564 columns

```
print("Information of test data")
```

```
print('*'*25, '\n')
test.info()
```

```
Information of test data
*************************

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2947 entries, 0 to 2946
Columns: 564 entries, tBodyAcc-mean()-X to ActivityName
dtypes: float64(561), int64(2), object(1)
memory usage: 12.7+ MB
```

```
print("Shape of test data:", test.shape)
print("Number of rows in test data:", test.shape[0])
print("Number of columns in test data:", test.shape[1])
```

```
Shape of test data: (2947, 564)
Number of rows in test data: 2947
Number of columns in test data: 564
```

Data Cleaning

- ***Check for duplicates***

```
if sum(train.duplicated()) == 0 & sum(test.duplicated()) == 0:
    print("Number of duplicates in train data:", train.duplicated().sum())
    print("Number of duplicates in test data:", test.duplicated().sum(), '\n')
    print("There are no duplicates in the dataset.")
else:
    print("Please remove the duplicates")
```

```
Number of duplicates in train data: 0
Number of duplicates in test data: 0

There are no duplicates in the dataset.
```

- ***Check for null values***

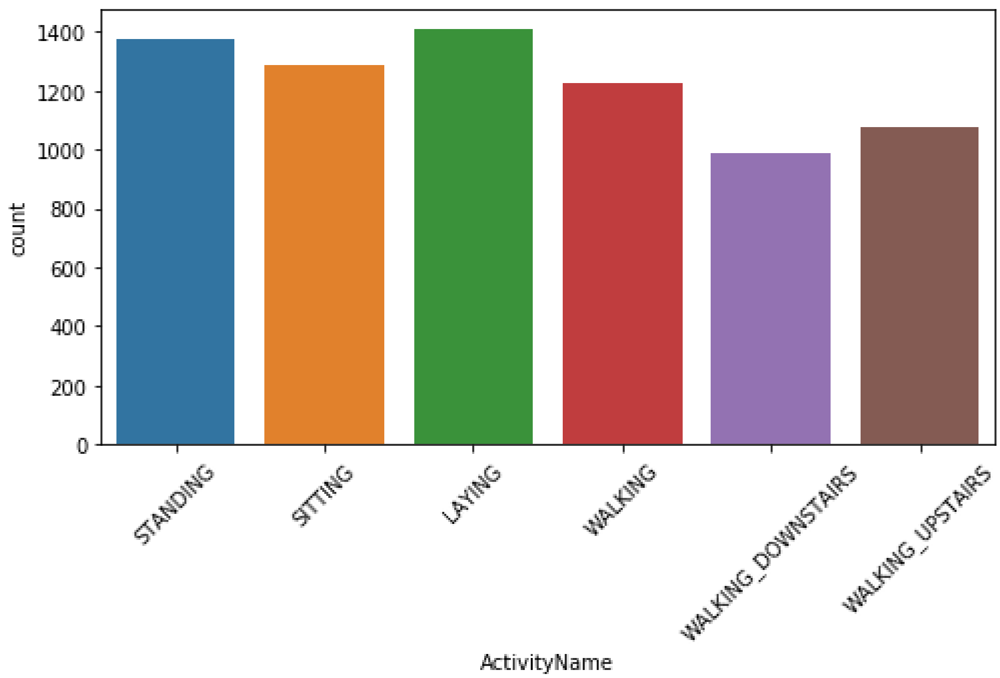
```
if train.isnull().values.sum() == 0 & test.isnull().values.sum() == 0:
    print("Null values count for train data:", train.isnull().values.sum())
    print("Null values count for test data:", test.isnull().values.sum(), '\n')
    print("From above heatmap and data, we found no null values.")
else:
    print("Please remove null values from data or fill with relevant data.")
```

```
Null values count for train data: 0
```

Graphical Visualization

```
plt.figure(figsize = (8,4))
sns.countplot(train['ActivityName'])
plt.xticks(rotation = 45)
```

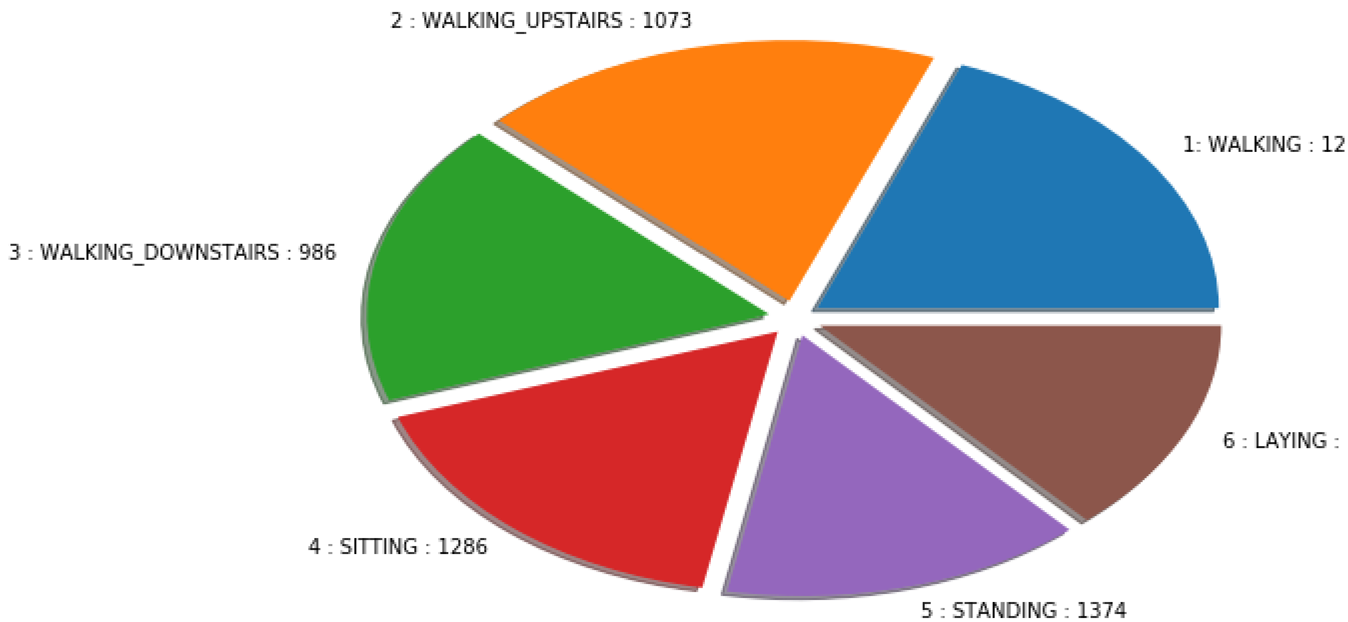
```
(array([0, 1, 2, 3, 4, 5]), <a list of 6 Text xticklabel objects>)
```



- ***Observation***
    - **Data is almost balanced.**

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

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

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

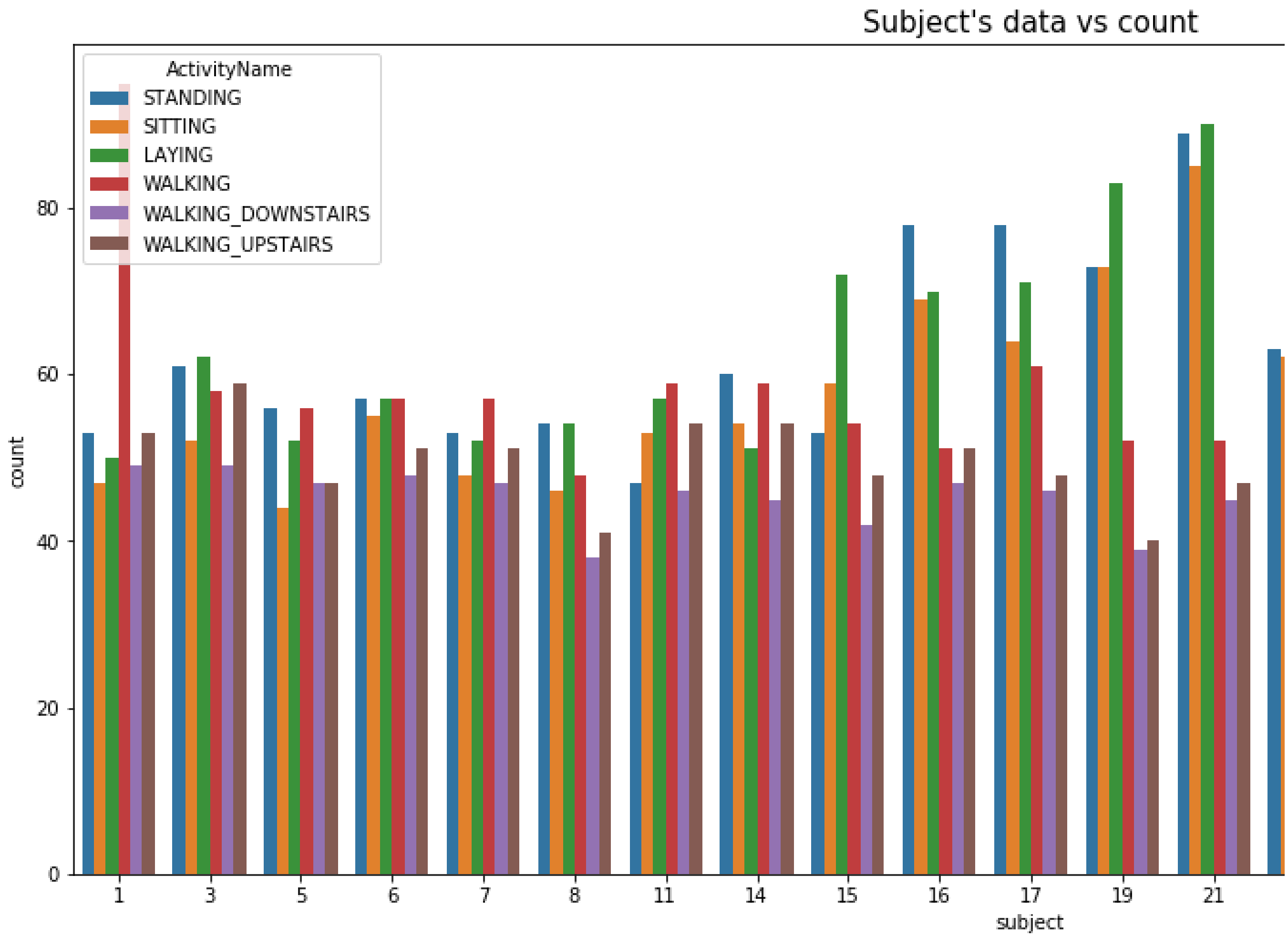
```
([<matplotlib.patches.Wedge at 0x2790e438>,
  <matplotlib.patches.Wedge at 0x2790ec18>,
  <matplotlib.patches.Wedge at 0x27916470>,
  <matplotlib.patches.Wedge at 0x27916c88>,
  <matplotlib.patches.Wedge at 0x279254e0>,
  <matplotlib.patches.Wedge at 0x27925cf8>],
 [Text(1.44312,0.989896,'1: WALKING : 1226'),
  Text(-0.379824,1.70828,'2 : WALKING_UPSTAIRS : 1073'),
  Text(-1.70957,0.373982,'3 : WALKING_DOWNSTAIRS : 986'),
  Text(-1.14436,-1.32399,'4 : SITTING : 1286'),
  Text(0.466162,-1.68677,'5 : STANDING : 1374'),
  Text(1.59696,-0.715704,'6 : LAYING : 1407')])
```



- ***Observation***
    - **Maximum count is for Laying i.e 1407**
    - **Minimum count is for Walking Downstairs i.e 986**

```
plt.figure(figsize = (18,8))
sns.countplot(x = train['subject'], hue = train['ActivityName'])
plt.title("Subject's data vs count", fontsize = 15)
```

```
Text(0.5,1,"Subject's data vs count")
```



- ***Observation***
    - **We have got almost same data from various users**

```
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

print("Train columns:\n")
print(train.columns, '\n')
print('*'*50, '\n')
print("Test columns:\n")
print(test.columns)
```

```
Train 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)

**************************************************

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)
```

- ***Saving the above data***

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

1. Featuring Engineering from Domain Knowledge

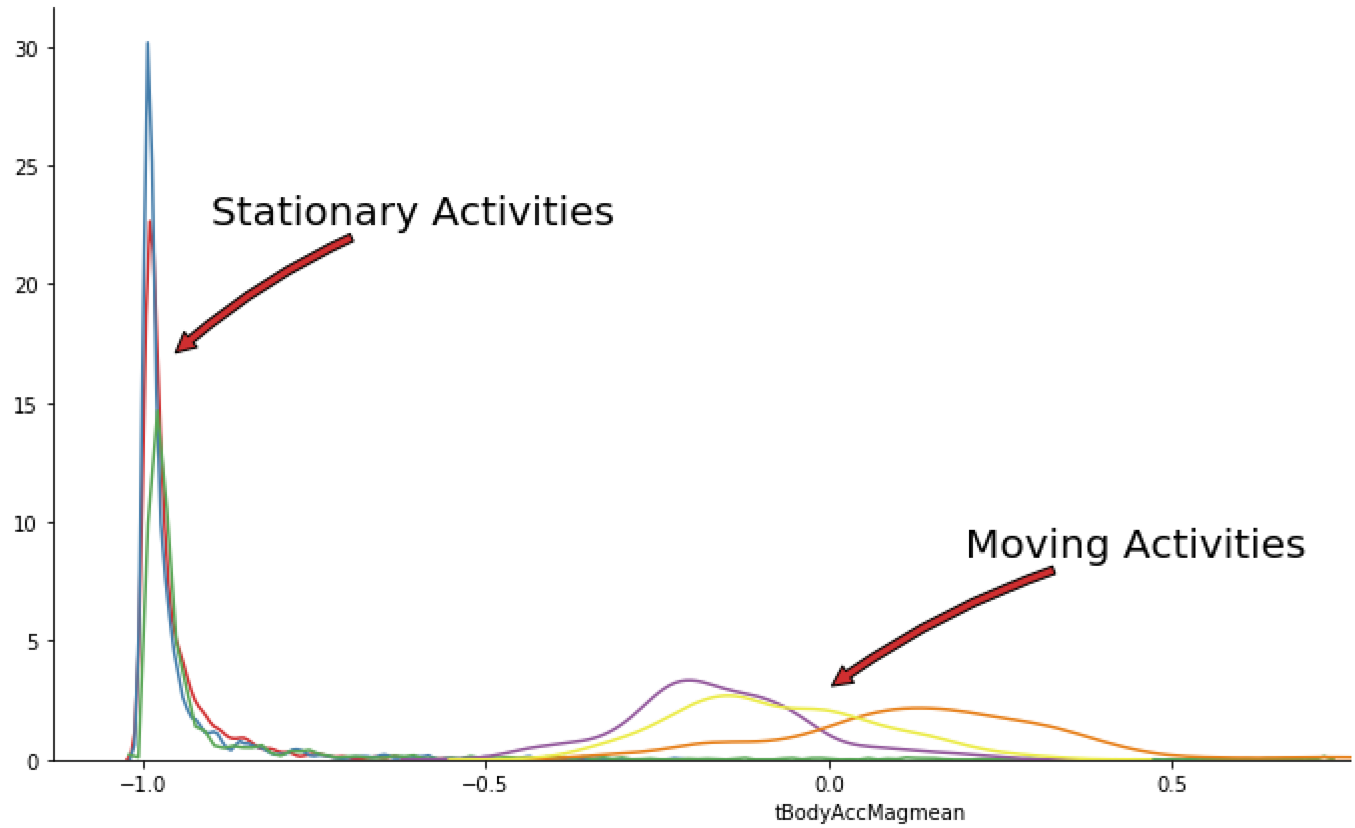
- **Static and Dynamic Activities**
  - In static activities (sit, stand, lie down) motion information will not be very useful.
  - In the dynamic activities (Walking, WalkingUpstairs,WalkingDownstairs) motion info will b

2. Stationary and Moving activities are completely different

```
sns.set_palette("Set1", desat=0.80)
g = sns.FacetGrid(train, hue = 'ActivityName', size = 6, aspect = 2)
g = g.map(sns.distplot, 'tBodyAccMagmean', hist = False)
g = g.add_legend()

plt.annotate("Stationary Activities", xy=(-0.956,17), xytext=(-0.9, 23), size=20,\
            va='center', ha='left',\
```

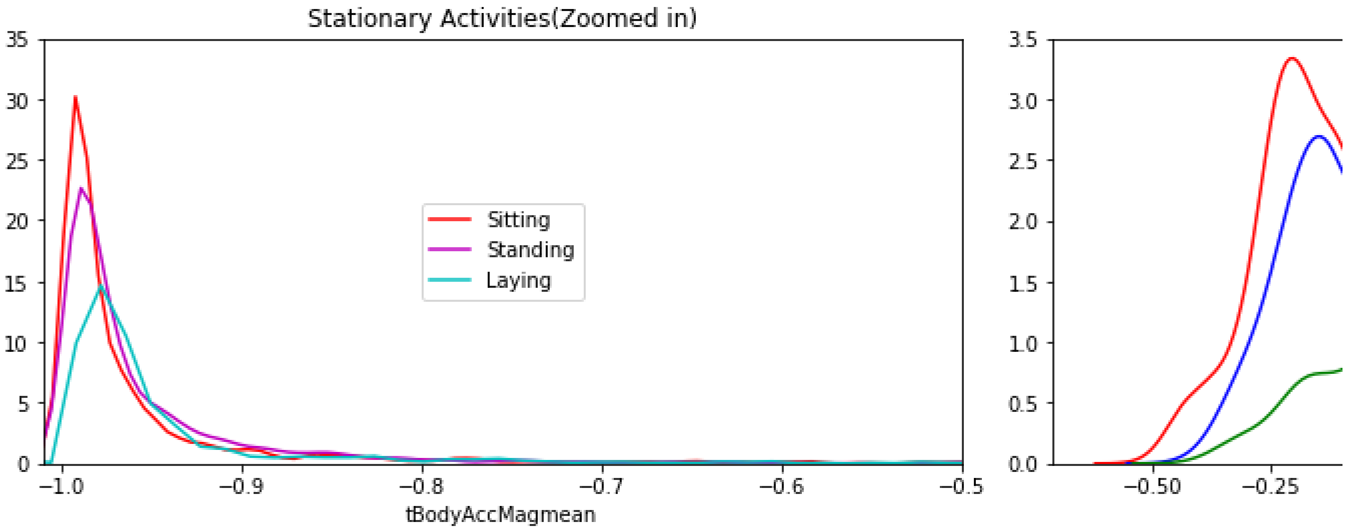
```
            arrowprops=dict(arrowstyle="simple",connectionstyle="arc3,rad=0.1"))

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"))
```

```
Text(0.2,9,'Moving Activities')
```




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

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

plt.subplot(2,2,2)
plt.title('Moving Activities')
sns.distplot(df1['tBodyAccMagmean'],color = 'red',hist = False, label = 'Walking')
sns.distplot(df2['tBodyAccMagmean'],color = 'blue',hist = False,label = 'Walking Up')
```

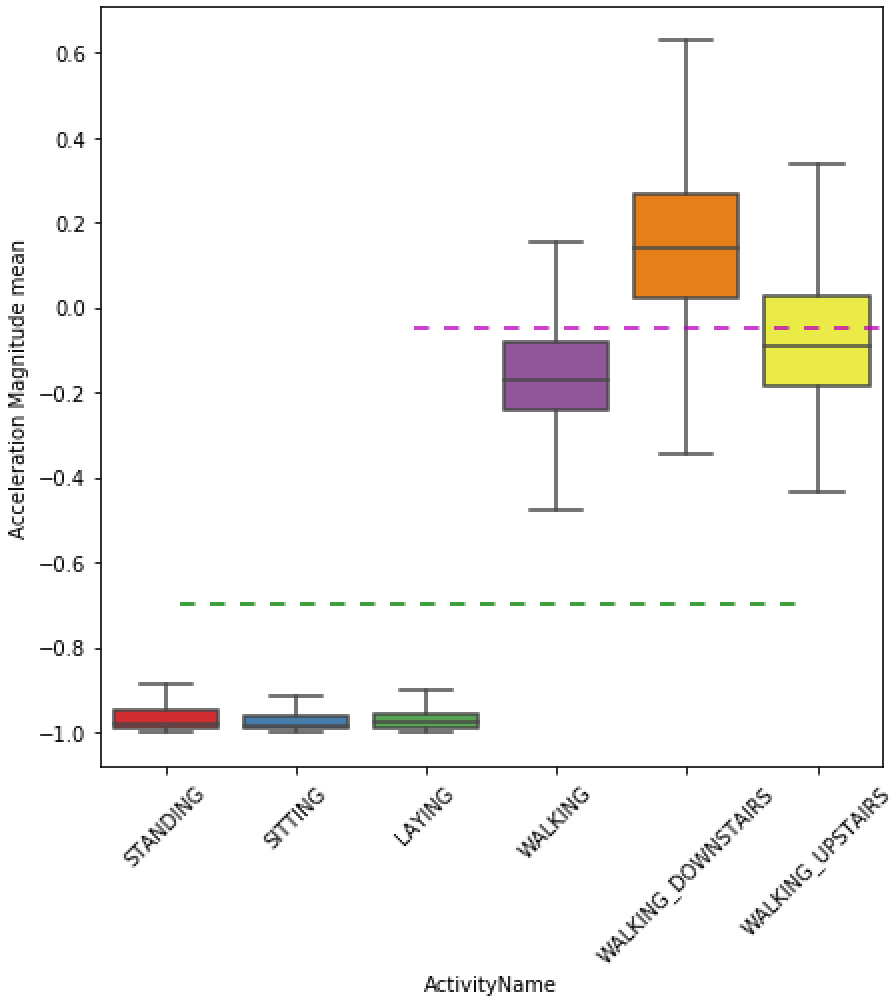
```
sns.distplot(df3['tBodyAccMagmean'],color = 'green',hist = False, label = 'Walking down')
plt.legend(loc='center right')


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



3. Magnitude of an acceleration can saperate it well

```
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=45)
```

```
(array([0, 1, 2, 3, 4, 5]), <a list of 6 Text xticklabel objects>)
```

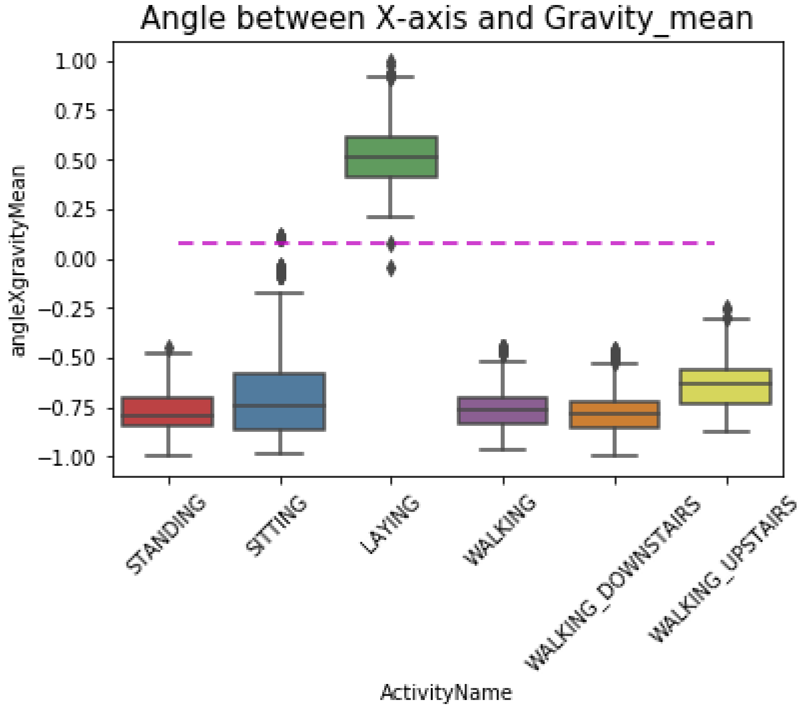


- **Observations:**
  - **If tAccMean is < -0.8 then the Activities are either Standing or Sitting or Laying.**
  - **If tAccMean is > -0.6 then the Activities are either Walking or WalkingDownstairs or Walk**
  - **If tAccMean > 0.0 then the Activity is WalkingDownstairs.**
  - **We can classify 75%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

4. Position of GravityAccelerationComponents 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 = 45)
```

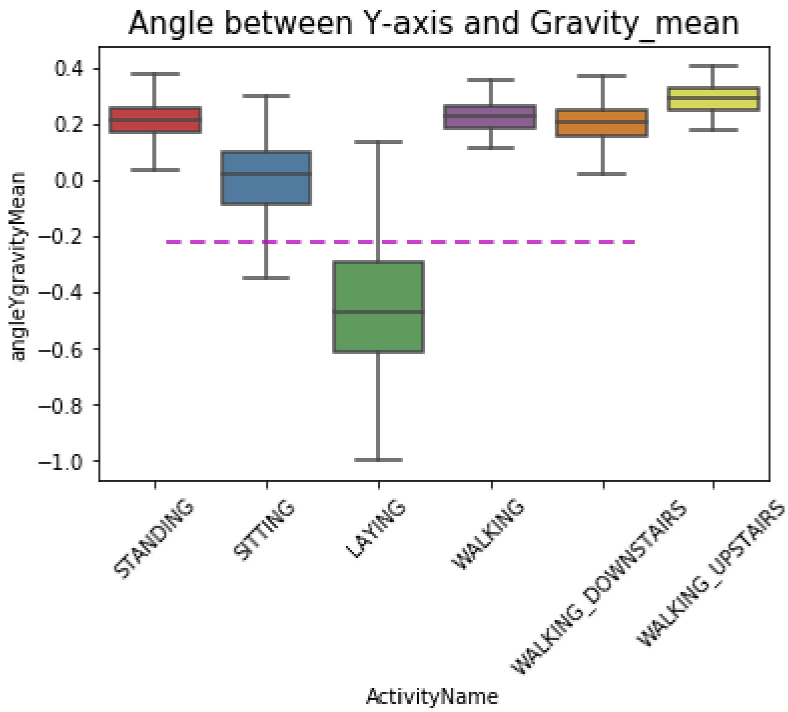
```
(array([0, 1, 2, 3, 4, 5]), <a list of 6 Text xticklabel objects>)
```



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

```
sns.boxplot(x='ActivityName', y='angleYgravityMean', data = train, showfliers=False)
plt.title('Angle between Y-axis and Gravity_mean', fontsize=15)
plt.xticks(rotation = 45)
plt.axhline(y=-0.22, xmin=0.1, xmax=0.8, dashes=(5,3), c='m')
```

```
<matplotlib.lines.Line2D at 0x2c22b748>
```



TSNE with different perplexities

Reducing 561 dimension to 2 dimension

```
# Importing library
from sklearn.manifold import TSNE
n_iter = 1000

def tsne(x_tr, y_tr, perp):

    perplexity = perp
    print("\nWith perplexity {} and number of iterations: {}\n" .format(perplexity, n_iter))

    ts = TSNE(n_components = 2, perplexity = perp, n_iter = n_iter, verbose = 2)
    ts = ts.fit_transform(x_tr)

    # Stacking to obtain dataframe which then can be used to visualize
    ts_stack = np.vstack((ts.T, y_tr)).T
    df_ts = pd.DataFrame(data = ts_stack, columns = ('x_dim', 'y_dim', 'label'))

    # Visualization
    sns.set_style('darkgrid')
    gt = sns.FacetGrid(data = df_ts, hue = 'label', size = 8)
    gt.map(plt.scatter, 'x_dim', 'y_dim').add_legend()
    plt.title('Visualizing in 2 dimension with perplexity: {} and iterations: {}' .format(per
    plt.xlabel("Dimension 1", size = 15)
    plt.ylabel("Dimension 2", size = 15)
```

```
    plt.ylabel('Dimension 2', size = 15)
```

Getting x and y data to input

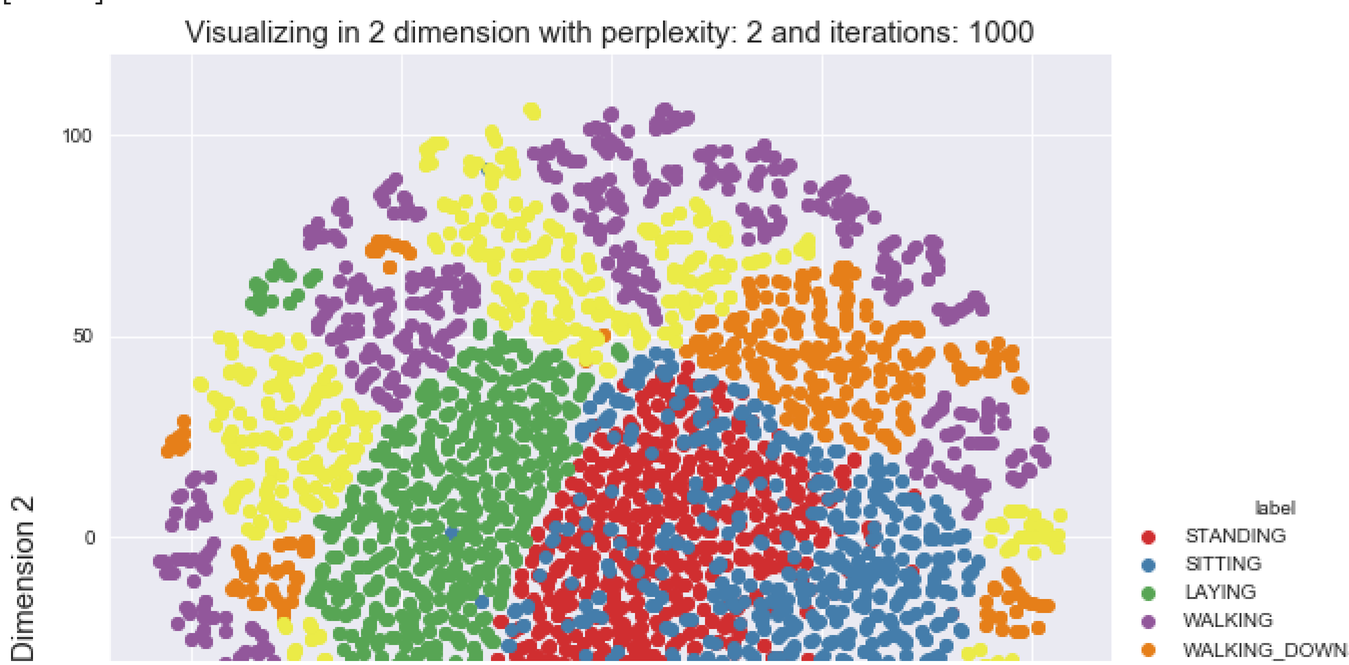
```
x_tr_ts = train.drop(['subject', 'Activity','ActivityName'], axis=1)
y_tr_ts = train['ActivityName']

print("Shape of x_train: {}" .format(x_tr_t.shape))
print("Shape of y_train: {}" .format(y_tr_t.shape))
```

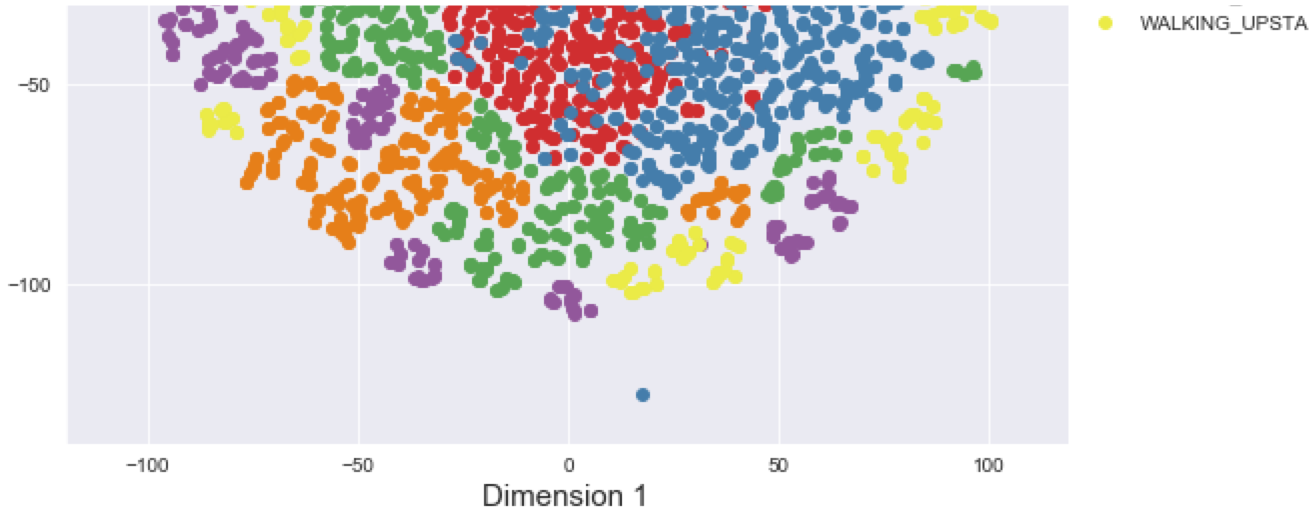
```
Shape of x_train: (7352, 561)
Shape of y_train: (7352,)
```

Perplexity 2

```
tsne(x_tr_ts, y_tr_ts, perp = 2)
```

```
With perplexity 2 and number of iterations: 1000

[t-SNE] Computing 7 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.489s...
[t-SNE] Computed neighbors for 7352 samples in 59.935s...
[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 0.109s
[t-SNE] Iteration 50: error = 124.6487350, gradient norm = 0.0251724 (50 iterations in 1
[t-SNE] Iteration 100: error = 106.8219147, gradient norm = 0.0299008 (50 iterations in
[t-SNE] Iteration 150: error = 100.7435150, gradient norm = 0.0223215 (50 iterations in
[t-SNE] Iteration 200: error = 97.4028244, gradient norm = 0.0190532 (50 iterations in 1
[t-SNE] Iteration 250: error = 95.1420975, gradient norm = 0.0194672 (50 iterations in 1
[t-SNE] KL divergence after 250 iterations with early exaggeration: 95.142097
[t-SNE] Iteration 300: error = 4.1154461, gradient norm = 0.0015610 (50 iterations in 10
[t-SNE] Iteration 350: error = 3.2074690, gradient norm = 0.0009945 (50 iterations in 9.
[t-SNE] Iteration 400: error = 2.7787852, gradient norm = 0.0007165 (50 iterations in 9.
[t-SNE] Iteration 450: error = 2.5152621, gradient norm = 0.0005695 (50 iterations in 10
[t-SNE] Iteration 500: error = 2.3322508, gradient norm = 0.0004811 (50 iterations in 10
[t-SNE] Iteration 550: error = 2.1947658, gradient norm = 0.0004128 (50 iterations in 10
[t-SNE] Iteration 600: error = 2.0852606, gradient norm = 0.0003690 (50 iterations in 10
[t-SNE] Iteration 650: error = 1.9955012, gradient norm = 0.0003340 (50 iterations in 10
[t-SNE] Iteration 700: error = 1.9199976, gradient norm = 0.0003055 (50 iterations in 10
[t-SNE] Iteration 750: error = 1.8552271, gradient norm = 0.0002750 (50 iterations in 10
[t-SNE] Iteration 800: error = 1.7986721, gradient norm = 0.0002568 (50 iterations in 10
[t-SNE] Iteration 850: error = 1.7488033, gradient norm = 0.0002368 (50 iterations in 10
[t-SNE] Iteration 900: error = 1.7041953, gradient norm = 0.0002239 (50 iterations in 10
[t-SNE] Iteration 950: error = 1.6639928, gradient norm = 0.0002107 (50 iterations in 10
[t-SNE] Iteration 1000: error = 1.6278090, gradient norm = 0.0002012 (50 iterations in 1
[t-SNE] Error after 1000 iterations: 1.627809
```





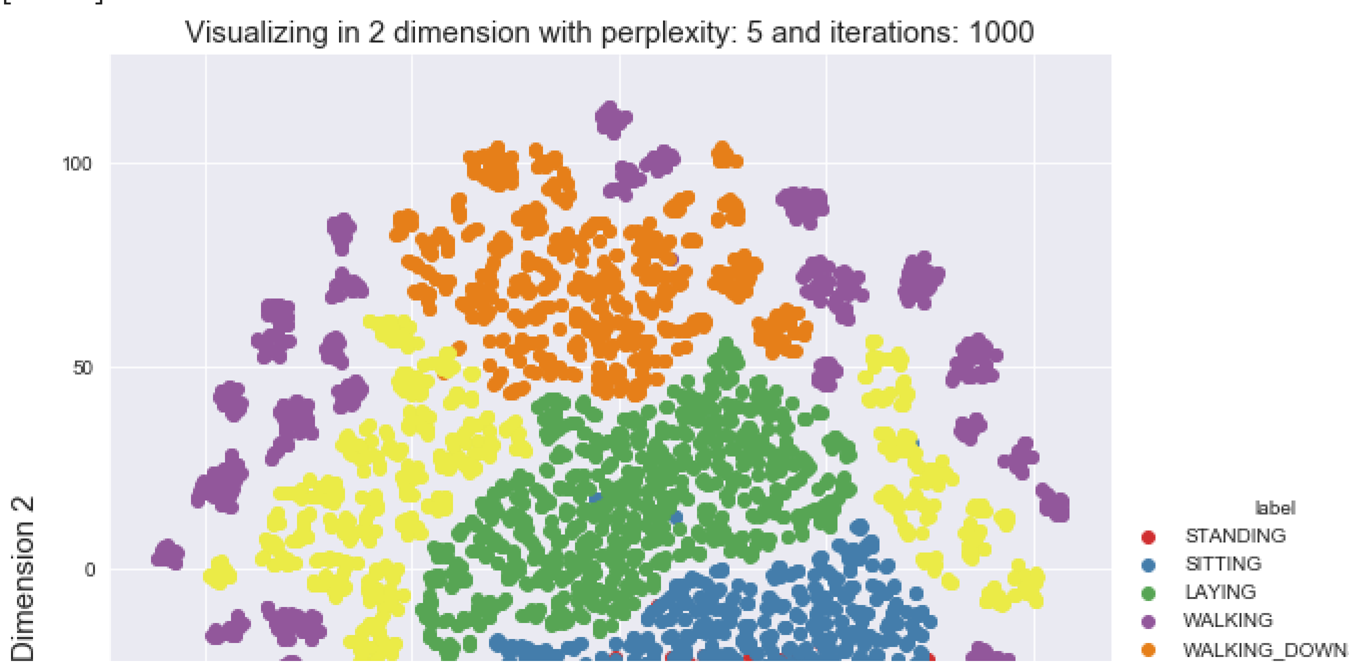
Observation

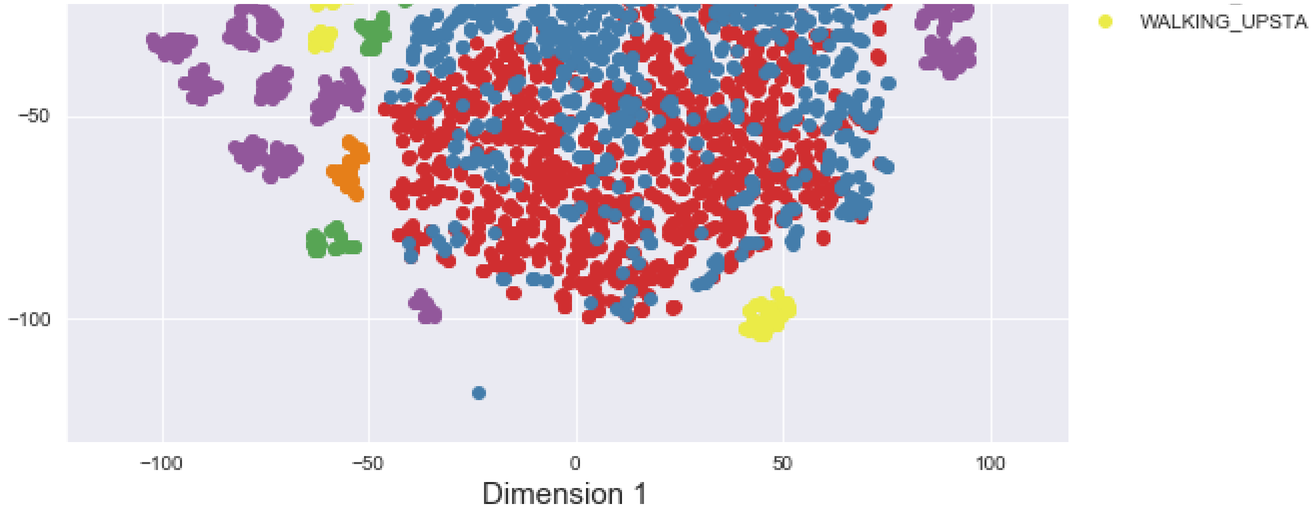
With perplexity 2 and iteration 1000, well, we can see good separation of activities but can be better w increase in perplexity, how separation is done.

Perplexity: 5

```
tsne(x_tr_ts, y_tr_ts, perp = 5)
```

```
With perplexity 5 and number of iterations: 1000

[t-SNE] Computing 16 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.480s...
[t-SNE] Computed neighbors for 7352 samples in 61.204s...
[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 0.101s
[t-SNE] Iteration 50: error = 114.0248871, gradient norm = 0.0217223 (50 iterations in 1
[t-SNE] Iteration 100: error = 97.6309814, gradient norm = 0.0145679 (50 iterations in 1
[t-SNE] Iteration 150: error = 93.3746948, gradient norm = 0.0176682 (50 iterations in 1
[t-SNE] Iteration 200: error = 91.3701782, gradient norm = 0.0075067 (50 iterations in 1
[t-SNE] Iteration 250: error = 90.1453018, gradient norm = 0.0047324 (50 iterations in 1
[t-SNE] KL divergence after 250 iterations with early exaggeration: 90.145302
[t-SNE] Iteration 300: error = 3.5762413, gradient norm = 0.0014577 (50 iterations in 10
[t-SNE] Iteration 350: error = 2.8199754, gradient norm = 0.0007487 (50 iterations in 10
[t-SNE] Iteration 400: error = 2.4395008, gradient norm = 0.0005245 (50 iterations in 10
[t-SNE] Iteration 450: error = 2.2219441, gradient norm = 0.0004040 (50 iterations in 10
[t-SNE] Iteration 500: error = 2.0768311, gradient norm = 0.0003316 (50 iterations in 10
[t-SNE] Iteration 550: error = 1.9714751, gradient norm = 0.0002849 (50 iterations in 10
[t-SNE] Iteration 600: error = 1.8902292, gradient norm = 0.0002468 (50 iterations in 10
[t-SNE] Iteration 650: error = 1.8249645, gradient norm = 0.0002195 (50 iterations in 10
[t-SNE] Iteration 700: error = 1.7711501, gradient norm = 0.0002012 (50 iterations in 10
[t-SNE] Iteration 750: error = 1.7258387, gradient norm = 0.0001793 (50 iterations in 10
[t-SNE] Iteration 800: error = 1.6870078, gradient norm = 0.0001648 (50 iterations in 11
[t-SNE] Iteration 850: error = 1.6532418, gradient norm = 0.0001530 (50 iterations in 10
[t-SNE] Iteration 900: error = 1.6234858, gradient norm = 0.0001428 (50 iterations in 10
[t-SNE] Iteration 950: error = 1.5970306, gradient norm = 0.0001352 (50 iterations in 11
[t-SNE] Iteration 1000: error = 1.5735505, gradient norm = 0.0001249 (50 iterations in 1
[t-SNE] Error after 1000 iterations: 1.573550
```





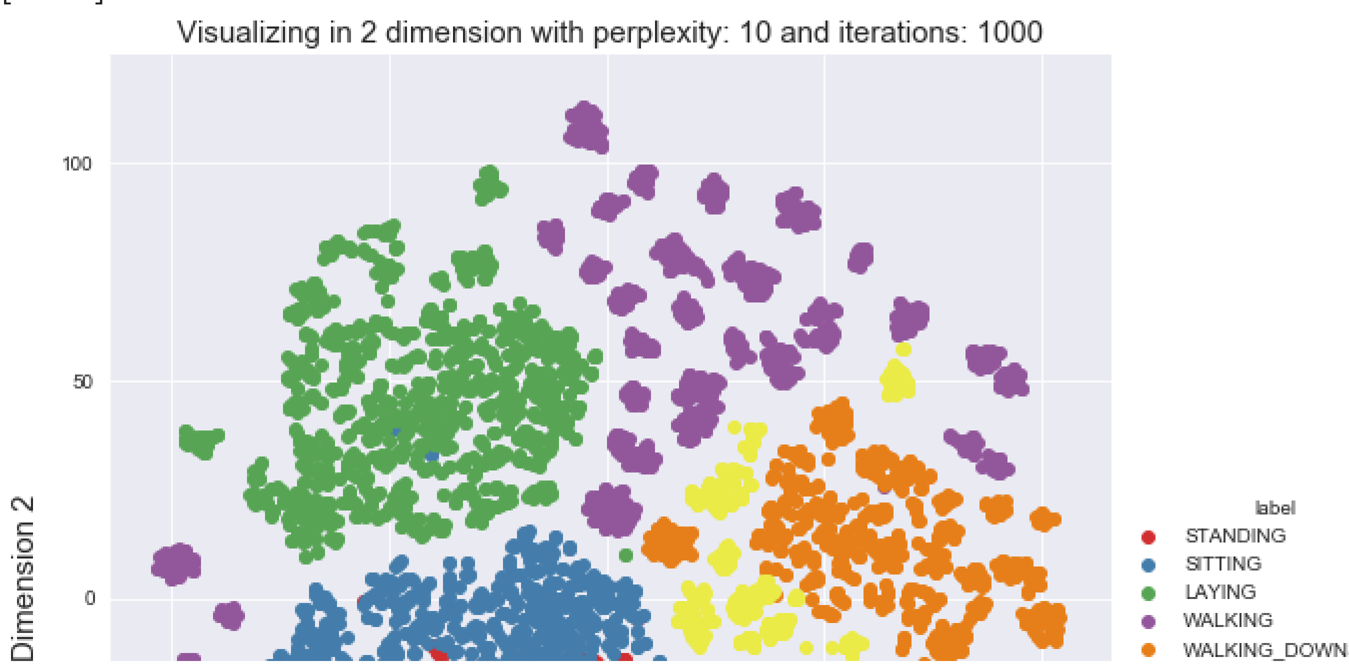
Observation

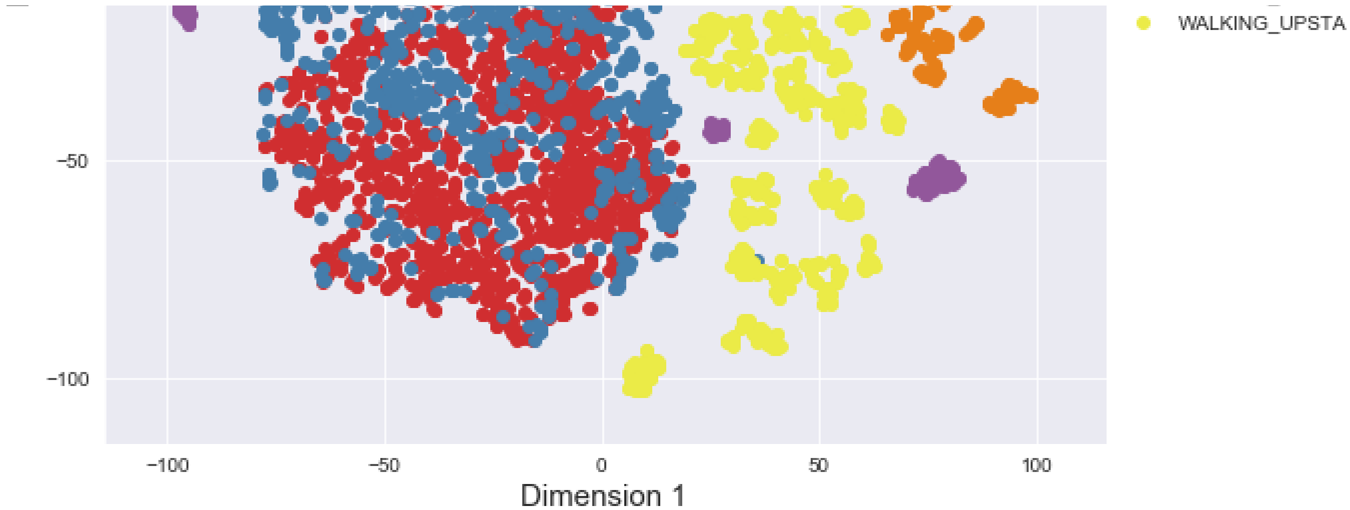
With perplexity 5 and iteration 1000, separation is improved when compared to perplexity 2.

Perplexity: 10

```
tsne(x_tr_ts, y_tr_ts, perp = 10)
```

```
With perplexity 10 and number of iterations: 1000

[t-SNE] Computing 31 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.485s...
[t-SNE] Computed neighbors for 7352 samples in 61.534s...
[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 0.184s
[t-SNE] Iteration 50: error = 105.9964523, gradient norm = 0.0148659 (50 iterations in 2
[t-SNE] Iteration 100: error = 90.6240616, gradient norm = 0.0103785 (50 iterations in 1
[t-SNE] Iteration 150: error = 87.3602829, gradient norm = 0.0052387 (50 iterations in 1
[t-SNE] Iteration 200: error = 86.0786819, gradient norm = 0.0035021 (50 iterations in 1
[t-SNE] Iteration 250: error = 85.3764648, gradient norm = 0.0030344 (50 iterations in 1
[t-SNE] KL divergence after 250 iterations with early exaggeration: 85.376465
[t-SNE] Iteration 300: error = 3.1300728, gradient norm = 0.0013928 (50 iterations in 12
[t-SNE] Iteration 350: error = 2.4876564, gradient norm = 0.0006483 (50 iterations in 11
[t-SNE] Iteration 400: error = 2.1687012, gradient norm = 0.0004231 (50 iterations in 11
[t-SNE] Iteration 450: error = 1.9844216, gradient norm = 0.0003142 (50 iterations in 12
[t-SNE] Iteration 500: error = 1.8664511, gradient norm = 0.0002502 (50 iterations in 11
[t-SNE] Iteration 550: error = 1.7831405, gradient norm = 0.0002089 (50 iterations in 11
[t-SNE] Iteration 600: error = 1.7204162, gradient norm = 0.0001813 (50 iterations in 12
[t-SNE] Iteration 650: error = 1.6711963, gradient norm = 0.0001620 (50 iterations in 14
[t-SNE] Iteration 700: error = 1.6313622, gradient norm = 0.0001448 (50 iterations in 12
[t-SNE] Iteration 750: error = 1.5986803, gradient norm = 0.0001302 (50 iterations in 12
[t-SNE] Iteration 800: error = 1.5711824, gradient norm = 0.0001177 (50 iterations in 12
[t-SNE] Iteration 850: error = 1.5476170, gradient norm = 0.0001094 (50 iterations in 12
[t-SNE] Iteration 900: error = 1.5273097, gradient norm = 0.0001016 (50 iterations in 12
[t-SNE] Iteration 950: error = 1.5094489, gradient norm = 0.0000957 (50 iterations in 12
[t-SNE] Iteration 1000: error = 1.4938735, gradient norm = 0.0000906 (50 iterations in 1
[t-SNE] Error after 1000 iterations: 1.493873
```





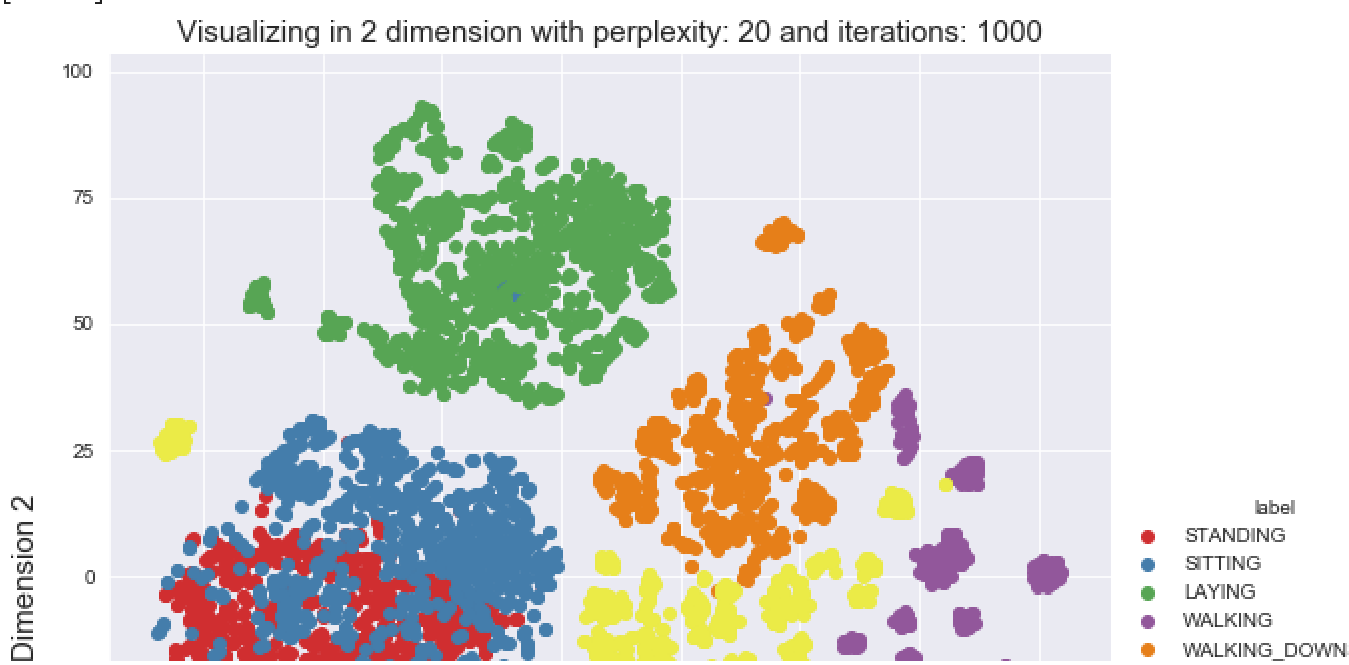
Observation

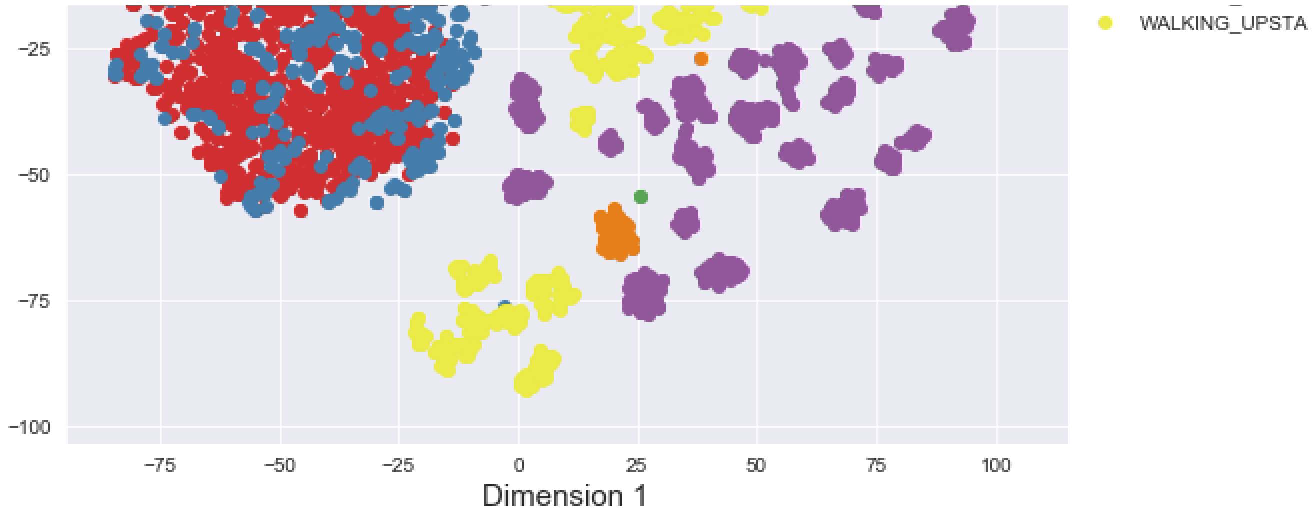
With perplexity 10 and iteration 1000, separation is improved when compared to perplexity 2 and 5.

Perplexity: 20

```
tsne(x_tr_ts, y_tr_ts, perp = 20)
```

```
With perplexity 20 and number of iterations: 1000

[t-SNE] Computing 61 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.478s...
[t-SNE] Computed neighbors for 7352 samples in 62.866s...
[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 0.362s
[t-SNE] Iteration 50: error = 97.4729080, gradient norm = 0.0184257 (50 iterations in 37
[t-SNE] Iteration 100: error = 83.8648605, gradient norm = 0.0081573 (50 iterations in 1
[t-SNE] Iteration 150: error = 81.8718338, gradient norm = 0.0032131 (50 iterations in 1
[t-SNE] Iteration 200: error = 81.1598358, gradient norm = 0.0028845 (50 iterations in 1
[t-SNE] Iteration 250: error = 80.7795334, gradient norm = 0.0030472 (50 iterations in 1
[t-SNE] KL divergence after 250 iterations with early exaggeration: 80.779533
[t-SNE] Iteration 300: error = 2.6973419, gradient norm = 0.0013124 (50 iterations in 16
[t-SNE] Iteration 350: error = 2.1634338, gradient norm = 0.0005763 (50 iterations in 14
[t-SNE] Iteration 400: error = 1.9137801, gradient norm = 0.0003476 (50 iterations in 14
[t-SNE] Iteration 450: error = 1.7669128, gradient norm = 0.0002479 (50 iterations in 14
[t-SNE] Iteration 500: error = 1.6730202, gradient norm = 0.0001936 (50 iterations in 14
[t-SNE] Iteration 550: error = 1.6087221, gradient norm = 0.0001578 (50 iterations in 14
[t-SNE] Iteration 600: error = 1.5622650, gradient norm = 0.0001349 (50 iterations in 15
[t-SNE] Iteration 650: error = 1.5273278, gradient norm = 0.0001170 (50 iterations in 15
[t-SNE] Iteration 700: error = 1.4999596, gradient norm = 0.0001052 (50 iterations in 15
[t-SNE] Iteration 750: error = 1.4783728, gradient norm = 0.0000974 (50 iterations in 15
[t-SNE] Iteration 800: error = 1.4613079, gradient norm = 0.0000868 (50 iterations in 14
[t-SNE] Iteration 850: error = 1.4470007, gradient norm = 0.0000835 (50 iterations in 15
[t-SNE] Iteration 900: error = 1.4350876, gradient norm = 0.0000813 (50 iterations in 15
[t-SNE] Iteration 950: error = 1.4255793, gradient norm = 0.0000770 (50 iterations in 14
[t-SNE] Iteration 1000: error = 1.4176060, gradient norm = 0.0000718 (50 iterations in 1
[t-SNE] Error after 1000 iterations: 1.417606
```





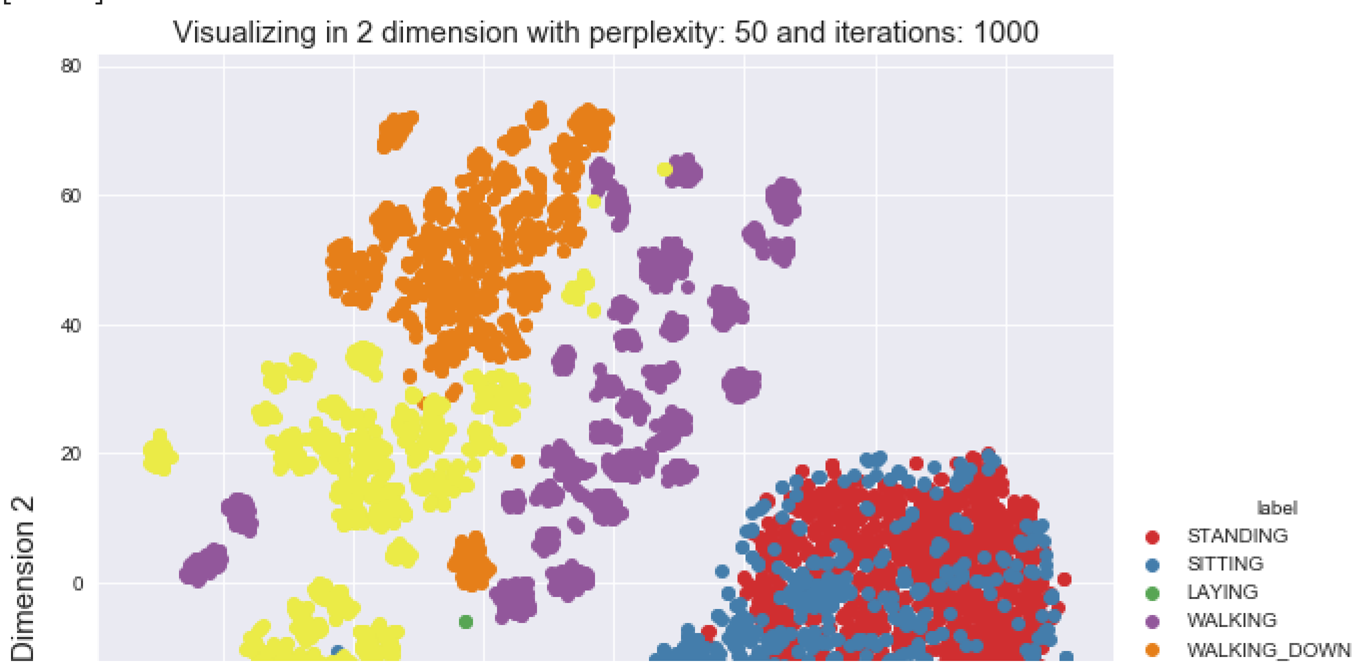
Observation

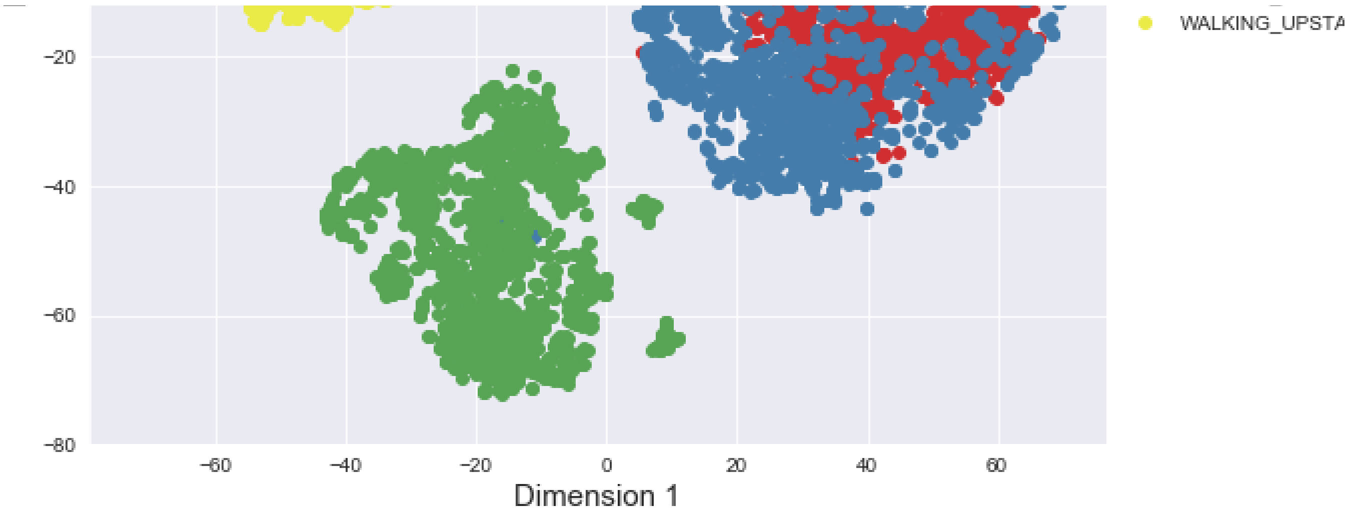
With perplexity 20 and iteration 1000, separation is improved when compared to perplexity 2, 5 and 2(

Perplexity: 50

```
tsne(x_tr_ts, y_tr_ts, perp = 50)
```

```
With perplexity 50 and number of iterations: 1000

[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.492s...
[t-SNE] Computed neighbors for 7352 samples in 65.634s...
[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 0.873s
[t-SNE] Iteration 50: error = 85.6891708, gradient norm = 0.0314668 (50 iterations in 32
[t-SNE] Iteration 100: error = 75.5255966, gradient norm = 0.0041454 (50 iterations in 2
[t-SNE] Iteration 150: error = 74.5598221, gradient norm = 0.0033632 (50 iterations in 2
[t-SNE] Iteration 200: error = 74.2188187, gradient norm = 0.0019453 (50 iterations in 2
[t-SNE] Iteration 250: error = 74.0449982, gradient norm = 0.0014738 (50 iterations in 2
[t-SNE] KL divergence after 250 iterations with early exaggeration: 74.044998
[t-SNE] Iteration 300: error = 2.1563859, gradient norm = 0.0011767 (50 iterations in 25
[t-SNE] Iteration 350: error = 1.7568213, gradient norm = 0.0004841 (50 iterations in 24
[t-SNE] Iteration 400: error = 1.5872295, gradient norm = 0.0002874 (50 iterations in 24
[t-SNE] Iteration 450: error = 1.4931519, gradient norm = 0.0001921 (50 iterations in 24
[t-SNE] Iteration 500: error = 1.4331502, gradient norm = 0.0001420 (50 iterations in 24
[t-SNE] Iteration 550: error = 1.3915766, gradient norm = 0.0001143 (50 iterations in 25
[t-SNE] Iteration 600: error = 1.3625135, gradient norm = 0.0000940 (50 iterations in 24
[t-SNE] Iteration 650: error = 1.3412588, gradient norm = 0.0000824 (50 iterations in 23
[t-SNE] Iteration 700: error = 1.3253678, gradient norm = 0.0000768 (50 iterations in 24
[t-SNE] Iteration 750: error = 1.3138707, gradient norm = 0.0000720 (50 iterations in 24
[t-SNE] Iteration 800: error = 1.3054717, gradient norm = 0.0000670 (50 iterations in 23
[t-SNE] Iteration 850: error = 1.2987312, gradient norm = 0.0000632 (50 iterations in 24
[t-SNE] Iteration 900: error = 1.2931911, gradient norm = 0.0000596 (50 iterations in 24
[t-SNE] Iteration 950: error = 1.2886649, gradient norm = 0.0000568 (50 iterations in 24
[t-SNE] Iteration 1000: error = 1.2845525, gradient norm = 0.0000534 (50 iterations in 2
[t-SNE] Error after 1000 iterations: 1.284552
```





Observation

With perplexity 20 and iteration 1000, separation is improved when compared to perplexity 2, 5, 20 an

```
# Overall observation

 1.As perplexity is changing i.e 2, 5, 10, 20 and 50, we can see that the separation of activ

 2.We can see that increase in perplexity upto ~50, separation of activities is actually impr
```

LSTM

```
# 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'])

# 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'drive/My Drive/CS_HAR/{signal}_{subset}.txt'
        #filename = f'UCI_HAR_Dataset/{subset}/Inertial Signals/{signal}_{subset}.txt'
        signals_data.append(
            _read_csv(filename).as_matrix()
        )

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



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

    return pd.get_dummies(y).as_matrix()


def load_data():
```

```
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
```

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

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

```
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
```

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

```
X_train.shape
```

```
(7352, 128, 9)
```

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

print("Timesteps:", timesteps)
print("Input dimension:", input_dim)
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of Y_train:", Y_train.shape)
print("Shape of Y_test:", Y_test.shape)
```

```
Timesteps: 128
Input dimension: 9
Shape of X_train: (7352, 128, 9)
Shape of X_test: (2947, 128, 9)
Shape of Y_train: (7352, 6)
Shape of Y_test: (2947, 6)
```

Defining 'plt_la' function

```
# Defining 'plt_la' function

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

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

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

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

```
    plt.grid()
```

Defining a function 'plotting' to visualize epoch vs loss

```
# Defining a function 'plotting' to visualize epoch vs loss

def plotting(history, t):

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

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

  # print(history.history.keys())
  # dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
  # history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verbos

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

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

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

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

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

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

  return vy, ty
```

Parameters

LSTM layers: 1 LSTM units: 64 Dropout rate: 0.25

```
n_hidden = 64

model_1 = Sequential()
```

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

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

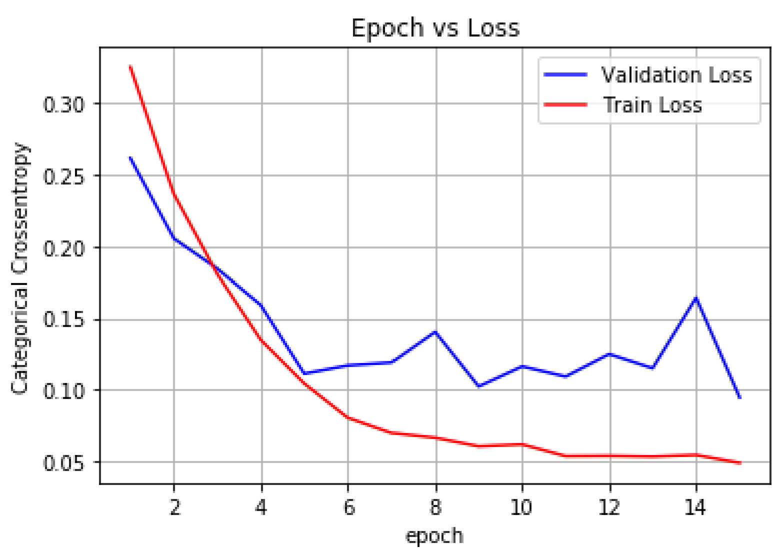
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_2 (LSTM)                (None, 64)                18944     
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 390       
=================================================================
Total params: 19,334
Trainable params: 19,334
Non-trainable params: 0
_________________________________________________________________
None
```

```
history_1 = model_1.fit(X_train, Y_train, epochs = epochs, batch_size = batch_size, validatio

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

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/15
7352/7352 [==============================] - 88s 12ms/step - loss: 0.3254 - acc: 0.8677
Epoch 2/15
7352/7352 [==============================] - 89s 12ms/step - loss: 0.2366 - acc: 0.8976
Epoch 3/15
7352/7352 [==============================] - 88s 12ms/step - loss: 0.1806 - acc: 0.9270
Epoch 4/15
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1347 - acc: 0.9525
Epoch 5/15
7352/7352 [==============================] - 87s 12ms/step - loss: 0.1042 - acc: 0.9647
```

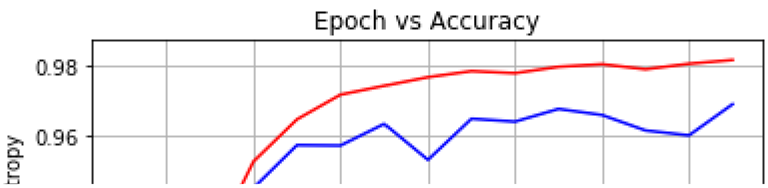
Calling 'plotting' function to visualize epoch vs loss

```
v_l_1, t_l_1 = plotting(history_1, 'loss')
```



Calling 'plotting' function to visualize epoch vs accuracy

```
v_a_1, t_a_1 = plotting(history_1, 'acc')
```



Observation

```
tr_a_1 = np.round(max(t_a_1),3)
va_a_1 = np.round(max(v_a_1),3)

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

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

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

```
Train accuracy: 0.982
Validation accuracy: 0.969

Train loss: 0.049
Validation loss: 0.094
```

Confusion Matrix

```
# Confusion Matrix
print(confusion_matrix(Y_test, model_1.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 530        7         0        0                   0
SITTING                  0      417        71        1                   0
STANDING                 0      126       405        1                   0
WALKING                  0        4         2      454                  23
WALKING_DOWNSTAIRS       0        0         0        0                 406
WALKING_UPSTAIRS         0        4         2        4                   3

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            2
STANDING                           0
WALKING                           13
WALKING_DOWNSTAIRS                14
WALKING_UPSTAIRS                 458
```

ObservationS

1.Laying: 530 correctly predicted and 0 wrongly predicted. 2.Sitting: 417 correctly predicted and 141 v correctly predicted and 75 wrongly predicted. 4.Walking: 454 correctly predicted and 6 wrongly predic predicted and 26 wrongly predicted. 6.Walking_Upstairs: 458 correctly predicted and 29 wrongly pred sitting shows more error when compared to other activities.

Parameters

1.LSTM layers: 2 2.LSTM units: 100 3.Dropout rate: 0.5

```
n_hidden_2 = 100

model_2 = Sequential()

# 2 LSTM layer
model_2.add(LSTM(n_hidden_2, input_shape = (timesteps, input_dim), return_sequences = True))
model_2.add(Dropout(0.50))
model_2.add(LSTM(n_hidden_2))                          # 2 LSTM

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

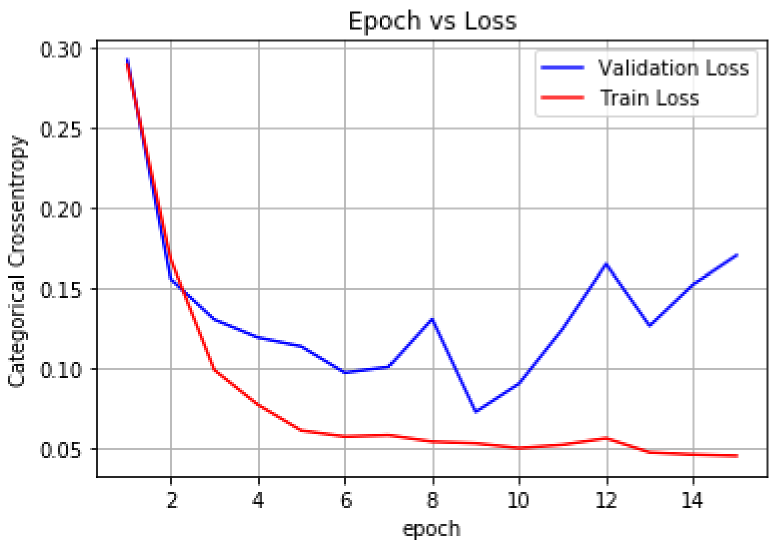
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 128, 100)          44000
_________________________________________________________________
dropout_3 (Dropout)          (None, 128, 100)          0
_________________________________________________________________
lstm_4 (LSTM)                (None, 100)               80400
_________________________________________________________________
dropout_4 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 606
=================================================================
Total params: 125,006
Trainable params: 125,006
Non-trainable params: 0
_________________________________________________________________
None
```

```
history_2 = model_2.fit(X_train, Y_train, epochs = epochs, batch_size = batch_size , validati

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

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/15
7352/7352 [==============================] - 175s 24ms/step - loss: 0.2893 - acc: 0.8780
Epoch 2/15
7352/7352 [==============================] - 175s 24ms/step - loss: 0.1678 - acc: 0.9347
Epoch 3/15
7352/7352 [==============================] - 176s 24ms/step - loss: 0.0990 - acc: 0.9662
Epoch 4/15
7352/7352 [==============================] - 177s 24ms/step - loss: 0.0773 - acc: 0.9728
Epoch 5/15
7352/7352 [==============================] - 174s 24ms/step - loss: 0.0611 - acc: 0.9777
Epoch 6/15
7352/7352 [==============================] - 174s 24ms/step - loss: 0.0573 - acc: 0.9802
Epoch 7/15
7352/7352 [==============================] - 175s 24ms/step - loss: 0.0582 - acc: 0.9778
Epoch 8/15
7352/7352 [==============================] - 172s 23ms/step - loss: 0.0541 - acc: 0.9805
Epoch 9/15
7352/7352 [==============================] - 175s 24ms/step - loss: 0.0531 - acc: 0.9800
Epoch 10/15
7352/7352 [==============================] - 173s 24ms/step - loss: 0.0501 - acc: 0.9810
Epoch 11/15
7352/7352 [==============================] - 175s 24ms/step - loss: 0.0522 - acc: 0.9811
Epoch 12/15
7352/7352 [==============================] - 173s 23ms/step - loss: 0.0563 - acc: 0.9813
Epoch 13/15
7352/7352 [==============================] - 173s 24ms/step - loss: 0.0474 - acc: 0.9826
Epoch 14/15
7352/7352 [==============================] - 173s 24ms/step - loss: 0.0460 - acc: 0.9818
Epoch 15/15
7352/7352 [==============================] - 172s 23ms/step - loss: 0.0453 - acc: 0.9833
2947/2947 [==============================] - 12s 4ms/step
```

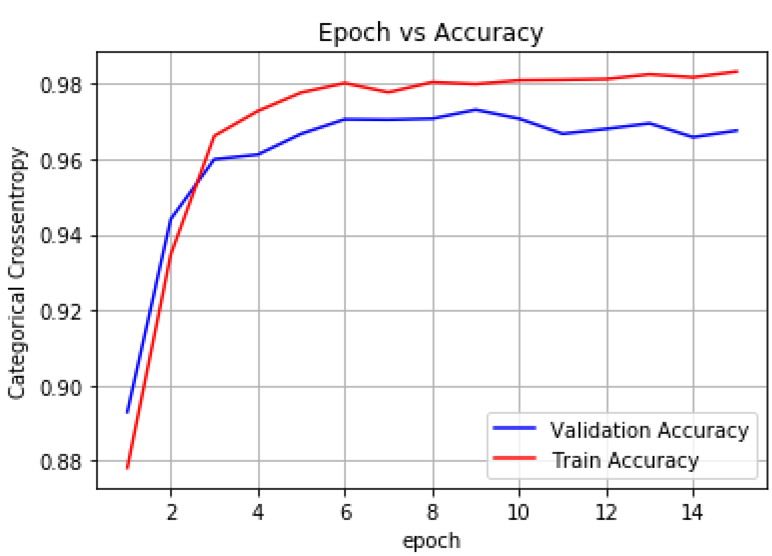
Calling 'plotting' function to visualize epoch vs loss

```
v_l_2, t_l_2 = plotting(history_2, 'loss')
```



Calling 'plotting' function to visualize epoch vs accuracy

```
v_a_2, t_a_2 = plotting(history_2, 'acc')
```



Observation

```
tr_a_2 = np.round(max(t_a_2),3)
va_a_2 = np.round(max(v_a_2),3)

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

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

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

```
Train accuracy: 0.983
Validation accuracy: 0.973

Train loss: 0.045
Validation loss: 0.893
```

Confusion Matrix

```
# Confusion Matrix
print(confusion_matrix(Y_test, model_2.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 515        0         0        0                   0
SITTING                  4      374        94        0                   0
STANDING                 0       53       478        1                   0
WALKING                  0        0         0      474                  22
WALKING_DOWNSTAIRS       0        0         0       34                 383
WALKING_UPSTAIRS         0        0         0       34                  11

Pred                WALKING_UPSTAIRS
True
LAYING                            22
SITTING                           19
STANDING                           0
WALKING                            0
WALKING_DOWNSTAIRS                 3
WALKING_UPSTAIRS                 426
```

Observations

1.Laying: 515 correctly predicted and 4 wrongly predicted. 2.Sitting: 374 correctly predicted and 53 wı
predicted and 94 wrongly predicted. 4.Walking_Downstairs: 383 correctly predicted and 33 wrongly pı
predicted and 69 wrongly predicted. 6.Walking_Upstairs: 426 correctly predicted and 44 wrongly pred

Parameters

1.LSTM layers: 3 2.LSTM units: 150 3.Dropout rate: 0.75

```
n_hidden_3 = 150

model_3 = Sequential()

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

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

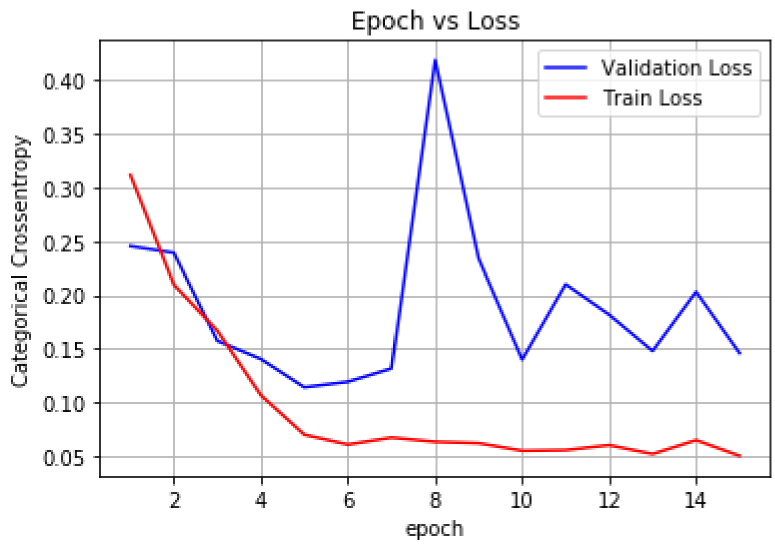
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_5 (LSTM)                (None, 128, 150)          96000
_________________________________________________________________
dropout_5 (Dropout)          (None, 128, 150)          0
_________________________________________________________________
lstm_6 (LSTM)                (None, 128, 150)          180600
_________________________________________________________________
dropout_6 (Dropout)          (None, 128, 150)          0
_________________________________________________________________
lstm_7 (LSTM)                (None, 150)               180600
_________________________________________________________________
dropout_7 (Dropout)          (None, 150)               0
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 906
=================================================================
Total params: 458,106
Trainable params: 458,106
Non-trainable params: 0
_________________________________________________________________
None
```

```
history_3 = model_3.fit(X_train, Y_train, epochs = epochs, batch_size = batch_size, validatio

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

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/15
7352/7352 [==============================] - 274s 37ms/step - loss: 0.3115 - acc: 0.8689
Epoch 2/15
7352/7352 [==============================] - 270s 37ms/step - loss: 0.2096 - acc: 0.9048
Epoch 3/15
7352/7352 [==============================] - 267s 36ms/step - loss: 0.1670 - acc: 0.9239
Epoch 4/15
7352/7352 [==============================] - 265s 36ms/step - loss: 0.1069 - acc: 0.9632
Epoch 5/15
7352/7352 [==============================] - 266s 36ms/step - loss: 0.0702 - acc: 0.9755
Epoch 6/15
7352/7352 [==============================] - 265s 36ms/step - loss: 0.0611 - acc: 0.9792
Epoch 7/15
7352/7352 [==============================] - 265s 36ms/step - loss: 0.0675 - acc: 0.9786
Epoch 8/15
7352/7352 [==============================] - 266s 36ms/step - loss: 0.0636 - acc: 0.9788
Epoch 9/15
7352/7352 [==============================] - 265s 36ms/step - loss: 0.0623 - acc: 0.9802
Epoch 10/15
7352/7352 [==============================] - 264s 36ms/step - loss: 0.0553 - acc: 0.9816
Epoch 11/15
7352/7352 [==============================] - 265s 36ms/step - loss: 0.0559 - acc: 0.9822
Epoch 12/15
7352/7352 [==============================] - 263s 36ms/step - loss: 0.0605 - acc: 0.9811
Epoch 13/15
7352/7352 [==============================] - 264s 36ms/step - loss: 0.0523 - acc: 0.9827
Epoch 14/15
7352/7352 [==============================] - 263s 36ms/step - loss: 0.0652 - acc: 0.9807
Epoch 15/15
7352/7352 [==============================] - 265s 36ms/step - loss: 0.0505 - acc: 0.9830
2947/2947 [==============================] - 17s 6ms/step
```

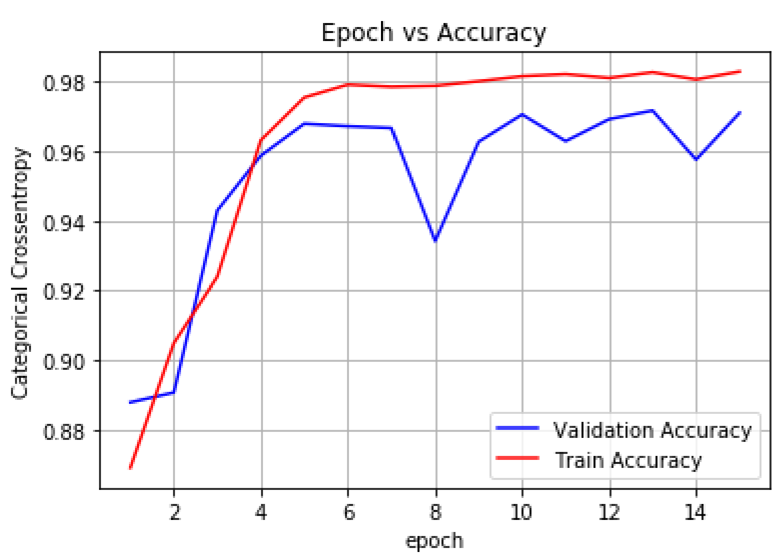
Calling 'plotting' function to visualize epoch vs loss

```
v_l_3, t_l_3 = plotting(history_3, 'loss')
```



Calling 'plotting' function to visualize epoch vs accuracy

```
v_a_3, t_a_3 = plotting(history_3, 'acc')
```



Observation

```
tr_a_3 = np.round(max(t_a_3),3)
va_a_3 = np.round(max(v_a_3),3)

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

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

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

```
Train accuracy: 0.983
Validation accuracy: 0.972

Train loss: 0.051
Validation loss: 0.888
```

Confusion Matrix

```
# Confusion Matrix
print(confusion_matrix(Y_test, model_3.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 537        0         0        0                   0
SITTING                  2      425        57        0                   0
STANDING                 0      115       414        0                   0
WALKING                  0        0         0      443                  11
WALKING_DOWNSTAIRS       0        0         0        5                 410
WALKING_UPSTAIRS         0        0         0        2                   8

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            7
STANDING                           3
WALKING                           42
WALKING_DOWNSTAIRS                 5
WALKING_UPSTAIRS                 461
```

ObservationS

1.Laying: 537 correctly predicted and 2 wrongly predicted. 2.Sitting: 425 correctly predicted and 115 v correctly predicted and 57 wrongly predicted. 4.Walking: 443 correctly predicted and 7 wrongly predic predicted and 19 wrongly predicted. 6.Walking_Upstairs: 461 correctly predicted and 57 wrongly pred

Pretty Table

```
from prettytable import PrettyTable

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

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

S.No	LSTM Units	LSTM Layers	Drop Out	Test Loss	Test Accuracy
1	64	1	0.25	0.094	0.969
2	100	2	0.5	0.893	0.973
3	150	3	0.75	0.888	0.972

CONCLUSIONS:

1.Import dataset 2.Apply Exploratory Data Analysis 3.Clean the data 4.feature engineering 5.visualize Lstm,while applying prepare data and obtain confusion matrix then do Lstm Hyperparameter tuning. / layers and dropout rate is actually resulting in increase in accuracy and loss. Accuracy:As LSTM layer 0.25, accuracy is increased by 0.2%.