HumanActivityRecognition

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

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

How data was recorded

By using the sensors(Gyroscope and accelerometer) in a smartphone, they have captured '3-axial linear acceleration'(*tAcc-XYZ*) from accelerometer and '3-axial angular velocity' (*tGyro-XYZ*) from Gyroscope with several variations.

> prefix 't' in those metrics denotes time.
>
> suffix 'XYZ' represents 3-axial signals in X , Y, and Z directions.

Feature names

1. These sensor signals are preprocessed by applying noise filters and then sampled in fixed-width windows(sliding windows) of 2.56 seconds each with 50% overlap. ie., each window has 128 readings.
2. From Each window, a feature vector was obtianed by calculating variables from the time and frequency domain.

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

3. The accelertion signal was saperated into Body and Gravity acceleration signals(***tBodyAcc-XYZ*** and ***tGravityAcc-XYZ***) using some low pass filter with corner frequecy of 0.3Hz.
4. After that, the body linear acceleration and angular velocity were derived in time to obtian *jerk signals* (***tBodyAccJerk-XYZ*** and ***tBodyGyroJerk-XYZ***).
5. The magnitude of these 3-dimensional signals were calculated using the Euclidian norm. This magnitudes are represented as features with names like *tBodyAccMag*, *tGravityAccMag*, *tBodyAccJerkMag*, *tBodyGyroMag* and *tBodyGyroJerkMag*.
6. Finally, We've got frequency domain signals from some of the available signals by applying a FFT (Fast Fourier Transform). These signals obtained were labeled with ***prefix 'f'*** just like original signals with ***prefix 't'***. These signals are labeled as ***fBodyAcc-XYZ***, ***fBodyGyroMag*** etc.,.
7. These are the signals that we got so far.

   - tBodyAcc-XYZ
   - tGravityAcc-XYZ
   - tBodyAccJerk-XYZ
   - tBodyGyro-XYZ
   - tBodyGyroJerk-XYZ
   - tBodyAccMag
   - tGravityAccMag
   - tBodyAccJerkMag
   - tBodyGyroMag
   - tBodyGyroJerkMag
   - fBodyAcc-XYZ
   - fBodyAccJerk-XYZ
   - fBodyGyro-XYZ
   - fBodyAccMag
   - fBodyAccJerkMag
   - fBodyGyroMag
   - fBodyGyroJerkMag
8. We can esitmate some set of variables from the above signals. ie., We will estimate the following properties on each and every signal that we recoreded so far.

   - ***mean()***: Mean value
   - ***std()***: Standard deviation
   - ***mad()***: Median absolute deviation
   - ***max()***: Largest value in array

- ***max()***: Largest value in array
- ***min()***: Smallest value in array
- ***sma()***: Signal magnitude area
- ***energy()***: Energy measure. Sum of the squares divided by the number of values.
- ***iqr()***: Interquartile range
- ***entropy()***: Signal entropy
- ***arCoeff()***: Autorregresion coefficients with Burg order equal to 4
- ***correlation()***: correlation coefficient between two signals
- ***maxInds()***: index of the frequency component with largest magnitude
- ***meanFreq()***: Weighted average of the frequency components to obtain a mean frequency
- ***skewness()***: skewness of the frequency domain signal
- ***kurtosis()***: kurtosis of the frequency domain signal
- ***bandsEnergy()***: Energy of a frequency interval within the 64 bins of the FFT of each window.
- ***angle()***: Angle between to vectors.

9. We can obtain some other vectors by taking the average of signals in a single window sample. These are used on the angle() variable' `

    - gravityMean
    - tBodyAccMean
    - tBodyAccJerkMean
    - tBodyGyroMean
    - tBodyGyroJerkMean

Y_Labels(Encoded)

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

Train and test data were saperated

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

Data

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

Data Size :

> 27 MB

Quick overview of the dataset :

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

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

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

Problem Framework

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

Problem Statement

- Given a new datapoint we have to predict the Activity

In [1]:

```
import numpy as np
import pandas as pd

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

```
No of Features: 561
```

Obtain the train data

In [2]:

```
# get the data from txt files to pandas dataffame
X_train = pd.read_csv(r'C:\Users\sagar\HumanActivityRecognition
(1)\HAR\UCI_HAR_Dataset\train\X_train.txt', delim_whitespace=True, header=None, names=features)

# add subject column to the dataframe
X_train['subject'] = pd.read_csv(r'C:\Users\sagar\HumanActivityRecognition
(1)\HAR\UCI_HAR_Dataset\train\subject_train.txt', header=None, squeeze=True)

y_train = pd.read_csv(r'C:\Users\sagar\HumanActivityRecognition
(1)\HAR\UCI_HAR_Dataset\train\y_train.txt', names=['Activity'], squeeze=True)
y_train_labels = y_train.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS',\
                       4:'SITTING', 5:'STANDING',6:'LAYING'})

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

```
C:\Users\sagar\Anaconda3\lib\site-packages\pandas\io\parsers.py:678: UserWarning: Duplicate names
specified. This will raise an error in the future.
```

```
  return _read(filepath_or_buffer, kwds)
```

Out[2]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X
1897	0.331529	0.053118	-0.18797	-0.088091	0.464054	0.205314	-0.145625	0.49083	0.240378	0.097249

1 rows × 564 columns

In [3]:

```
train.shape
```

Out[3]:

```
(7352, 564)
```

Obtain the test data

In [4]:

```
# get the data from txt files to pandas dataffame
X_test = pd.read_csv(r'C:\Users\sagar\HumanActivityRecognition
(1)\HAR\UCI_HAR_Dataset\test\X_test.txt', delim_whitespace=True, header=None, names=features)

# add subject column to the dataframe
X_test['subject'] = pd.read_csv(r'C:\Users\sagar\HumanActivityRecognition
(1)\HAR\UCI_HAR_Dataset\test\subject_test.txt', header=None, squeeze=True)

# get y labels from the txt file
y_test = pd.read_csv(r'C:\Users\sagar\HumanActivityRecognition
(1)\HAR\UCI_HAR_Dataset\test\y_test.txt', names=['Activity'], squeeze=True)
y_test_labels = y_test.map({1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS',\
                       4:'SITTING', 5:'STANDING',6:'LAYING'})


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

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

Out[4]:

	tBodyAcc-mean()-X	tBodyAcc-mean()-Y	tBodyAcc-mean()-Z	tBodyAcc-std()-X	tBodyAcc-std()-Y	tBodyAcc-std()-Z	tBodyAcc-mad()-X	tBodyAcc-mad()-Y	tBodyAcc-mad()-Z	tBodyAcc-max()-X
1081	0.276379	-0.01521	-0.108411	-0.998237	-0.990655	-0.995439	-0.998549	-0.989867	-0.995978	-0.943655

1 rows × 564 columns

In [5]:

```
test.shape
```

Out[5]:

```
(2947, 564)
```

Data Cleaning

1. Check for Duplicates

In [6]:

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

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

2. Checking for NaN/null values

In [7]:

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

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

observations:

1.There is no null values and duplicates

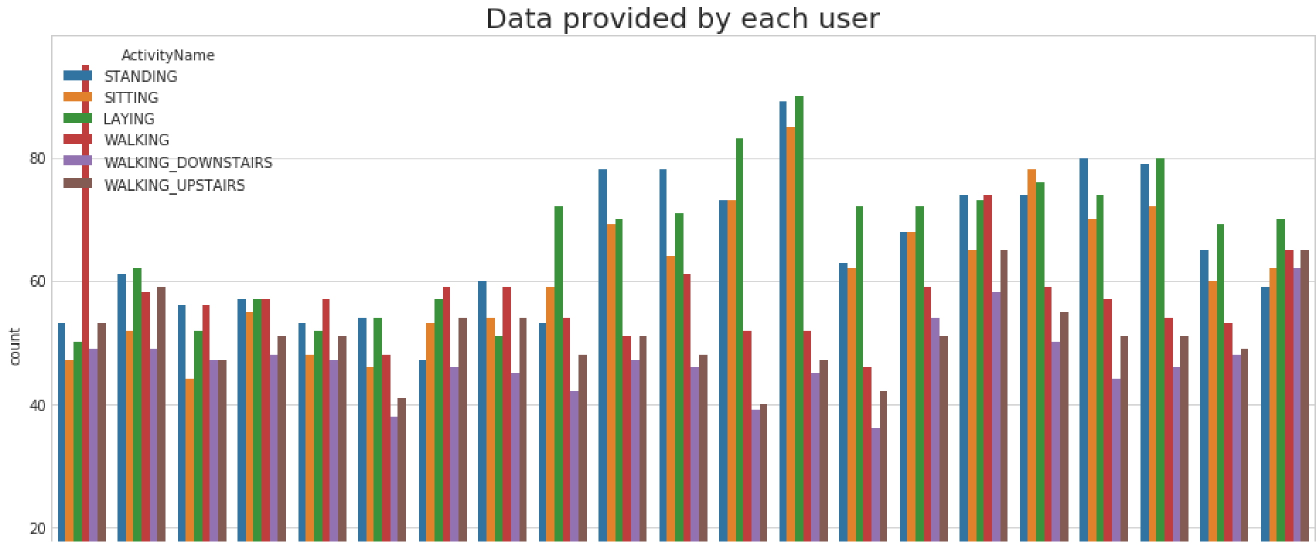
3. Check for data imbalance

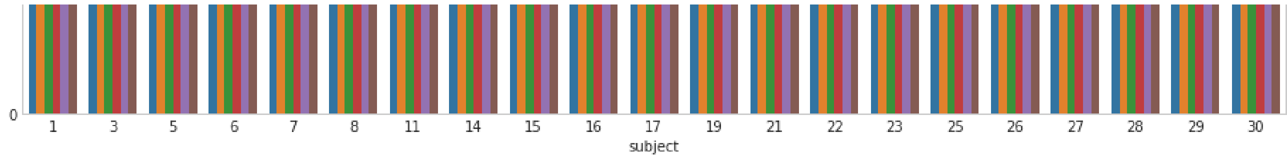
In [10]:

```
import matplotlib.pyplot as plt
import seaborn as sns

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

In [11]:

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

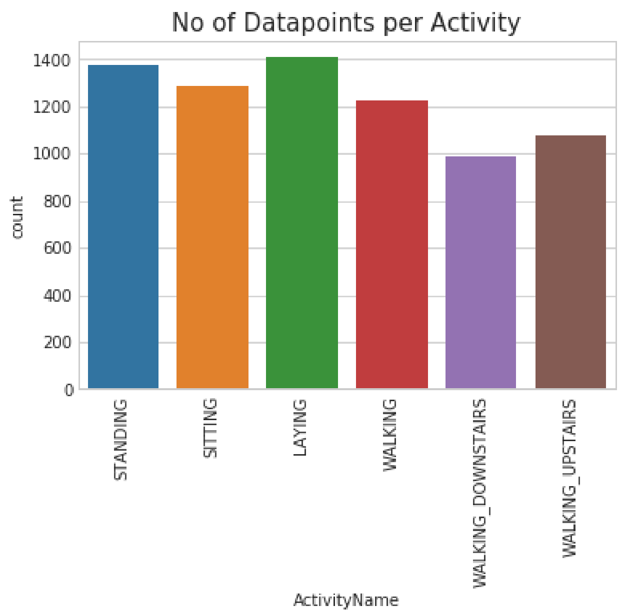




> We have got almost same number of reading from all the subjects

In [12]:

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



Observation

> Our data is well balanced (almost)

4. Changing feature names

In [13]:

```
columns = train.columns

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

train.columns = columns
test.columns = columns

test.columns
```

Out[13]:

```
Index(['tBodyAccmeanX', 'tBodyAccmeanY', 'tBodyAccmeanZ', 'tBodyAccstdX',
       'tBodyAccstdY', 'tBodyAccstdZ', 'tBodyAccmadX', 'tBodyAccmadY',
       'tBodyAccmadZ', 'tBodyAccmaxX',
       ...
       'angletBodyAccMeangravity', 'angletBodyAccJerkMeangravityMean',
       'angletBodyGyroMeangravityMean', 'angletBodyGyroJerkMeangravityMean',
```

```
       'angleXgravityMean', 'angleYgravityMean', 'angleZgravityMean',
       'subject', 'Activity', 'ActivityName'],
      dtype='object', length=564)
```

5. Save this dataframe in a csv files

In [14]:

```
train.to_csv(r'C:\Users\sagar\HumanActivityRecognition (1)\HAR\UCI_HAR_Dataset\csv_files\train.csv', index=False)
test.to_csv(r'C:\Users\sagar\HumanActivityRecognition (1)\HAR\UCI_HAR_Dataset\csv_files\test.csv', index=False)
```

Exploratory Data Analysis

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

1. Featuring Engineering from Domain Knowledge

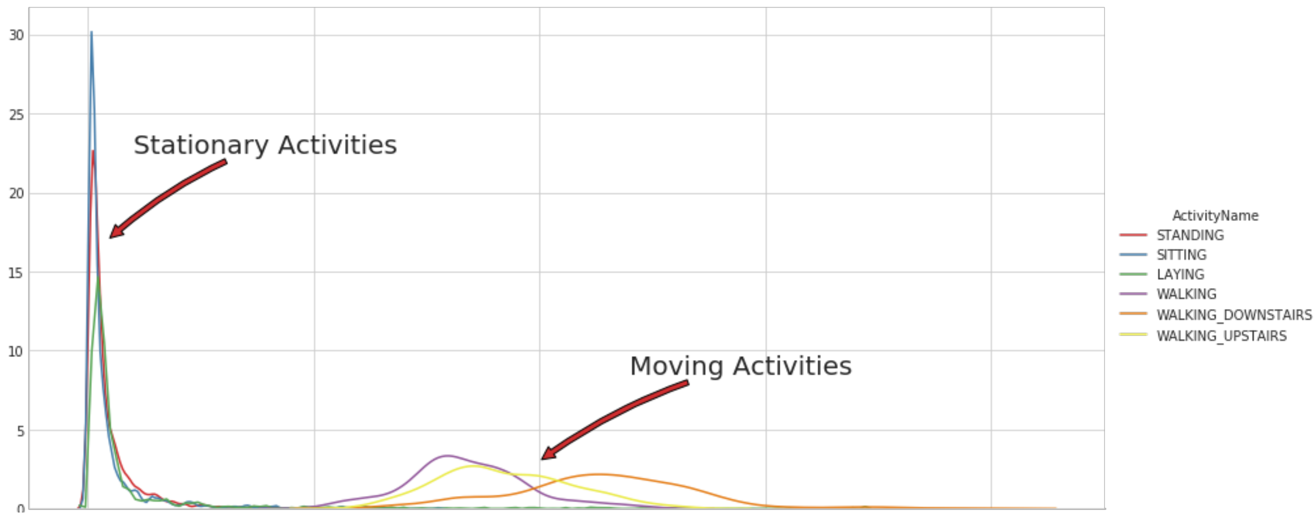
- **Static and Dynamic Activities**
  - In static activities (sit, stand, lie down) motion information will not be very useful.
  - In the dynamic activities (Walking, WalkingUpstairs,WalkingDownstairs) motion info will be significant.

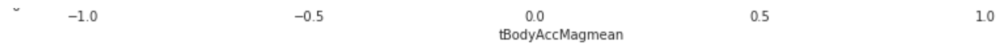
2. Stationary and Moving activities are completely different

In [15]:

```
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\sagar\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```

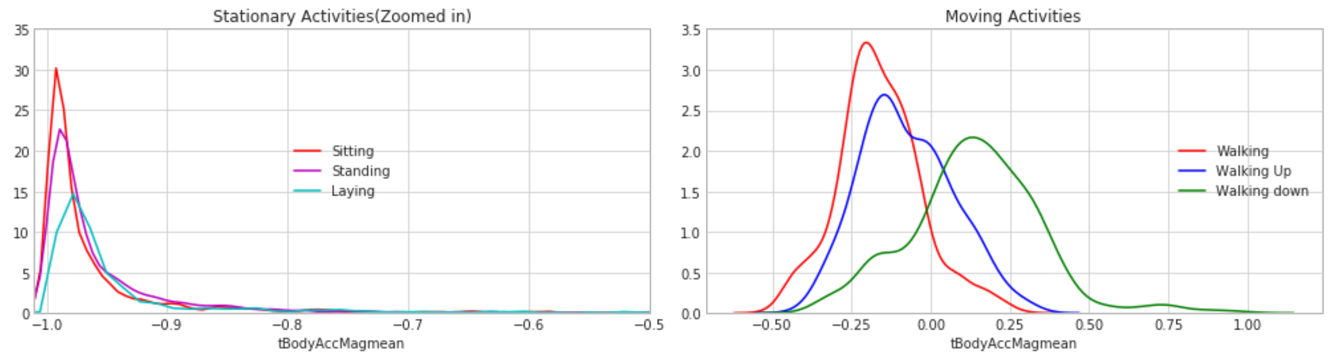




In [16]:

```
# 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()
```

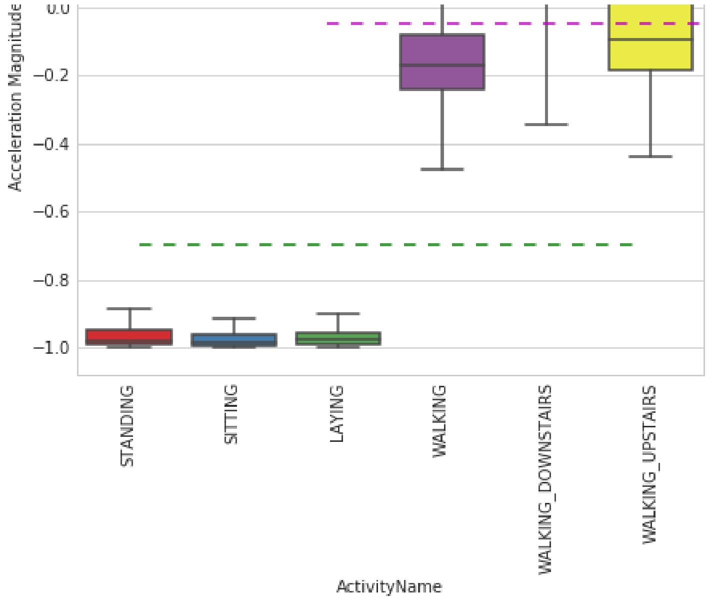
```
C:\Users\sagar\Anaconda3\lib\site-packages\scipy\stats\stats.py:1713: FutureWarning: Using a non-t
uple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[s
eq]`. In the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will r
esult either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



3. Magnitude of an acceleration can saperate it well

In [17]:

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



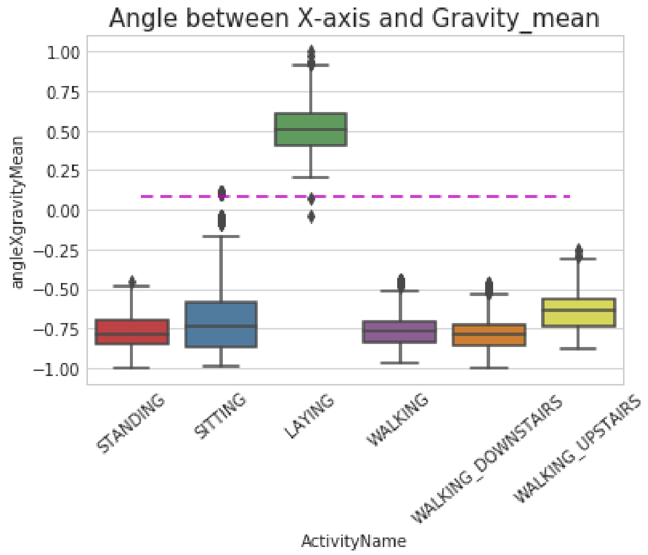
Observations:

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

4. Position of GravityAccelerationComponants also matters

In [18]:

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



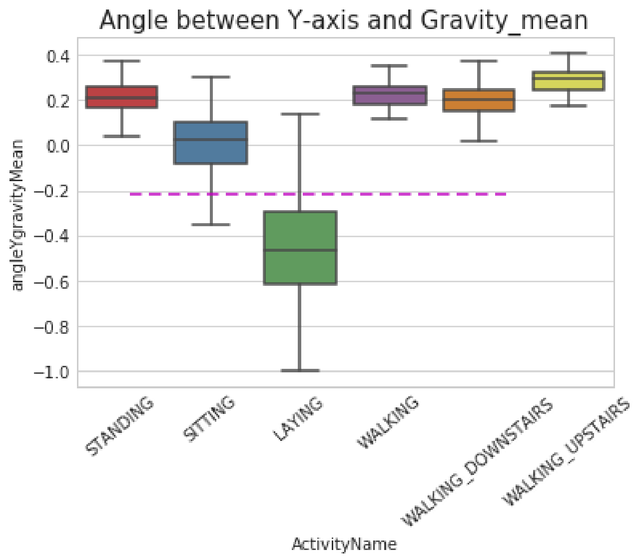
Observations:

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

In [19]:

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

```
plt.show()
```



Apply t-sne on the data

In [20]:

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

In [21]:

```
# 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(perplexit
y, n_iter))
        X_reduced = TSNE(verbose=2, perplexity=perplexity).fit_transform(X_data)
        print('Done..')

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

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

In [48]:

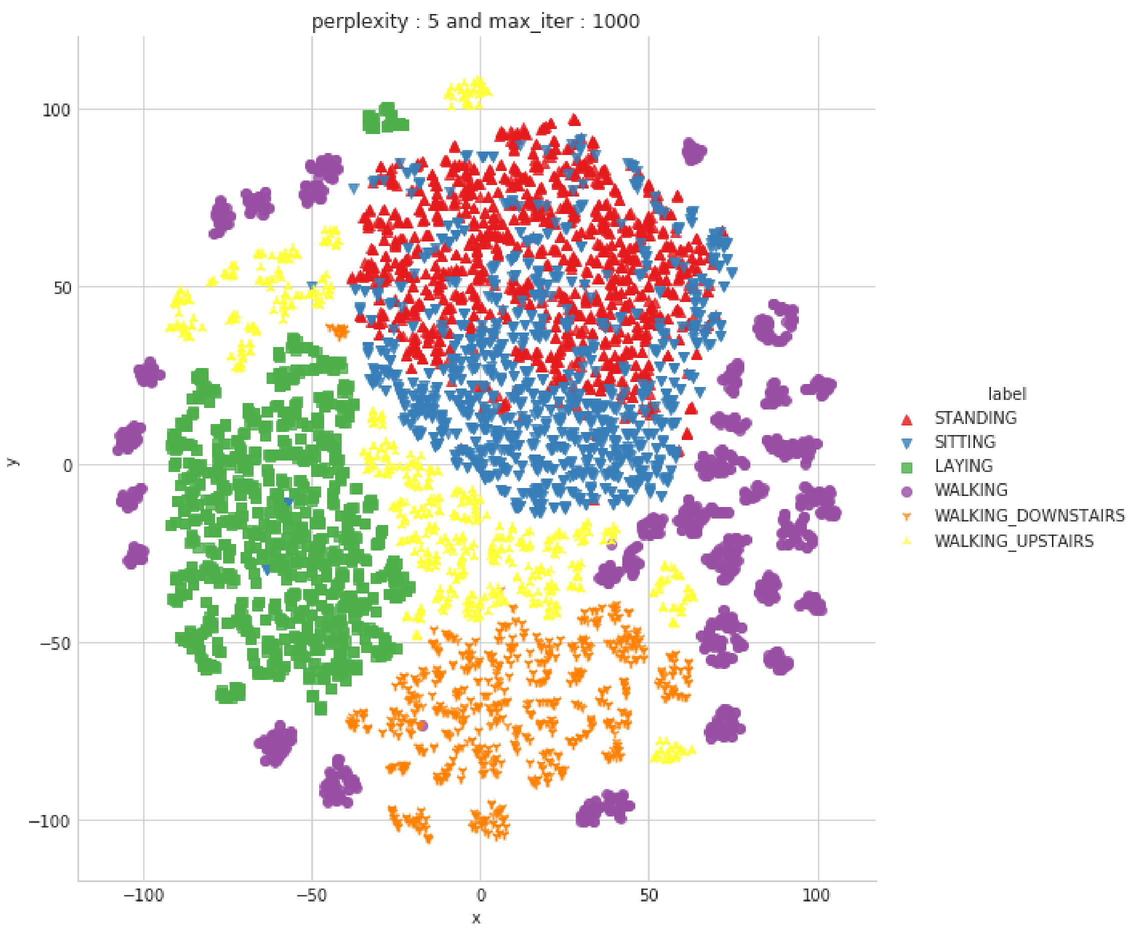
```
X_pre_tsne = train.drop(['subject', 'Activity','ActivityName'], axis=1)
y_pre_tsne = train['ActivityName']
perform_tsne(X_data = X_pre_tsne,y_data=y_pre_tsne, perplexities =[2,5,10,20,50])
```

```
performing tsne with perplexity 2 and with 1000 iterations at max
[t-SNE] Computing 7 nearest neighbors...
```

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

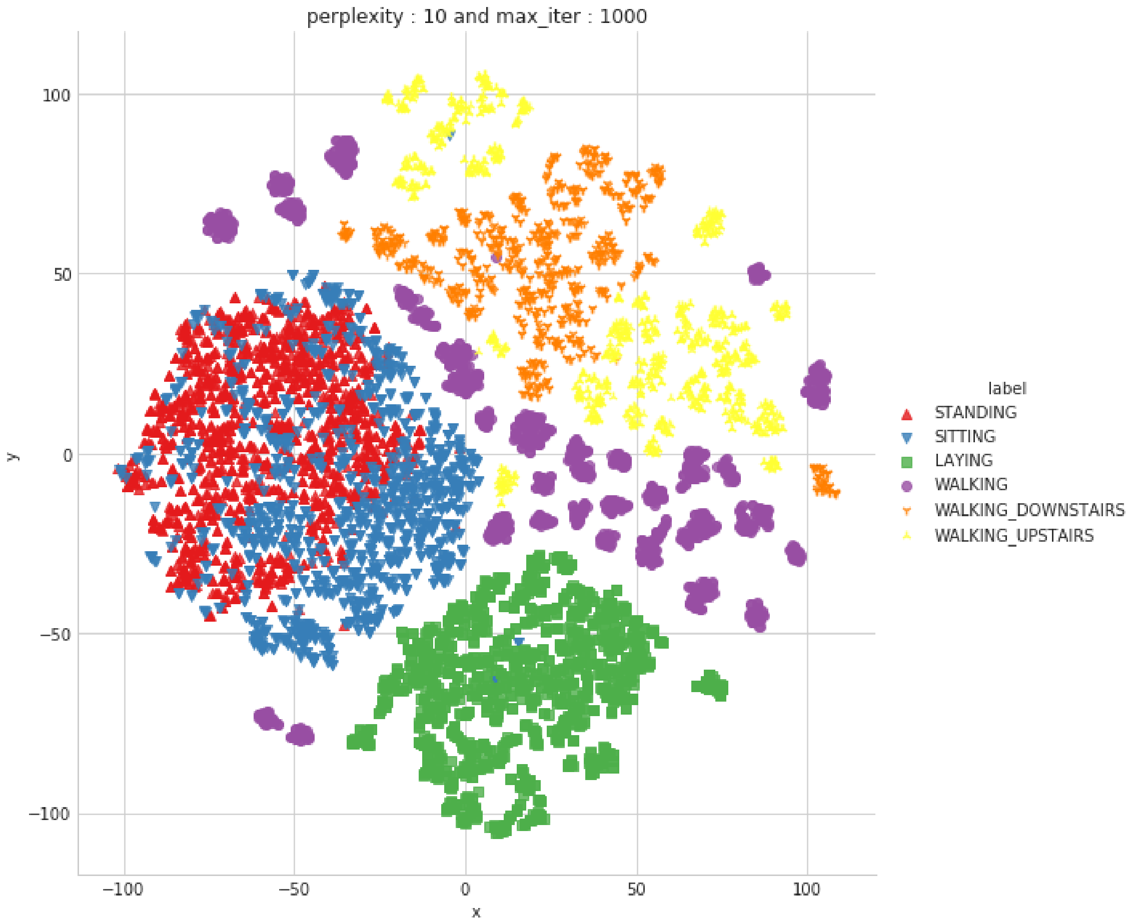

```
Done

performing tsne with perplexity 5 and with 1000 iterations at max
```

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



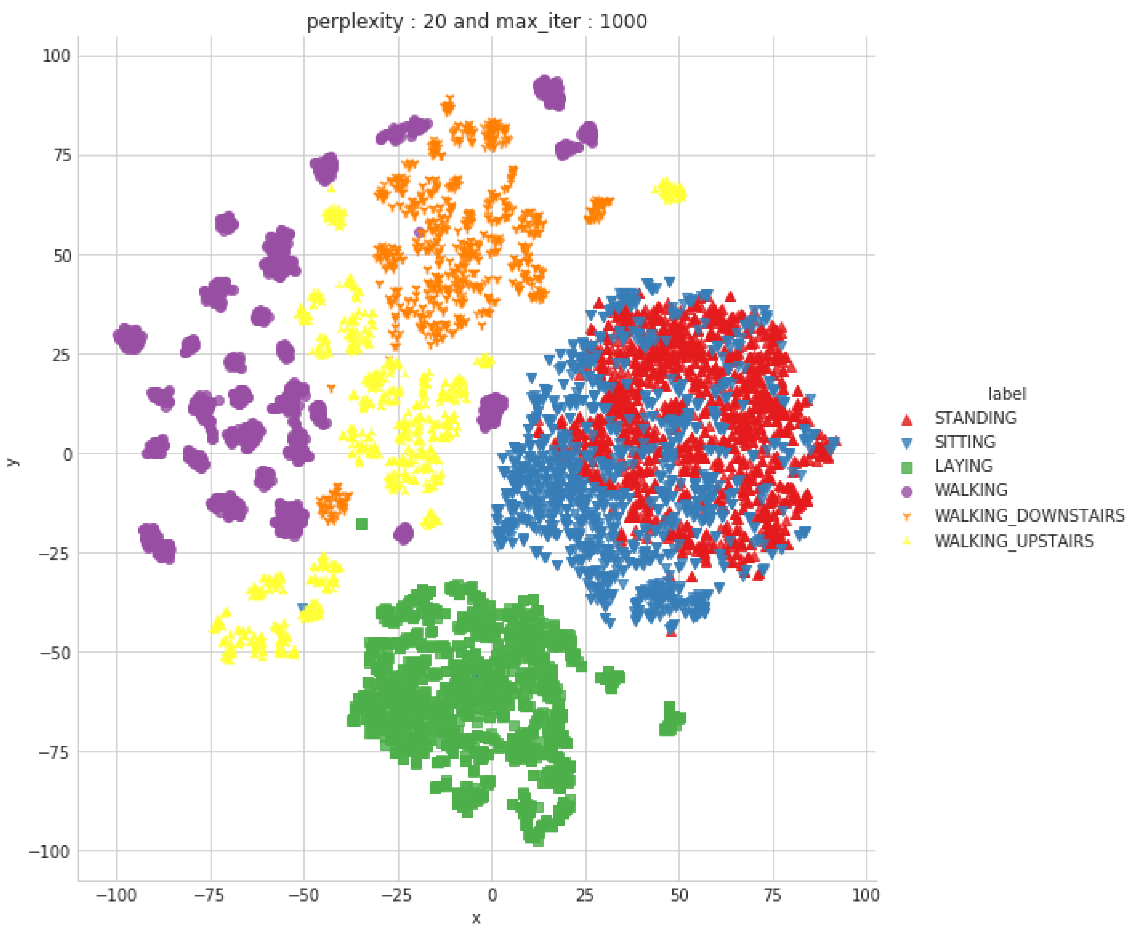

```
Done

performing tsne with perplexity 10 and with 1000 iterations at max
```

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




```
Done
```

```
performing tsne with perplexity 20 and with 1000 iterations at max
[t-SNE] Computing 61 nearest neighbors...
[t-SNE] Indexed 7352 samples in 0.425s...
[t-SNE] Computed neighbors for 7352 samples in 61.792s...
[t-SNE] Computed conditional probabilities for sample 1000 / 7352
[t-SNE] Computed conditional probabilities for sample 2000 / 7352
[t-SNE] Computed conditional probabilities for sample 3000 / 7352
[t-SNE] Computed conditional probabilities for sample 4000 / 7352
[t-SNE] Computed conditional probabilities for sample 5000 / 7352
[t-SNE] Computed conditional probabilities for sample 6000 / 7352
[t-SNE] Computed conditional probabilities for sample 7000 / 7352
[t-SNE] Computed conditional probabilities for sample 7352 / 7352
[t-SNE] Mean sigma: 1.274335
[t-SNE] Computed conditional probabilities in 0.355s
[t-SNE] Iteration 50: error = 97.5202179, gradient norm = 0.0223863 (50 iterations in 21.168s)
[t-SNE] Iteration 100: error = 83.9500732, gradient norm = 0.0059110 (50 iterations in 17.306s)
[t-SNE] Iteration 150: error = 81.8804779, gradient norm = 0.0035797 (50 iterations in 14.258s)
[t-SNE] Iteration 200: error = 81.1615143, gradient norm = 0.0022536 (50 iterations in 14.130s)
[t-SNE] Iteration 250: error = 80.7704086, gradient norm = 0.0018108 (50 iterations in 15.340s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 80.770409
[t-SNE] Iteration 300: error = 2.6957574, gradient norm = 0.0012993 (50 iterations in 13.605s)
[t-SNE] Iteration 350: error = 2.1637220, gradient norm = 0.0005765 (50 iterations in 13.248s)
[t-SNE] Iteration 400: error = 1.9143614, gradient norm = 0.0003474 (50 iterations in 14.774s)
[t-SNE] Iteration 450: error = 1.7684202, gradient norm = 0.0002458 (50 iterations in 15.502s)
[t-SNE] Iteration 500: error = 1.6744757, gradient norm = 0.0001923 (50 iterations in 14.808s)
[t-SNE] Iteration 550: error = 1.6101606, gradient norm = 0.0001575 (50 iterations in 14.043s)
[t-SNE] Iteration 600: error = 1.5641028, gradient norm = 0.0001344 (50 iterations in 15.769s)
[t-SNE] Iteration 650: error = 1.5291905, gradient norm = 0.0001182 (50 iterations in 15.834s)
[t-SNE] Iteration 700: error = 1.5024391, gradient norm = 0.0001055 (50 iterations in 15.398s)
[t-SNE] Iteration 750: error = 1.4809053, gradient norm = 0.0000965 (50 iterations in 14.594s)
[t-SNE] Iteration 800: error = 1.4631859, gradient norm = 0.0000884 (50 iterations in 15.025s)
[t-SNE] Iteration 850: error = 1.4486470, gradient norm = 0.0000832 (50 iterations in 14.060s)
[t-SNE] Iteration 900: error = 1.4367288, gradient norm = 0.0000804 (50 iterations in 12.389s)
[t-SNE] Iteration 950: error = 1.4270191, gradient norm = 0.0000761 (50 iterations in 10.392s)
[t-SNE] Iteration 1000: error = 1.4189968, gradient norm = 0.0000787 (50 iterations in 12.355s)
[t-SNE] Error after 1000 iterations: 1.418997
Done..
Creating plot for this t-sne visualization..
saving this plot as image in present working directory...
```




```
Done
```

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


```
Done
```

1.standing and sitting are not well classified for visualization

2.laying is fully classified

In [22]:

```
# 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'])
```

In [44]:

```
# Data directory
DATADIR = r'C:\Users\sagar\HumanActivityRecognition (1)\HAR\UCI_HAR_Dataset'
```

In [45]:

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

In [71]:

```
# 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'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))
```

In [72]:

```
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'UCI_HAR_Dataset/{subset}/y_{subset}.txt'
    y = _read_csv(filename)[0]

    return pd.get_dummies(y).as_matrix()
```

In [73]:

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

    return X_train, X_test, y_train, y_test
```

In [74]:

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

In [75]:

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

In [76]:

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

In [77]:

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

In [92]:

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

In [93]:

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

In [95]:

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

```
C:\Users\sagar\Anaconda3\lib\site-packages\ipykernel_launcher.py:12: FutureWarning: Method
.as_matrix will be removed in a future version. Use .values instead.
  if sys.path[0] == '':
```

In [96]:

```
X_train.shape
```

Out[96]:

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

In [97]:

```
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_layer = 1, hidden_layers = 64

In [105]:

```
#LSTM layer:1
#units : 64


hidden_layer = 64

# Initiliazing the sequential model

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

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

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_5 (LSTM)                (None, 64)                18944
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 6)                 390
=================================================================
Total params: 19,334
Trainable params: 19,334
Non-trainable params: 0
_________________________________________________________________
```

In [106]:

In [106]:

```
# Training the model
hist_1=model.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=epochs)
# Final evaluation of the model
scores_1 = model.evaluate(X_test, Y_test, verbose = 1)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/15
7352/7352 [==============================] - 55s 7ms/step - loss: 1.2215 - acc: 0.4521 - val_loss:
1.2634 - val_acc: 0.4194
Epoch 2/15
7352/7352 [==============================] - 54s 7ms/step - loss: 0.9640 - acc: 0.5816 - val_loss:
0.9309 - val_acc: 0.5786
Epoch 3/15
7352/7352 [==============================] - 59s 8ms/step - loss: 0.6091 - acc: 0.7431 - val_loss:
0.5257 - val_acc: 0.8324
Epoch 4/15
7352/7352 [==============================] - 60s 8ms/step - loss: 0.4501 - acc: 0.8523 - val_loss:
0.8609 - val_acc: 0.7441
Epoch 5/15
7352/7352 [==============================] - 63s 9ms/step - loss: 0.3551 - acc: 0.8853 - val_loss:
0.4354 - val_acc: 0.8592
Epoch 6/15
7352/7352 [==============================] - 62s 8ms/step - loss: 0.3335 - acc: 0.8894 - val_loss:
0.3607 - val_acc: 0.8836
Epoch 7/15
7352/7352 [==============================] - 63s 9ms/step - loss: 0.2145 - acc: 0.9230 - val_loss:
0.2386 - val_acc: 0.9070
Epoch 8/15
7352/7352 [==============================] - 61s 8ms/step - loss: 0.2020 - acc: 0.9295 - val_loss:
0.2371 - val_acc: 0.9097
Epoch 9/15
7352/7352 [==============================] - 65s 9ms/step - loss: 0.1798 - acc: 0.9388 - val_loss:
0.2206 - val_acc: 0.9104
Epoch 10/15
7352/7352 [==============================] - 63s 9ms/step - loss: 0.1716 - acc: 0.9369 - val_loss:
0.2307 - val_acc: 0.9077
Epoch 11/15
7352/7352 [==============================] - 62s 8ms/step - loss: 0.1547 - acc: 0.9449 - val_loss:
0.3987 - val_acc: 0.8887
Epoch 12/15
7352/7352 [==============================] - 62s 8ms/step - loss: 0.1613 - acc: 0.9434 - val_loss:
0.3868 - val_acc: 0.9050
Epoch 13/15
7352/7352 [==============================] - 63s 9ms/step - loss: 0.1543 - acc: 0.9433 - val_loss:
0.2554 - val_acc: 0.9118
Epoch 14/15
7352/7352 [==============================] - 62s 8ms/step - loss: 0.1512 - acc: 0.9452 - val_loss:
0.3410 - val_acc: 0.9077
Epoch 15/15
7352/7352 [==============================] - 58s 8ms/step - loss: 0.1601 - acc: 0.9433 - val_loss:
0.2643 - val_acc: 0.9169
2947/2947 [==============================] - 3s 1ms/step
```

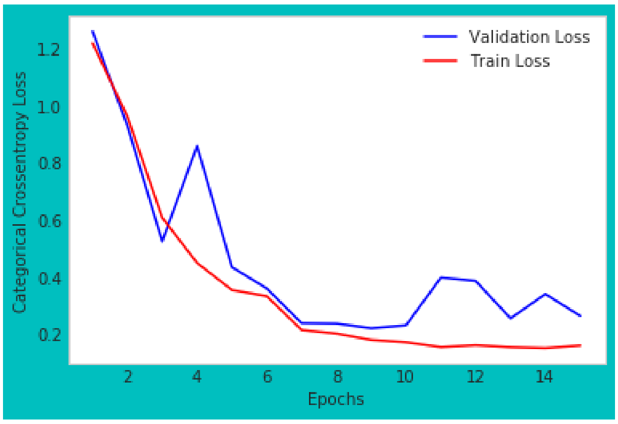
In [110]:

```
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334
# this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty):
  fig = plt.figure( facecolor='c', edgecolor='k')
  plt.plot(x, vy, 'b', label="Validation Loss")
  plt.plot(x, ty, 'r', label="Train Loss")
  plt.xlabel('Epochs')
  plt.ylabel('Categorical Crossentropy Loss')
  plt.legend()
  plt.grid()
  plt.show()
```

In [111]:

```
print("Test Score: %f" % (scores_1[0]))
test_acc1= scores[1]*100
train_acc1=(max(hist_1.history['acc']))* 100
print("Train Accuracy: %f%%"% (train_acc1))

print("Test Accuracy: %f%%" % (test_acc1))

# error plot
x=list(range(1,epochs+1))
vy=hist_1.history['val_loss'] #validation loss
ty=hist_1.history['loss'] # train loss
plt_dynamic(x, vy, ty)
```

```
Test Score: 0.264262
Train Accuracy: 94.518498%
Test Accuracy: 60.230743%
```



In [112]:

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

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 521        0         0        0                   0
SITTING                  0      409        81        0                   0
STANDING                 0      107       423        2                   0
WALKING                  0        0         0      478                   2
WALKING_DOWNSTAIRS       0        0         0        4                 416
WALKING_UPSTAIRS         0        0         0       15                   1

Pred                WALKING_UPSTAIRS
True
LAYING                            16
SITTING                            1
STANDING                           0
WALKING                           16
WALKING_DOWNSTAIRS                 0
WALKING_UPSTAIRS                 455
```

1. according to the confusion matrix LAYING is the correctly classified

2. walking can be some noise so not well classified

100_lstm_layers,adam optimizer

In [115]:

```
#LSTM layers:2
#units :100
#dropout rate : 0.50
```

```
# Initiliazing the sequential model


hidden_layer_2 = 100


model_1 = Sequential()
# Configuring the parameters
model_1.add(LSTM(hidden_layer_2, input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model_1.add(Dropout(0.50))
# Adding a dense output layer with sigmoid activation
model_1.add(Dense(n_classes, activation='sigmoid'))
model_1.summary()

# Compiling the model
model_1.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
# Training the model
hist2=model_1.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=epochs)
```

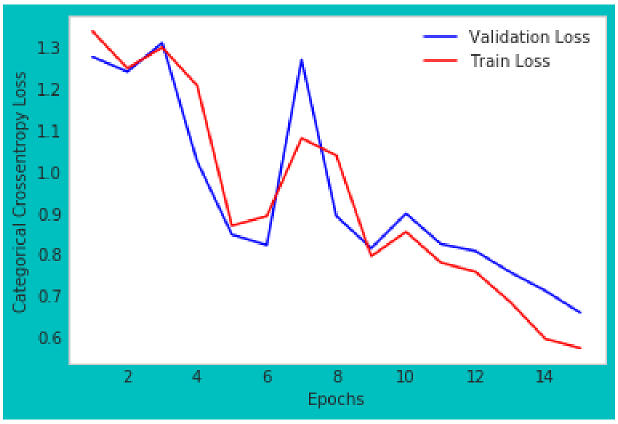
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_7 (LSTM)                (None, 100)               44000
_________________________________________________________________
dropout_7 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_7 (Dense)              (None, 6)                 606
=================================================================
Total params: 44,606
Trainable params: 44,606
Non-trainable params: 0
_________________________________________________________________
Train on 7352 samples, validate on 2947 samples
Epoch 1/15
7352/7352 [==============================] - 70s 9ms/step - loss: 1.3365 - acc: 0.4120 - val_loss:
1.2754 - val_acc: 0.4622
Epoch 2/15
7352/7352 [==============================] - 78s 11ms/step - loss: 1.2479 - acc: 0.4539 - val_loss
: 1.2395 - val_acc: 0.4313
Epoch 3/15
7352/7352 [==============================] - 83s 11ms/step - loss: 1.2979 - acc: 0.3979 - val_loss
: 1.3091 - val_acc: 0.3838
Epoch 4/15
7352/7352 [==============================] - 83s 11ms/step - loss: 1.2064 - acc: 0.4759 - val_loss
: 1.0247 - val_acc: 0.5606
Epoch 5/15
7352/7352 [==============================] - 79s 11ms/step - loss: 0.8697 - acc: 0.5864 - val_loss
: 0.8483 - val_acc: 0.4897
Epoch 6/15
7352/7352 [==============================] - 80s 11ms/step - loss: 0.8925 - acc: 0.5570 - val_loss
: 0.8229 - val_acc: 0.6037
Epoch 7/15
7352/7352 [==============================] - 78s 11ms/step - loss: 1.0799 - acc: 0.5076 - val_loss
: 1.2687 - val_acc: 0.4130
Epoch 8/15
7352/7352 [==============================] - 81s 11ms/step - loss: 1.0382 - acc: 0.5057 - val_loss
: 0.8928 - val_acc: 0.5830
Epoch 9/15
7352/7352 [==============================] - 80s 11ms/step - loss: 0.7964 - acc: 0.5820 - val_loss
: 0.8153 - val_acc: 0.6328
Epoch 10/15
7352/7352 [==============================] - 83s 11ms/step - loss: 0.8549 - acc: 0.5952 - val_loss
: 0.8990 - val_acc: 0.5976
Epoch 11/15
7352/7352 [==============================] - 82s 11ms/step - loss: 0.7811 - acc: 0.6001 - val_loss
: 0.8255 - val_acc: 0.5965
Epoch 12/15
```

```
7352/7352 [==============================] - 87s 12ms/step - loss: 0.7588 - acc: 0.6307 - val_loss
: 0.8088 - val_acc: 0.6020
Epoch 13/15
7352/7352 [==============================] - 91s 12ms/step - loss: 0.6856 - acc: 0.7110 - val_loss
: 0.7576 - val_acc: 0.6681
Epoch 14/15
7352/7352 [==============================] - 89s 12ms/step - loss: 0.5981 - acc: 0.7686 - val_loss
: 0.7134 - val_acc: 0.6932
Epoch 15/15
7352/7352 [==============================] - 87s 12ms/step - loss: 0.5758 - acc: 0.7669 - val_loss
: 0.6606 - val_acc: 0.7248
```

In [116]:

```
scores = model_1.evaluate(X_test, Y_test, verbose=1)
print("Test Score: %f" % (scores[0]))
test_acc2= scores[1]*100
train_acc2=(max(hist2.history['acc']))* 100
print("Train Accuracy: %f%%"% (train_acc2))

print("Test Accuracy: %f%%" % (test_acc2))
# error plot
x=list(range(1,epochs+1))
vy=hist2.history['val_loss'] #validation loss
ty=hist2.history['loss'] # train loss
plt_dynamic(x, vy, ty)
```

```
2947/2947 [==============================] - 4s 2ms/step
Test Score: 0.660586
Train Accuracy: 76.863439%
Test Accuracy: 72.480489%
```



In [117]:

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

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 510        0         0        5                   0
SITTING                  0      242       240        4                   1
STANDING                 0       20       507        2                   0
WALKING                  0        0         0      466                   5
WALKING_DOWNSTAIRS       0        0         1      132                 218
WALKING_UPSTAIRS         0        0         1      246                  31

Pred                WALKING_UPSTAIRS
True
LAYING                            22
SITTING                            4
STANDING                           3
WALKING                           25
WALKING_DOWNSTAIRS                69
WALKING_UPSTAIRS                 193
```

150 LSTM layers with dropout 0.75 and rmsprop optimizer

150_LSTM_layers with dropout 0.75 and rmsprop optimizer

In [118]:

```
#LSTM:3
#units:150
#dropout:0.75

hidden_layer_3 = 150



# Initiliazing the sequential model
model_2 = Sequential()
# Configuring the parameters
model_2.add(LSTM(hidden_layer_3, input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model_2.add(Dropout(0.75))
# Adding a dense output layer with sigmoid activation
model_2.add(Dense(n_classes, activation='sigmoid'))
model_2.summary()

# Compiling the model
model_2.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
# Training the model
hist_3=model_2.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=epochs)
```

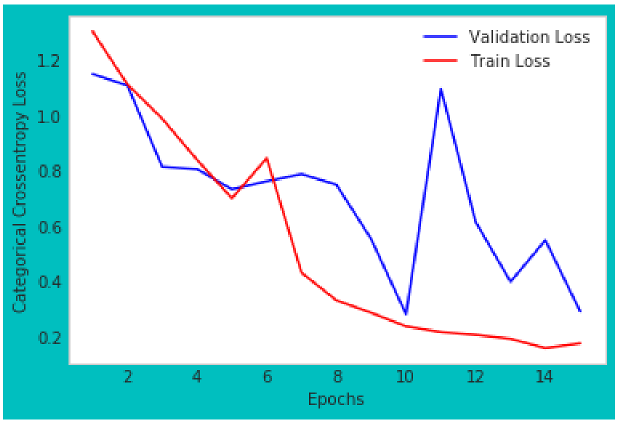
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_8 (LSTM)                (None, 150)               96000     
_________________________________________________________________
dropout_8 (Dropout)          (None, 150)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 6)                 906       
=================================================================
Total params: 96,906
Trainable params: 96,906
Non-trainable params: 0
_________________________________________________________________
Train on 7352 samples, validate on 2947 samples
Epoch 1/15
7352/7352 [==============================] - 122s 17ms/step - loss: 1.3050 - acc: 0.4255 - val_los
s: 1.1507 - val_acc: 0.5402
Epoch 2/15
7352/7352 [==============================] - 115s 16ms/step - loss: 1.1127 - acc: 0.5200 - val_los
s: 1.1086 - val_acc: 0.5212
Epoch 3/15
7352/7352 [==============================] - 107s 15ms/step - loss: 0.9888 - acc: 0.5642 - val_los
s: 0.8143 - val_acc: 0.6210
Epoch 4/15
7352/7352 [==============================] - 111s 15ms/step - loss: 0.8395 - acc: 0.6442 - val_los
s: 0.8060 - val_acc: 0.6797
Epoch 5/15
7352/7352 [==============================] - 112s 15ms/step - loss: 0.7005 - acc: 0.7191 - val_los
s: 0.7331 - val_acc: 0.7669
Epoch 6/15
7352/7352 [==============================] - 112s 15ms/step - loss: 0.8454 - acc: 0.7296 - val_los
s: 0.7623 - val_acc: 0.7628
Epoch 7/15
7352/7352 [==============================] - 110s 15ms/step - loss: 0.4309 - acc: 0.8541 - val_los
s: 0.7880 - val_acc: 0.8018
Epoch 8/15
7352/7352 [==============================] - 113s 15ms/step - loss: 0.3310 - acc: 0.8940 - val_los
s: 0.7500 - val_acc: 0.7991
Epoch 9/15
7352/7352 [==============================] - 109s 15ms/step - loss: 0.2858 - acc: 0.9127 - val_los
s: 0.5522 - val_acc: 0.8765
Epoch 10/15
7352/7352 [==============================] - 105s 14ms/step - loss: 0.2370 - acc: 0.9230 - val_los
s: 0.2805 - val_acc: 0.9074
```

```
Epoch 11/15
7352/7352 [==============================] - 112s 15ms/step - loss: 0.2160 - acc: 0.9272 - val_los
s: 1.0972 - val_acc: 0.8018
Epoch 12/15
7352/7352 [==============================] - 108s 15ms/step - loss: 0.2061 - acc: 0.9338 - val_los
s: 0.6147 - val_acc: 0.8510
Epoch 13/15
7352/7352 [==============================] - 107s 15ms/step - loss: 0.1915 - acc: 0.9391 - val_los
s: 0.3982 - val_acc: 0.9013
Epoch 14/15
7352/7352 [==============================] - 108s 15ms/step - loss: 0.1581 - acc: 0.9404 - val_los
s: 0.5489 - val_acc: 0.8992
Epoch 15/15
7352/7352 [==============================] - 111s 15ms/step - loss: 0.1754 - acc: 0.9419 - val_los
s: 0.2906 - val_acc: 0.9009
```

In [119]:

```
scores = model_2.evaluate(X_test, Y_test, verbose=1)
print("Test Score: %f" % (scores[0]))
test_acc3= scores[1]*100
train_acc3=(max(hist_3.history['acc']))* 100
print("Train Accuracy: %f%%"% (train_acc3))

print("Test Accuracy: %f%%" % (test_acc3))
# error plot
x=list(range(1,epochs+1))
vy=hist_3.history['val_loss'] #validation loss
ty=hist_3.history['loss'] # train loss
plt_dynamic(x, vy, ty)
```

```
2947/2947 [==============================] - 7s 2ms/step
Test Score: 0.290615
Train Accuracy: 94.192057%
Test Accuracy: 90.091619%
```



In [120]:

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

```
Pred                LAYING  SITTING  STANDING  WALKING  WALKING_DOWNSTAIRS  \
True
LAYING                 513        0        24        0                   0
SITTING                  0      386       105        0                   0
STANDING                 0       98       433        1                   0
WALKING                  0        0         0      467                  25
WALKING_DOWNSTAIRS       0        0         0        6                 402
WALKING_UPSTAIRS         0        1         8        7                   1

Pred                WALKING_UPSTAIRS
True
LAYING                             0
SITTING                            0
STANDING                           0
WALKING                            4
WALKING_DOWNSTAIRS                12
WALKING_UPSTAIRS                 454
```

In [121]:

```
#LSTM:3
#units:150
#dropout:0.75

hidden_layer_4 = 150



# Initiliazing the sequential model
model_4 = Sequential()
# Configuring the parameters
model_4.add(LSTM(hidden_layer_3, input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model_4.add(Dropout(0.75))
# Adding a dense output layer with sigmoid activation
model_4.add(Dense(n_classes, activation='sigmoid'))
model_4.summary()

# Compiling the model
model_4.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
# Training the model
hist_4=model_4.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=epochs)
```

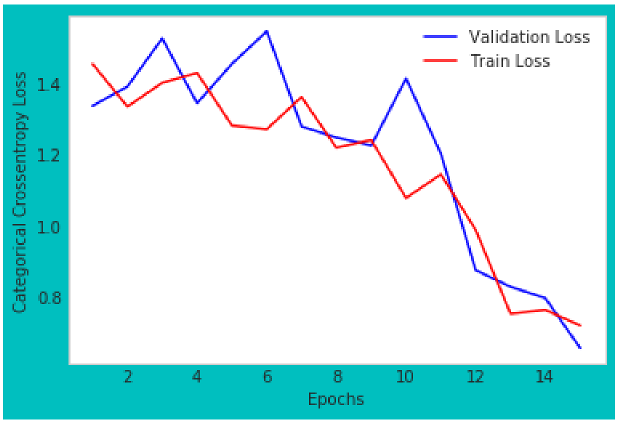
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_9 (LSTM)                (None, 150)               96000
_________________________________________________________________
dropout_9 (Dropout)          (None, 150)               0
_________________________________________________________________
dense_9 (Dense)              (None, 6)                 906
=================================================================
Total params: 96,906
Trainable params: 96,906
Non-trainable params: 0
_________________________________________________________________
Train on 7352 samples, validate on 2947 samples
Epoch 1/15
7352/7352 [==============================] - 111s 15ms/step - loss: 1.4544 - acc: 0.3546 - val_los
s: 1.3370 - val_acc: 0.3855
Epoch 2/15
7352/7352 [==============================] - 107s 15ms/step - loss: 1.3349 - acc: 0.3870 - val_los
s: 1.3909 - val_acc: 0.3536
Epoch 3/15
7352/7352 [==============================] - 95s 13ms/step - loss: 1.4016 - acc: 0.3619 - val_loss
: 1.5260 - val_acc: 0.3207
Epoch 4/15
7352/7352 [==============================] - 99s 13ms/step - loss: 1.4291 - acc: 0.3655 - val_loss
: 1.3443 - val_acc: 0.4177
Epoch 5/15
7352/7352 [==============================] - 104s 14ms/step - loss: 1.2822 - acc: 0.4584 - val_los
s: 1.4545 - val_acc: 0.4204
Epoch 6/15
7352/7352 [==============================] - 107s 15ms/step - loss: 1.2718 - acc: 0.4441 - val_los
s: 1.5456 - val_acc: 0.3485
Epoch 7/15
7352/7352 [==============================] - 109s 15ms/step - loss: 1.3619 - acc: 0.4014 - val_los
s: 1.2795 - val_acc: 0.4696
Epoch 8/15
7352/7352 [==============================] - 108s 15ms/step - loss: 1.2199 - acc: 0.4642 - val_los
s: 1.2487 - val_acc: 0.4123
Epoch 9/15
7352/7352 [==============================] - 96s 13ms/step - loss: 1.2415 - acc: 0.4354 - val_loss
: 1.2262 - val_acc: 0.4058
Epoch 10/15
7352/7352 [==============================] - 104s 14ms/step - loss: 1.0793 - acc: 0.4827 - val_los
```

```
s: 1.4135 - val_acc: 0.4533
Epoch 11/15
7352/7352 [==============================] - 114s 16ms/step - loss: 1.1456 - acc: 0.4909 - val_los
s: 1.2033 - val_acc: 0.4561
Epoch 12/15
7352/7352 [==============================] - 126s 17ms/step - loss: 0.9899 - acc: 0.5597 - val_los
s: 0.8776 - val_acc: 0.6030
Epoch 13/15
7352/7352 [==============================] - 121s 16ms/step - loss: 0.7555 - acc: 0.6253 - val_los
s: 0.8310 - val_acc: 0.5928
Epoch 14/15
7352/7352 [==============================] - 146s 20ms/step - loss: 0.7661 - acc: 0.6175 - val_los
s: 0.7992 - val_acc: 0.6020
Epoch 15/15
7352/7352 [==============================] - 133s 18ms/step - loss: 0.7229 - acc: 0.6298 - val_los
s: 0.6596 - val_acc: 0.6138
```

In [126]:

```
scores = model_4.evaluate(X_test, Y_test, verbose=1)
print("Test Score: %f" % (scores[0]))
test_acc4= scores[1]*100
train_acc4=(max(hist_4.history['acc']))* 100
print("Train Accuracy: %f%%"% (train_acc4))

print("Test Accuracy: %f%%" % (test_acc4))
# error plot
x=list(range(1,epochs+1))
vy=hist_4.history['val_loss'] #validation loss
ty=hist_4.history['loss'] # train loss
plt_dynamic(x, vy, ty)
```

```
2947/2947 [==============================] - 6s 2ms/step
Test Score: 0.659581
Train Accuracy: 62.976061%
Test Accuracy: 61.384459%
```



In [127]:

```
# Confusion Matrix
print(confusion_matrix(Y_test, model_4.predict(X_test)))
```

```
Pred                LAYING  SITTING  STANDING  WALKING
True
LAYING                 520       12         0        5
SITTING                  0      358       126        7
STANDING                 0       85       435       12
WALKING                  0        0         0      496
WALKING_DOWNSTAIRS       0        0         0      420
WALKING_UPSTAIRS         0        0         1      470
```

In [128]:

```
from prettytable import PrettyTable

print('\n')
a = PrettyTable()
```

```
a.field_names = ['S.No', 'LSTM Units', 'LSTM Layers','Drop Out','optimizer', 'Training accuracy', '
Test Accuracy']
a.add_row([1, 64, 1, 0.25,'rmsprop', train_acc1, test_acc1])
a.add_row([2, 100, 2, 0.5,'adam', train_acc2, test_acc2])
a.add_row([3, 150, 3, 0.75,'rmsprop', train_acc3, test_acc3])
a.add_row([4, 150, 3, 0.75,'adm', train_acc4, test_acc4])

print(a.get_string(title = "LSTM 2 and 4 Activation: sigmoid,    Optimizer: adam"))
```

```
+------+------------+-------------+----------+-----------+-------------------+--------------------+
| S.No | LSTM Units | LSTM Layers | Drop Out | optimizer | Training accuracy |   Test Accuracy    
|
+------+------------+-------------+----------+-----------+-------------------+--------------------+
|  1   |     64     |      1      |   0.25   |  rmsprop  | 94.51849836779108 | 60.230743129616634 |
|  2   |    100     |      2      |   0.5    |    adam   | 76.86343852013057 | 72.48048863250763  |
|  3   |    150     |      3      |   0.75   |  rmsprop  | 62.97606093579978 | 61.38445877163217  |
|  4   |    150     |      3      |   0.75   |    adm    | 62.97606093579978 | 61.38445877163217  |
+------+------------+-------------+----------+-----------+-------------------+--------------------+
```

observation: 1.for the layer first with 64 hidden layer we get traingin accuracy high but test accuracy get to low so it will not give best performance

2.for the layer 2 with 100 hidden layer we get probalbly same train and test accuracy but we can still improve the performance by the increasing the number of ephoc steps

2.adam optimizer is giving roughly same results as compare to the rmsprop optimizer