Data Mining Project Report

on

“HUMAN ACTIVITY RECOGNITION USING RECURRENT NEURAL NETWORK(LONG SHORT TERM MEMORY NETWORKS)”

Submitted to

IIT BHU Varanasi

BY

Neha Kumari	**18074012**
Kumar Shivam Ranjan	**18075031**
Madhav Bansal	**18075036**

UNDER THE GUIDANCE OF

Dr. Bhaskar Biswas

DEPARTMENT OF COMPUTER ENGINEERING

IIT BHU Varanasi

IIT BHU(Varanasi)

Department of Computer Science Engineering
LOCATION IN Banaras,UP – 221005

CERTIFICATE

This is certify that the project entitled

“Human Activity Recognition using recurrent neural network(Long short term memory networks)“

submitted by

Neha Kumari	**18074012**
Madhav Bansal	**18075036**
Kumar Shivam Ranjan	**18075031**

is a record of bonafide work carried out by them, in the partial fulfilment of the requirement for the award of Degree of Bachelor of Engineering (Computer Engineering) at IIT BHU Varanasi,Banaras. This work is done during year 2020-2021, under our guidance.

Date: 11/ 2/ 2020

(Dr. Bhaskar Biswas)
Project Guide

(Phd Scholar Shivansh Mishra)
Project Coordinator

Acknowledgements

We are profoundly grateful to **Dr. Bhaskar Biswas Sir** for his expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion.

We would like to express deepest appreciation towards Project Coordinator **Phd Scholar Mr. Shivansh Mishra**, whose invaluable guidance supported us in completing this project.

At last we must express our sincere heartfelt gratitude to all the staff members of Computer Engineering Department who helped me directly or indirectly during this course of work.

Kumar Shivam Ranjan
Neha Kumari
Madhav Bansal

ABSTRACT

Human Activity Recognition, is the problem of predicting what kind of activity a person is performing based on a signals detected by smartphone sensors on their waist.

Two types of sensors present in smartphones are:
1) Accelerometer
2) Gyroscope

Accelerometer measures acceleration and **Gyroscope** measures angular velocity

Contents

List of Figures

Chapter 1

About the Dataset

1.1 UCI HAR Dataset

1.1.1 Quick Overview of the Dataset

1) Data is downloaded from following source: **Link to Dataset**.
2. Feature names are present in UCI HAR dataset/features.txt
3. **Train Data**
a) UCI HAR dataset/train/X train.txt
b) UCI HAR dataset/train/subject train.txt
c) UCI HAR dataset/train/y train.txt
4. **Test Data**
a) UCI HAR dataset/test/X test.txt
b) UCI HAR dataset/test/subject test.txt
c) UCI HAR dataset/test/y test.txt

1.1.2 How the data was prepared?

30 volunteers referred to as subjects performed the experiment for data collection wearing smartphones sensors on their waist.
The two smartphone sensors captured the 3 axial linear acceleration as well as the 3 axial angular velocity of the subject

Chapter 2

Business Problem

2.1 Problem Statement

Predict one of the following six activities that a Smartphone user is performing at that 2.56 Seconds time window by using either 561 feature data or raw features of 128 reading.

1. Walking
2. Walking Upstairs
3. Walking Downstairs
4. Sitting
5. Standing
6. Laying

Chapter 3

Additional Information

3.1 Dataset

3.1.1 Few more points to take into consideration

1. 561 feature vector were engineered from each time window of 2.56 second with both time and frequency domain variables
2. Features are normalized and bounded within [-1,1]
3. The gyroscopic data is measured in radian/sec The units used for the accelerations are 'g' (9.8
4. Accelerometer readings are divided into gravity acceleration and body acceleration readings, which has x,y and z components each.
5. Gyroscope readings are the measure of angular velocities which has x,y and z components
6. Fourier Transforms are made on the above time readings to obtain frequency readings.
7. Now, on all the base signal readings, mean, std,max, skewness, kurtosis, etc are calculated for each window.

Chapter 4

Model to solve Human Activity Recognition problem

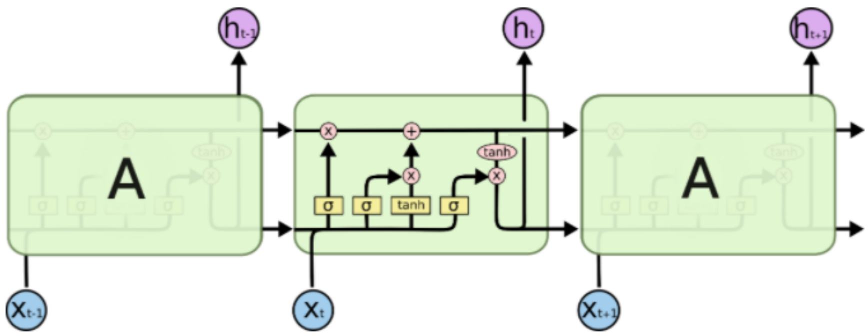
4.1 Deep Learning approach

1. Accelerometer data was divided into body acceleration and total acceleration (body = total - gravitational force)
2. We can use LSTM (long short term memory) model of the Recurrent Neural Network (RNN) to recognize various activities of humans like standing, climbing upstairs and downstairs etc.
3. LSTM model is a type of recurrent neural network.It helps us to look at recent information to perform the present task
4. LSTM works well on time-series data.
5. This model is used as this helps in remembering values over arbitrary intervals.

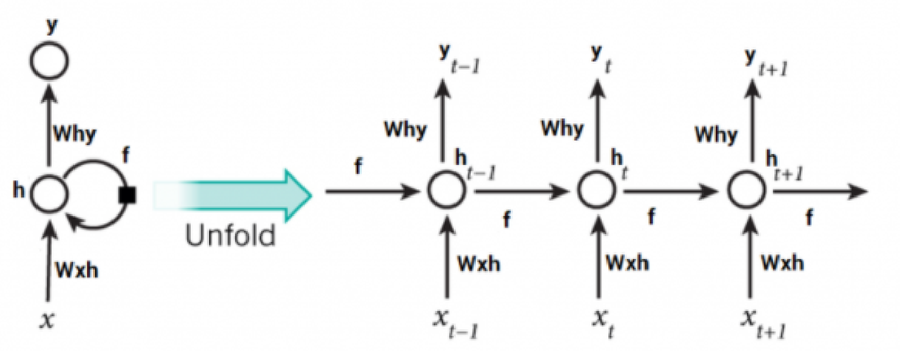
Chapter 5

Model Architecture

5.1 LSTM network design

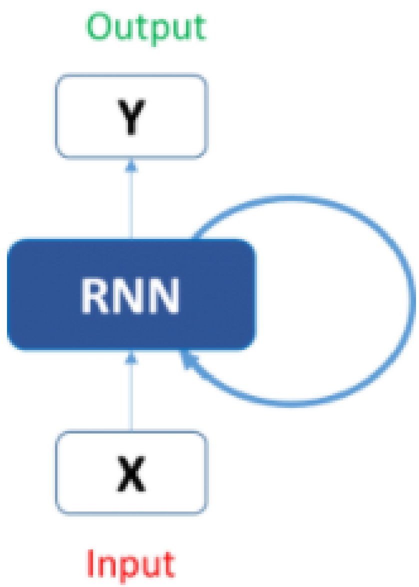


The repeating module in an LSTM contains four interacting layers.



Unfolded LSTM network

.



Typical RNN

.

Chapter 6

Plan of Action

1. We will apply classical machine learning models on 561 sized domain expert engineered features
2. As we know that LSTM works better on time series data or data that is sequential or contains sequence of information where order is very important, so we decided that we will apply LSTM of Recurrent neural networks on 128 sized raw readings that we obtained from accelerometer and gyroscope signals

6.0.1 Data Point Distribution

1. 30 subjects data is randomly split to 70 % (21) train and 30 % (9) test data
2. Each data point corresponds to one of the 6 activities

Chapter 7

Machine Learning Model

7.1 Logistic Regression

1. In Deep learning Model , we used raw features of 128 readings to predict the activity.The Dataset used there was basically a 3D matrix of(7352x128x9).
2. The Same Dataset cannot be used to train the machine learning model as RNNs are designed to work with time series data not machine learning.
3. This is why 561 features were engineered from raw 128 accelerometer and gyroscope signal readings so that we can use them in our machine learning model.
4. We will apply classical Machine Learning models on these 561 sized domain expert engineered features
5. As we know that LSTM works well on time-series data, so,we will apply LSTM of Recurrent Neural Networks on 128 sized raw readings that we obtained from accelerometer and gyroscope signals.

Chapter 8

Implementation

Output Labels

a. WALKING (1)

b. WALKING UPSTAIRS (2)

c. WALKING DOWNSTAIRS (3)

d. SITTING (4)

e. STANDING (5)

f. LYING (6)

We need to predict the output as one of the 6 labels the user is performing using either handcoded engineered 561 features or raw features of 128 readings.
Built With

1. **ipython-notebook** - Python Text Editor
2. **Anaconda** -Data Science platform
3. **Python Pip**- for installing python libraries
4. **Sklearn** -A Machine learning library for logistic Regression
5. **Seaborn** - Python Visualization library
6. **Matplotlib** - Python plotting library
7. **Numpy, scipy**- number python library
8. **Pandas** - data handling library
9. **keras and tensorflow** - Used for making deep learning models

Libraries used

```

import numpy as np        #numpy
import pandas as pd       #pandas
import matplotlib.pyplot as plt    #matplotlib
import seaborn as sns              #seaborn
import tensorflow as tf            #tensorflow
from keras.models import Sequential     #keras
from keras.layers import LSTM           #keras layers
from keras.layers.core import Dense, Dropout  #keras layers core
from keras.layers.normalization import BatchNormalization  #keras layers
    normalisation
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
import seaborn as sn
import matplotlib.pyplot as plt
```

Chapter 9

Screenshots of Project

9.1 Project Code

Obtaining the train data

```
X_train = pd.read_csv('UCI_HAR_Dataset/train/X_train.txt', delim_whitespace=True, header=None)


X_train['subject'] = pd.read_csv('UCI_HAR_Dataset/train/subject_train.txt', header=None, squeeze=True)

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

train = X_train
train['Activity'] = y_train
train['ActivityName'] = y_train_labels
train.head()
```

	0	1	2	3	4	5	6	7	8	9	...	554	555	556	557
0	0.288585	-0.020294	-0.132905	-0.995279	-0.983111	-0.913526	-0.995112	-0.983185	-0.923527	-0.934724	...	-0.112754	0.030400	-0.464761	-0.018446
1	0.278419	-0.016411	-0.123520	-0.998245	-0.975300	-0.960322	-0.998807	-0.974914	-0.957686	-0.943068	...	0.053477	-0.007435	-0.732626	0.703511
2	0.279653	-0.019467	-0.113462	-0.995380	-0.967187	-0.978944	-0.996520	-0.963668	-0.977469	-0.938692	...	-0.118559	0.177899	0.100699	0.808529
3	0.279174	-0.026201	-0.123283	-0.996091	-0.983403	-0.990675	-0.997099	-0.982750	-0.989302	-0.938692	...	-0.036788	-0.012892	0.640011	-0.485366
4	0.276629	-0.016570	-0.115362	-0.998139	-0.980817	-0.990482	-0.998321	-0.979672	-0.990441	-0.942469	...	0.123320	0.122542	0.693578	-0.615971

5 rows × 564 columns

Obtaining the test data

```
X_test = pd.read_csv('UCI_HAR_Dataset/test/X_test.txt', delim_whitespace=True, header=None)


X_test['subject'] = pd.read_csv('UCI_HAR_Dataset/test/subject_test.txt', header=None, squeeze=True)


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'})


test = X_test
test['Activity'] = y_test
test['ActivityName'] = y_test_labels
test.head()
```

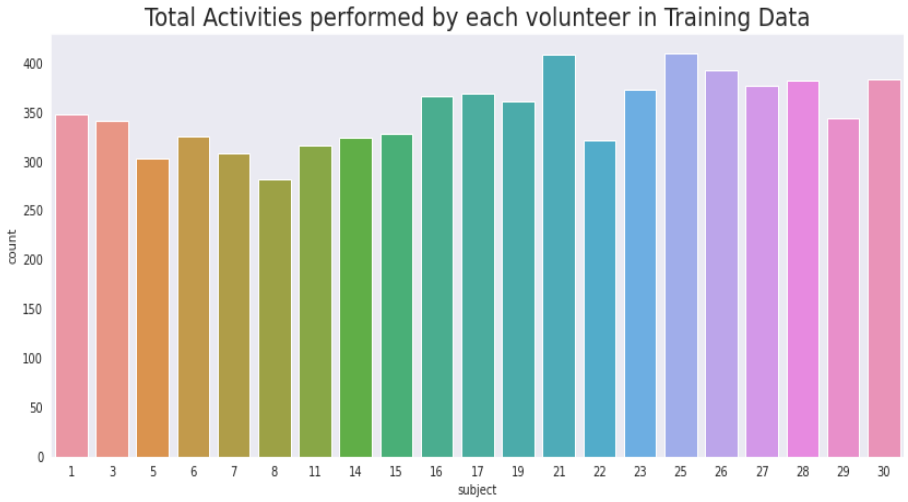
	0	1	2	3	4	5	6	7	8	9	...	554	555	556	557
0	0.257178	-0.023285	-0.014654	-0.938404	-0.920091	-0.667683	-0.952501	-0.925249	-0.674302	-0.894088	...	0.006462	0.162920	-0.825886	0.271151
1	0.286027	-0.013163	-0.119083	-0.975415	-0.967458	-0.944958	-0.986799	-0.968401	-0.945823	-0.894088	...	-0.083495	0.017500	-0.434375	0.920593
2	0.275485	-0.026050	-0.118152	-0.993819	-0.969926	-0.962748	-0.994403	-0.970735	-0.963483	-0.939260	...	-0.034956	0.202302	0.064103	0.145068
3	0.270298	-0.032614	-0.117520	-0.994743	-0.973268	-0.967091	-0.995274	-0.974471	-0.968897	-0.938610	...	-0.017067	0.154438	0.340134	0.296407
4	0.274833	-0.027848	-0.129527	-0.993852	-0.967445	-0.978295	-0.994111	-0.965953	-0.977346	-0.938610	...	-0.002223	-0.040046	0.736715	-0.118545

5 rows × 564 columns

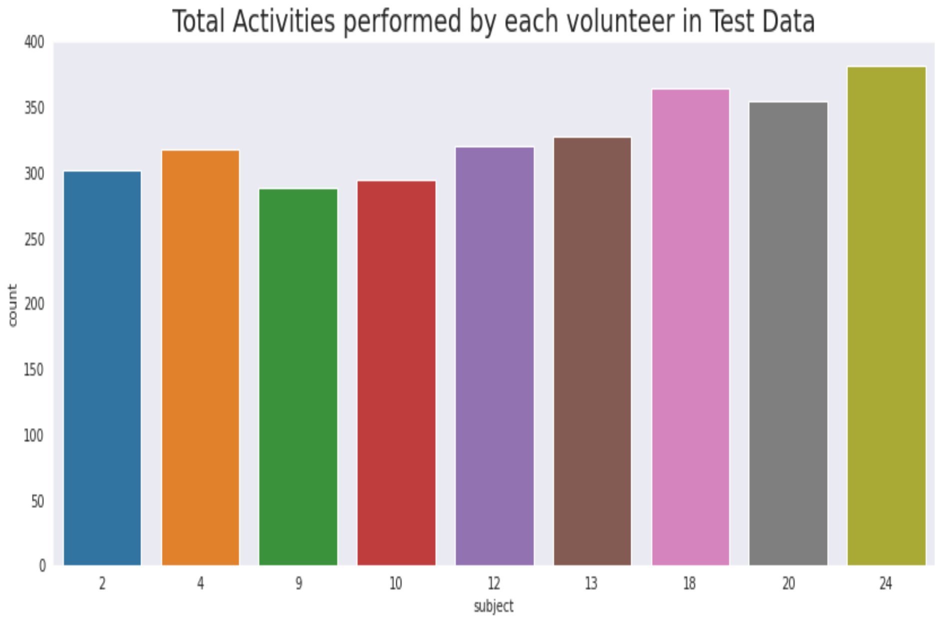
1. Visualising the Dataset

```
sns.set_style('dark')
plt.rcParams['lines.linewidth'] = 1
plt.rcParams['font.family'] = 'DejaVu Sans'
```

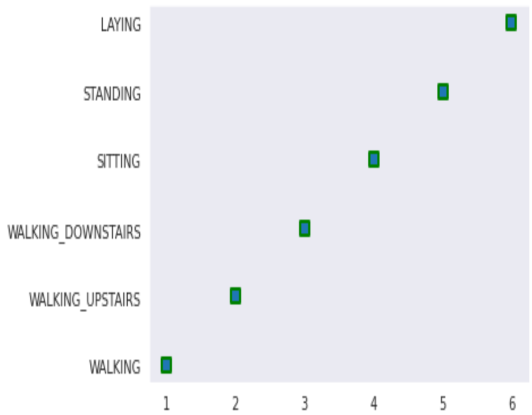
```
plt.figure(figsize=(14,6))
plt.title("Total Activities performed by each volunteer in Training Data",fontsize=20)
sns.countplot(x=train.subject)
plt.show()
```

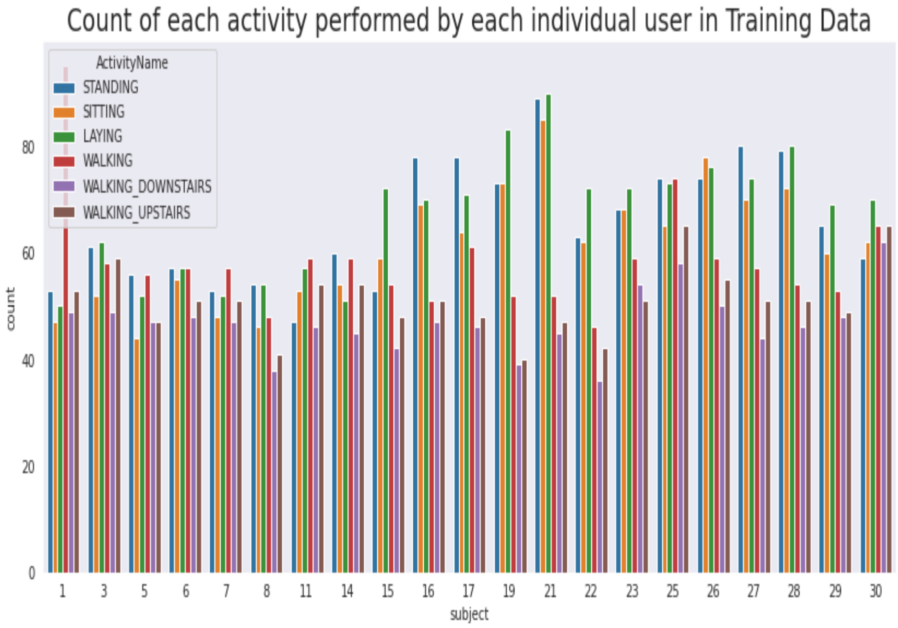


```
plt.figure(figsize=(14,6))
plt.title("Total Activities performed by each volunteer in Test Data",fontsize=20)
sns.countplot(x=test.subject)
plt.show()
```

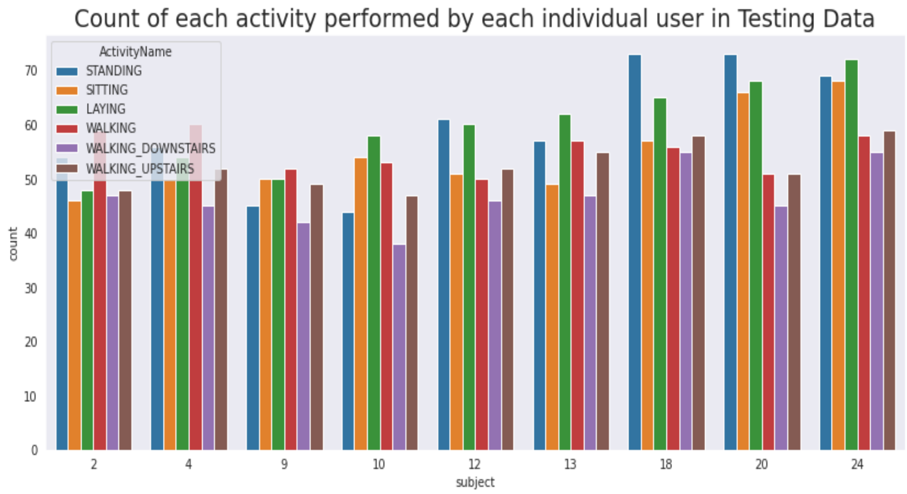



```
y_train_labels = {1: 'WALKING', 2:'WALKING_UPSTAIRS',3:'WALKING_DOWNSTAIRS',4:'SITTING', 5:'STANDING',6:'LAYING'}
lists=sorted(y_train_labels.items())
x, y = zip(*lists)
plt.scatter(x, y,linewidths = 2,marker ="s",edgecolor ="green", s = 50)
plt.show()
```

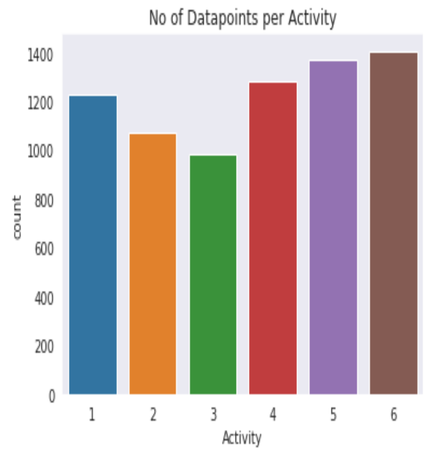





```
plt.figure(figsize=(14,6))
plt.title('Count of each activity performed by each individual user in Testing Data', fontsize=20)
sns.countplot(x='subject',hue='ActivityName', data = test)
plt.show()
```



```
plt.title('No of Datapoints per Activity')
sns.countplot(x=train.Activity)
plt.xticks(rotation=0)
plt.show()
```



LSTM model for HAR

```
np.random.seed(42)
import tensorflow as tf
tf.random.set_seed(42)
```

```
# Configuring a session
session_conf = tf.compat.v1.ConfigProto(
    intra_op_parallelism_threads=1,
    inter_op_parallelism_threads=1
)
```

```
# Import Keras
from tensorflow.compat.v1.keras import backend as K
sess = tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(), config=session_conf)
K.set_session(sess)
```

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

```
DATADIR = 'UCI_HAR_Dataset'

ACTIVITIES = {
    0: 'WALKING',
    1: 'WALKING_UPSTAIRS',
    2: 'WALKING_DOWNSTAIRS',
    3: 'SITTING',
    4: 'STANDING',
    5: 'LAYING',
}

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'])
```

```
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"
]
```

```
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

def generate_x_data(subset):
    signals_data = []

    for signal in SIGNALS:
        filename = f'UCI_HAR_Dataset/{subset}/Inertial Signals/{signal}_{subset}.txt'
        signals_data.append(_read_csv(filename).values)
    return np.transpose(signals_data, (1, 2, 0))


def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

```
def generate_y_data(subset):

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

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

```
def load_data():

    X_train, X_test = generate_x_data('train'), generate_x_data('test')
    y_train, y_test = generate_y_data('train'), generate_y_data('test')

    return X_train, X_test, y_train, y_test
```

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

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

```
7352
128
9
6
```

```
epochs = 30
batch_size = 32
n_hidden = 128
pv = 0.25
```

```
model = Sequential()

model.add(LSTM(n_hidden, input_shape=(timesteps, input_dim)))
model.add(BatchNormalization())

model.add(Dropout(pv))

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

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 128)               70656
_________________________________________________________________
batch_normalization (BatchNo (None, 128)               512
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 6)                 774
=================================================================
Total params: 71,942
Trainable params: 71,686
Non-trainable params: 256
```

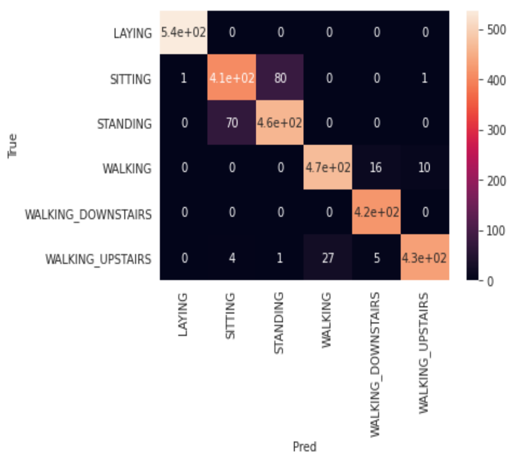
```
model.compile(loss='categorical_crossentropy', optimizer='rmsprop',metrics=['accuracy'])
```

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

```
Epoch 1/30
230/230 [==============================] - 17s 76ms/step - loss: 0.9775 - accuracy: 0.5635 - val_loss: 0.9544 - val_accuracy: 0.5836
Epoch 2/30
230/230 [==============================] - 16s 70ms/step - loss: 0.7032 - accuracy: 0.6602 - val_loss: 0.7071 - val_accuracy: 0.6566
Epoch 3/30
230/230 [==============================] - 17s 74ms/step - loss: 0.6357 - accuracy: 0.7076 - val_loss: 0.7523 - val_accuracy: 0.6203
Epoch 4/30
230/230 [==============================] - 17s 74ms/step - loss: 0.4320 - accuracy: 0.8298 - val_loss: 0.3244 - val_accuracy: 0.8894
Epoch 5/30
230/230 [==============================] - 17s 75ms/step - loss: 0.2150 - accuracy: 0.9214 - val_loss: 0.3481 - val_accuracy: 0.8823
Epoch 6/30
230/230 [==============================] - 17s 75ms/step - loss: 0.1855 - accuracy: 0.9320 - val_loss: 0.3101 - val_accuracy: 0.9006
Epoch 7/30
230/230 [==============================] - 17s 75ms/step - loss: 0.1877 - accuracy: 0.9328 - val_loss: 0.3312 - val_accuracy: 0.8887
Epoch 8/30
230/230 [==============================] - 17s 75ms/step - loss: 0.1598 - accuracy: 0.9377 - val_loss: 0.3471 - val_accuracy: 0.9019
Epoch 9/30
230/230 [==============================] - 17s 75ms/step - loss: 0.1444 - accuracy: 0.9377 - val_loss: 0.5204 - val_accuracy: 0.9087
Epoch 10/30
230/230 [==============================] - 17s 75ms/step - loss: 0.1532 - accuracy: 0.9421 - val_loss: 0.3719 - val_accuracy: 0.8911
Epoch 11/30
230/230 [==============================] - 17s 75ms/step - loss: 0.1623 - accuracy: 0.9429 - val_loss: 0.2438 - val_accuracy: 0.9240
Epoch 12/30
230/230 [==============================] - 17s 75ms/step - loss: 0.1344 - accuracy: 0.9436 - val_loss: 0.2704 - val_accuracy: 0.9125
Epoch 13/30
230/230 [==============================] - 17s 75ms/step - loss: 0.1401 - accuracy: 0.9437 - val_loss: 0.3632 - val_accuracy: 0.9036
Epoch 14/30
230/230 [==============================] - 17s 75ms/step - loss: 0.1305 - accuracy: 0.9437 - val_loss: 0.2415 - val_accuracy: 0.9138
Epoch 15/30
230/230 [==============================] - 17s 75ms/step - loss: 0.1403 - accuracy: 0.9438 - val_loss: 0.3694 - val_accuracy: 0.9067
Epoch 16/30
230/230 [==============================] - 17s 75ms/step - loss: 0.1271 - accuracy: 0.9475 - val_loss: 0.3459 - val_accuracy: 0.9128
Epoch 17/30
230/230 [==============================] - 17s 75ms/step - loss: 0.1247 - accuracy: 0.9464 - val_loss: 0.3513 - val_accuracy: 0.9179
Epoch 18/30
230/230 [==============================] - 17s 75ms/step - loss: 0.1455 - accuracy: 0.9479 - val_loss: 0.2809 - val_accuracy: 0.9118
Epoch 19/30
230/230 [==============================] - 17s 75ms/step - loss: 0.1230 - accuracy: 0.9467 - val_loss: 0.3377 - val_accuracy: 0.9237
Epoch 20/30
```

```
result=confusion_matrix(Y_test, model.predict(X_test))
sn.heatmap(result, annot=True)
```

```
<AxesSubplot:xlabel='Pred', ylabel='True'>
```




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

```
93/93 [==============================] - 2s 18ms/step - loss: 0.3387 - accuracy: 0.9270
```

```
score
```

```
[0.33865147829055786, 0.9270444512367249]
```

- With a simple semi tuned stacked LSTM architecture we got 92.7% accuracy and a loss of 0.33

LSTM Model2

stacking 2 LSTM

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

```
model1 = Sequential()

model1.add(LSTM(n_hidden1, return_sequences=True, input_shape=(timesteps, input_dim)))

model1.add(Dropout(pv1))

model1.add(LSTM(n_hidden2))

model1.add(Dropout(pv2))

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

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 128, 128)          70656
_________________________________________________________________
dropout_1 (Dropout)          (None, 128, 128)          0
_________________________________________________________________
lstm_2 (LSTM)                (None, 64)                49408
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0



_________________________________________________________________
```

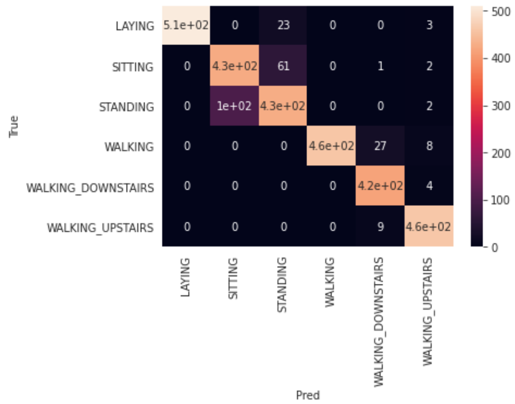
```
model1.compile(loss='categorical_crossentropy',optimizer='rmsprop',metrics=['accuracy'])
```

```
model1.fit(X_train,Y_train,batch_size=batch_size,validation_data=(X_test, Y_test),epochs=epochs1)
```

```
Epoch 1/30
230/230 [==============================] - 30s 129ms/step - loss: 1.1419 - accuracy: 0.5018 - val_loss: 0.9634 - val_accuracy: 0.5304
Epoch 2/30
230/230 [==============================] - 29s 126ms/step - loss: 0.9308 - accuracy: 0.5928 - val_loss: 0.8050 - val_accuracy: 0.6220
Epoch 3/30
230/230 [==============================] - 29s 126ms/step - loss: 0.7290 - accuracy: 0.6783 - val_loss: 0.9499 - val_accuracy: 0.6356
Epoch 4/30
230/230 [==============================] - 29s 127ms/step - loss: 0.7000 - accuracy: 0.7114 - val_loss: 0.5584 - val_accuracy: 0.7659
Epoch 5/30
230/230 [==============================] - 29s 126ms/step - loss: 0.4265 - accuracy: 0.8513 - val_loss: 0.4920 - val_accuracy: 0.8354
Epoch 6/30
230/230 [==============================] - 29s 125ms/step - loss: 0.2448 - accuracy: 0.9191 - val_loss: 0.3404 - val_accuracy: 0.8914
Epoch 7/30
230/230 [==============================] - 29s 127ms/step - loss: 0.1919 - accuracy: 0.9354 - val_loss: 1.1314 - val_accuracy: 0.7862
Epoch 8/30
230/230 [==============================] - 29s 126ms/step - loss: 0.1676 - accuracy: 0.9410 - val_loss: 0.3229 - val_accuracy: 0.9067
Epoch 9/30
230/230 [==============================] - 29s 126ms/step - loss: 0.1585 - accuracy: 0.9437 - val_loss: 0.4327 - val_accuracy: 0.8938
Epoch 10/30
230/230 [==============================] - 29s 126ms/step - loss: 0.1697 - accuracy: 0.9455 - val_loss: 0.3099 - val_accuracy: 0.9036
Epoch 11/30
230/230 [==============================] - 29s 127ms/step - loss: 0.1355 - accuracy: 0.9495 - val_loss: 0.4072 - val_accuracy: 0.8955
Epoch 12/30
230/230 [==============================] - 29s 127ms/step - loss: 0.1358 - accuracy: 0.9484 - val_loss: 0.4100 - val_accuracy: 0.9023
Epoch 13/30
230/230 [==============================] - 29s 126ms/step - loss: 0.1504 - accuracy: 0.9483 - val_loss: 0.4309 - val_accuracy: 0.9026
Epoch 14/30
230/230 [==============================] - 29s 126ms/step - loss: 0.1349 - accuracy: 0.9484 - val_loss: 0.4948 - val_accuracy: 0.9002
Epoch 15/30
230/230 [==============================] - 29s 126ms/step - loss: 0.1277 - accuracy: 0.9539 - val_loss: 0.4652 - val_accuracy: 0.9013
Epoch 16/30
230/230 [==============================] - 29s 127ms/step - loss: 0.1405 - accuracy: 0.9504 - val_loss: 0.3800 - val_accuracy: 0.9135
Epoch 17/30
230/230 [==============================] - 29s 126ms/step - loss: 0.1289 - accuracy: 0.9512 - val_loss: 0.4618 - val_accuracy: 0.9006
```

```
result=confusion_matrix(Y_test, model1.predict(X_test))
sn.heatmap(result, annot=True)
```

```
<AxesSubplot:xlabel='Pred', ylabel='True'>
```




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

```
93/93 [==============================] - 3s 28ms/step - loss: 0.4102 - accuracy: 0.9182
```

```
score1
```

```
[0.41017231345176697, 0.9182218909263611]
```

Final Comments

- By Simple two layered LSTM, we got a good accuracy of 91.82%.
- In short, Deep Learning help us to built models even when we don't have domain expert engineered features.
- LSTM model can be further improved by running it for more epochs and more evaluations while tuning hyper-parameter.

Machine learning model(Logistic Regression)

```
print('X_train and y_train : ({},{})'.format(X_train.shape, y_train.shape))
print('X_test  and y_test  : ({},{})'.format(X_test.shape, y_test.shape))
y_test.sample()
y_test
```

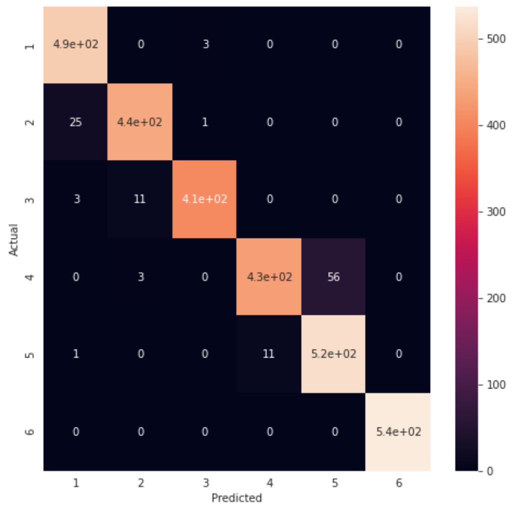
```
X_train and y_train : ((7352, 128, 9),(7352,))
X_test  and y_test  : ((2947, 128, 9),(2947,))
0       5
1       5
2       5
3       5
4       5
       ..
2942    2
2943    2
2944    2
2945    2
2946    2
Name: Activity, Length: 2947, dtype: int64
```

```
logistic_regression= LogisticRegression(solver='lbfgs',max_iter=100000)
result=logistic_regression.fit(X_train,y_train)
y_pred=logistic_regression.predict(X_test)
```

```
plt.figure(figsize=(8,8))
plt.grid(b=False)
confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
```

```
plt.figure(figsize=(8,8))
plt.grid(b=False)
confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
```

```
<AxesSubplot:xlabel='Predicted', ylabel='Actual'>
```




```
print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))
plt.show()
```

```
Accuracy:  0.9613165931455717
```

Chapter 10

Conclusion

The final accuracy for the LSTM model comes out to be almost **93%!** And it can peak to values such as **94%**, at some moments during the training, depending on random initialisation of network's weights.
Machine learning algorithm too has an outstanding accuracy of **96%** but it uses the expert engineered 561 features to build its model while LSTM uses the raw features of data signals because it works well on time series data. This is one of the most important advantage of LSTM deep learning algorirthm over other machine learning algorithms.

Final Comments:

Model	**Accuracy**
Machine learning (Logistic Regression)	96.13%
LSTM Model (Recurrent Neural Network)	92.70%

References

[1] https://towardsdatascience.com/time-series-classification-for-human-acti

[2] https://machinelearningmastery.com/deep-learning-models-for-human-activ

[3] https://machinelearningmastery.com/how-to-develop-rnn-models-for-human-