Problem Statement :

Activity recognition (Walking and Running) using accelerometer data value from a smartphone.

Device Information :

The accelerometer in Android phones measures the acceleration of the device on the x (lateral), y (longitudinal), and z (vertical) axes. Accelerometers can be used to detect movement and the rate of change of the speed of movement. The data received from the accelerometer was in the form of a three-valued vector of floating point numbers that represented the individual accelerations of the smartphone device in the X, Y, and Z axes subtracted by the gravity vector G.

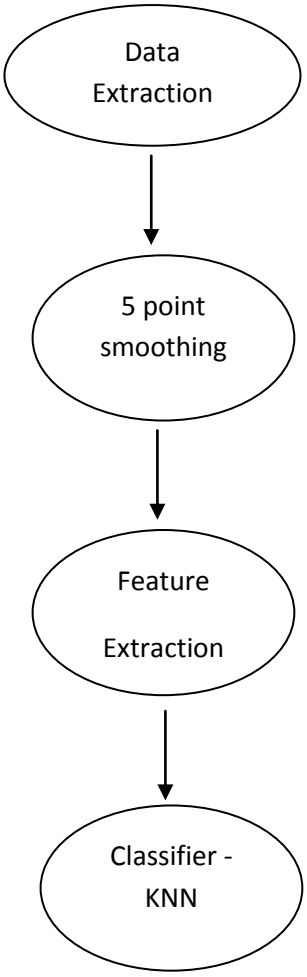
Previous research works-

SNO	Research Paper	Activity Recognised	Pre-Processing	Window Size	Features Used	Classifier	Results(approx)
1	Detecting User Activities using the Accelerometer on Android Smartphones[1]	Phone detached, walking , Running, jumping	1 – Combining acceleration vector into a single magnitude 2- Noise reduction using linearization and smoothing	Window of 512 for feature extraction	1) Fundamental frequency 2) average acceleration 3)Maximum Amplitude 4) Minimum amplitude	Nearest neighbor Bayes naive	93%
2	Activity Recognition from Accelerometer Data[3]	Standing, Walking, Running,Climbing up stairs,	No noise filter	Feature size of 256 with 128 samples overlapping	1) Mean 2)Standard Deviation 3) Energy 4)	Decision Tables Decision Trees (C4.5),	99%

		Climbing down stairs, Sit-ups, Vacuuming, Brushing teeth.			Correlation.	K-nearest neighbors, SVM, Naive Bayes.	
3	Recognizing Human Activities Userindependently on Smartphones Based on Accelerometer Data[4]	walking, running, cycling, driving a car and sitting/standing	Combining acceleration vector into a single magnitude	The windows were of the length of 300 observations, which is 7.5 seconds,	Standard deviation, mean, minimum, maximum, five different percentiles (10, 25, 50, 75, and 90), and a sum and square sum of observations above/below certain percentile (5, 10, 25, 75, 90, and 95)	knn (k nearest neighbors) QDA (quadratic discriminant analysis	97%
4	Machine Learning Methods for Classifying Human Physical Activity from On-Body Accelerometers[6]	Sitting Walking lying stair climbing standing Running cycling	No noise filter	Feature vecotr 50%-overlapping sliding windows with 512 samples	The DC component, the energy, the frequency-domain entropy, and the correlation coefficients	Hidden Markov Models (HMMs)	95-98%
5	Human Activity Recognition on Smartphones using a Multiclass Hardware-Friendly Support Vector Machine[5]	(standing, walking, laying, walking, walking upstairs and walking downstairs	Noise Filter	Features-sampled in _xed-width sliding windows of 2.56 sec and 50% overlap	mean, standard deviation, signal magnitude area, entropy, signal-pair cor-relation, FFT for frequency	SVM	,89%
6	Activity Recognition using Cell Phone	walking, jogging, climbing	No noise filter	Feature - 10-second segments and	Average. Standard Deviation.	Logistic Regression, Multilayer	93%

	Accelerometers [7]	stairs, sitting, and standing		then generated features that were based on the 200 readings contained within each 10-second segment.	Average Absolute Difference, Average Resultant Acceleration, Time Between Peaks, Binned Distribution[	Perceptron, Straw Man	

Method Flow :



Technology used :

Python and its API – Numpy , Scipy ,Matplotlib was used for all of the above processes.

Data Extraction :

Accelerometer reading was recorded using Physics toolbox accelerometer (https://play.google.com/store/apps/details?id=com.chrystianvieyra.android.physicstoolboxaccelerometer&hl=en)

The activities – Walking and Running – were performed by 5 People(4 friends + me) with the smartphone **kept in their hand.** All the persons were male , age – 23-25

The data from 4 people were used as the training data and the 5th one was used for testing.

The data contained the X,Y,Z acceleration with the time and the data was returned in a .csv file –

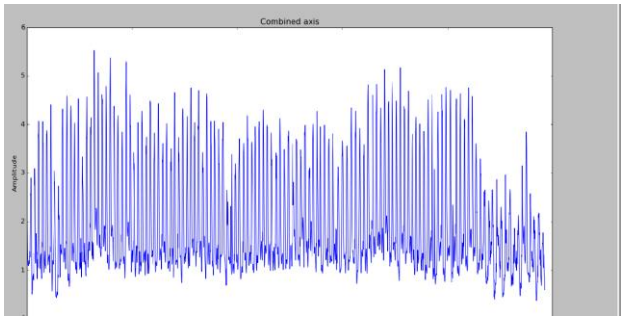
	A	B	C	D	E
1	time	x	y	z	
2	0.003	0.38	-0.19	0.58	
3	0.004	0.44	-0.22	0.53	
4	0.014	0.47	-0.26	0.45	
5	0.015	0.5	-0.29	0.36	
6	0.019	0.5	-0.35	0.27	
7	0.024	0.46	-0.42	0.19	
8	0.029	0.4	-0.5	0.13	
9	0.034	0.34	-0.57	0.07	
10	0.039	0.28	-0.62	0	
11	0.048	0.24	-0.66	-0.07	
12	0.049	0.21	-0.67	-0.14	
13	0.054	0.22	-0.68	-0.22	
14	0.062	0.24	-0.68	-0.31	
15	0.064	0.32	-0.65	-0.43	
16	0.069	0.43	-0.58	-0.51	
17	0.074	0.5	-0.53	-0.57	

Pre-Processing :

1) The first step taken was to merge the three dimensional input signal into one acceleration magnitude. We found the magnitude of the acceleration vector by taking the Euclidean magnitude of the three individual values, that is:

   $m = \text{sqrt}(x^2 + y^2 + z^2)$

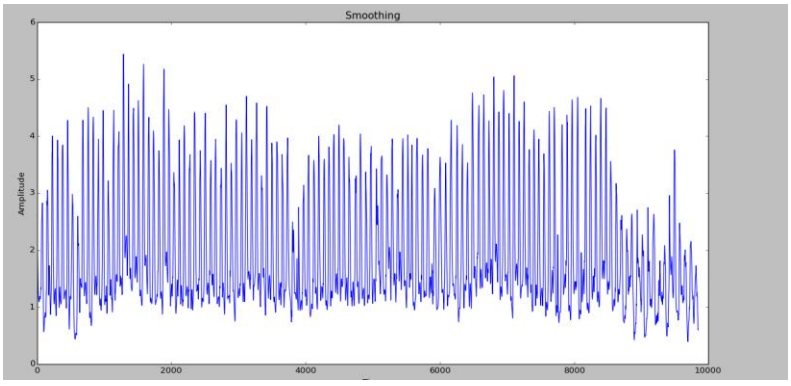
The merger was done to simplify feature extraction as mentioned in [1] as it was sufficient to determine the features from the same. Directional features are important for activities where direction information is required such as martial arts.



Code :

```
# Reading the data from the file
time, x, y, z = np.loadtxt('E:/IIIT delhi/Activity/sushant running.csv',delimiter =',',unpack =True)
# combining the acceleration
mpre = x*x+y*y+z*z
m = np.sqrt(mpre)
#ploting the data
plt.plot(m)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('Combined axis')
plt.show()
```

2) Noise Filter - The input signal was passed through a 5-point smoothing signal to reduce the noise. This additional noise could have come from any of a number of sources (e.g., idle orientation shifts, screentaps, bumps on the road while walking). The 5-point smoothing algorithm calculates each point to be the average of its four nearest neighbors, the two nearest before and the two nearest after.



Code :

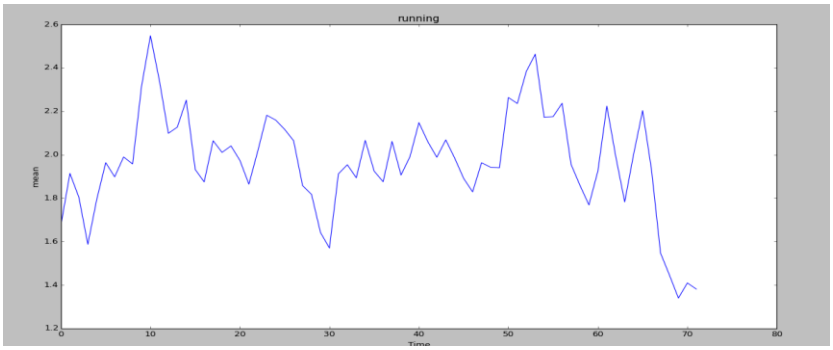
```
#initializing the variable
m_smooth = np.zeros(len(m))
m_smooth = m;

# performing the 5 point smoothin
for i ,val in enumerate(m_smooth[2 : (len(m_smooth) -2)]):
   m_smooth[i] = (m_smooth[i-2] + m_smooth[i-1] +m_smooth[i] + m_smooth[i+1] + m_smooth[i+2])/5
   m = m_smooth

#plotting the data
plt.plot(m)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('Smoothing')
plt.show()
```

Features Extraction –

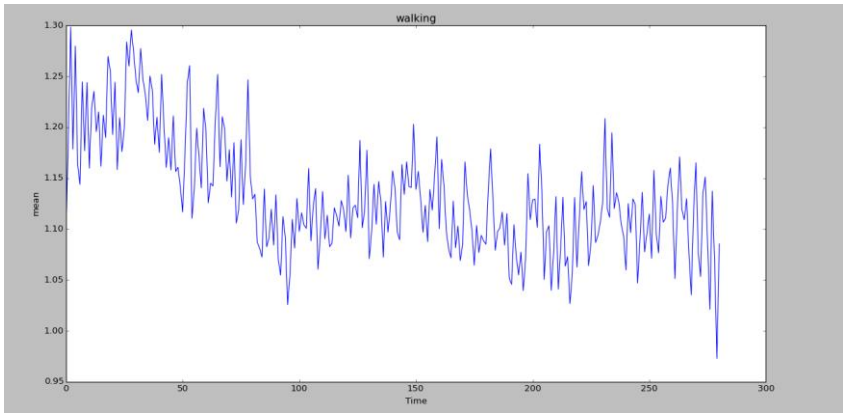
Six features were extracted based on the study of the previous works using a sample window of 256 samples with 50 % overlap .The magnitude of the below graphs shows the difference between the features values when a person is running and walking. The difference is quite significant for detecting them separately.

1) Mean -

Running

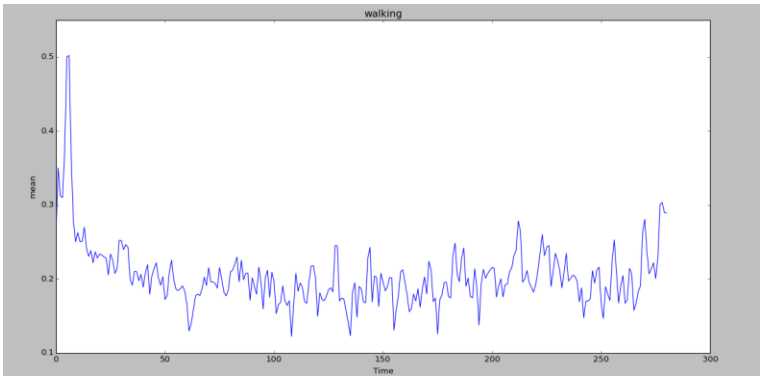


Walking

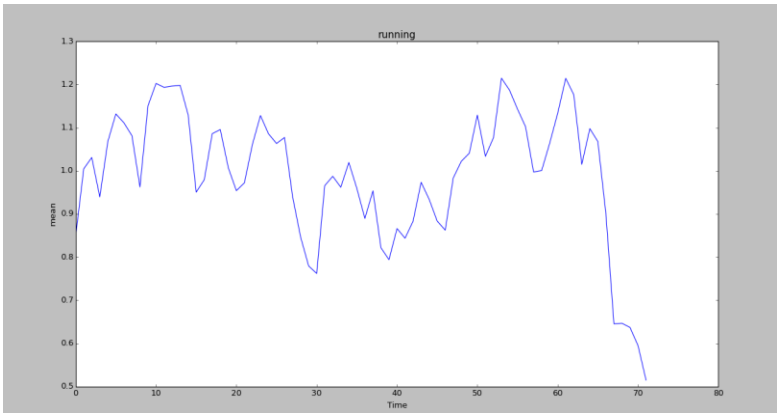


2) Standard Deviation

Walking

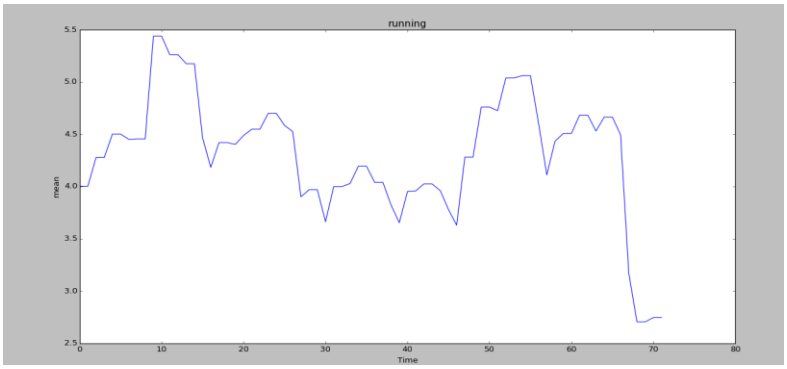


Runnning

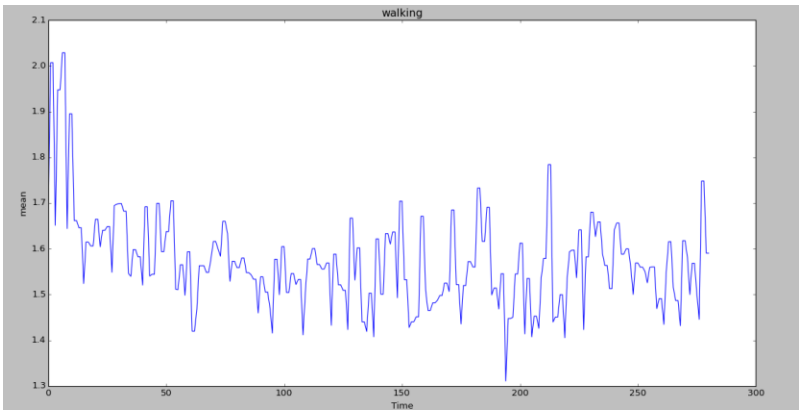


3) Maximum amplitude

Running

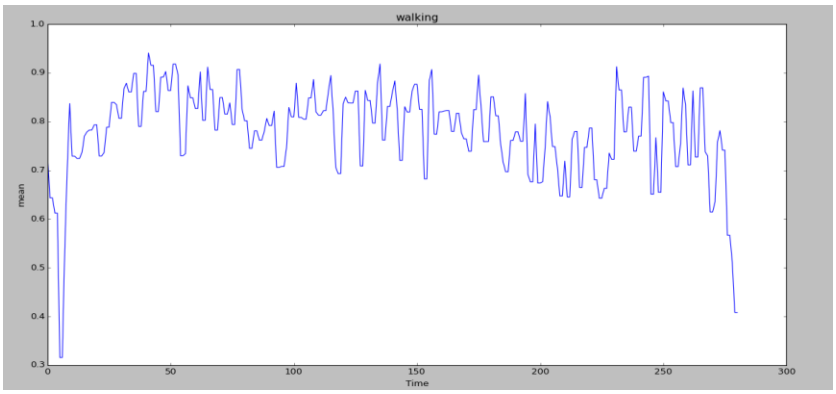


Walking

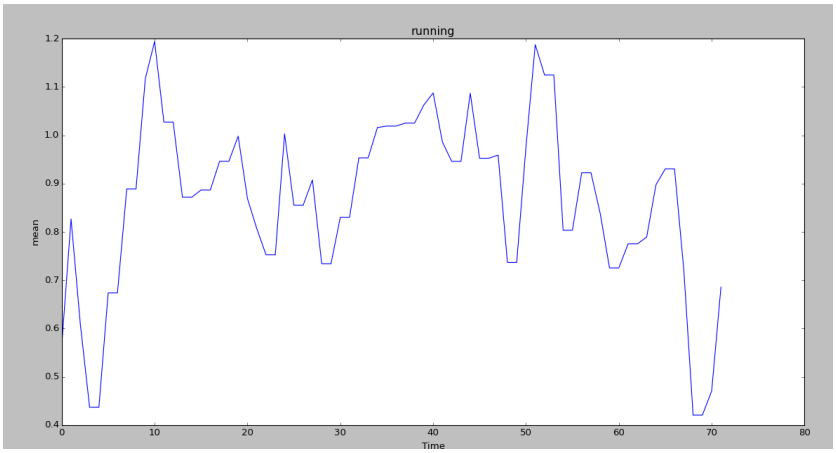


4) Minimum amplitude

Walking

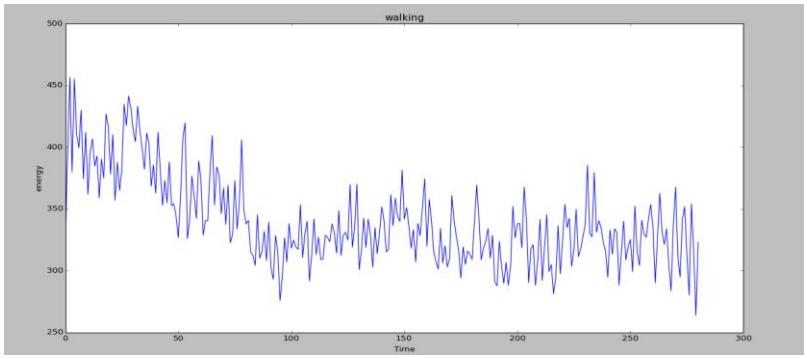


Running

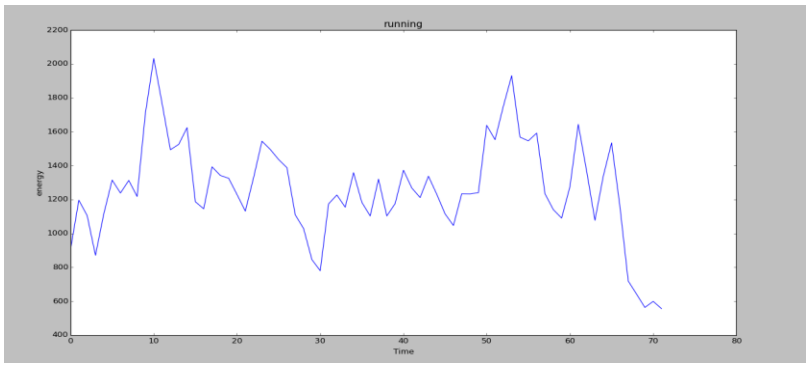


5) Energy Time domain –

For Walking

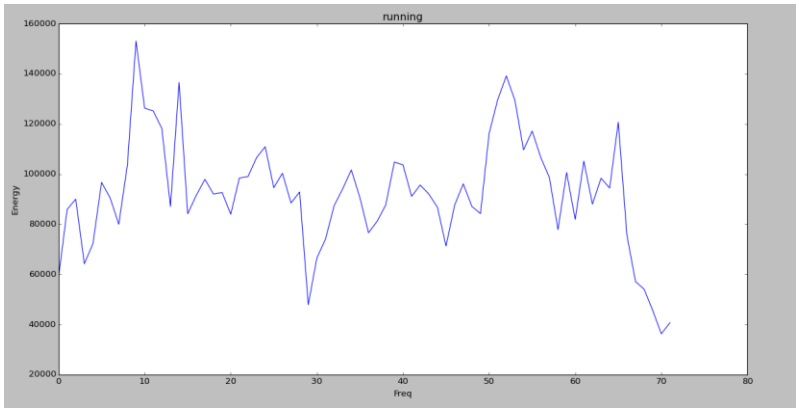


For Running

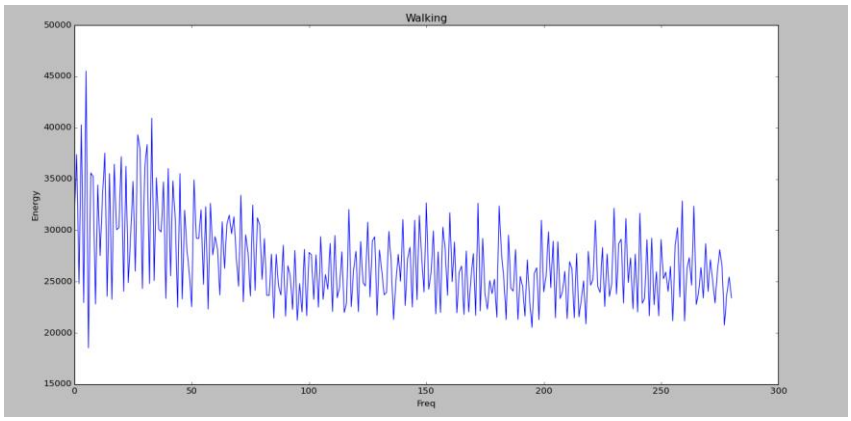


6) Energy Frequency domain

Running



Walking



There were other features as mentioned in the papers – correlation , percentiles , zero crossing , log energy .Only minimum needed features were selected in order to avoid the curse of dimensionality

Code :

```
# function for extracting the time domain features

def feature(x, fftsize=256, overlap=2):
    meanamp = []
    maxamp = []
    minamp = []
    stdamp = []
    energyamp = []
    hop = fftsize / overlap
    for i in range(0, len(x)-fftsize, hop):
        meanamp.append(np.array(np.mean(x[i:i+fftsize])))
        maxamp.append(np.array(np.max(x[i:i+fftsize])))
        minamp.append(np.array(np.min(x[i:i+fftsize])))
```

```
        stdamp.append(np.array(np.std(x[i:i+fftsize])))
        energyamp.append(np.array(np.sum(np.power(x[i:i+fftsize],2))))


    return meanamp ,maxamp ,minamp,stdamp,energyamp

#passing the signal in the function
valmean ,valmax,valmin,valstd,valenergy= feature(m)
```

Short Time Fourier Transform was applied for extracting frequency domain energy feature.

```
#function for STFT

def stft(x, fftsize=256, overlap=2):
    hop = fftsize / overlap
    w = scipy.hanning(fftsize+1)[:-1]

    return np.array([np.fft.rfft(w*x[i:i+fftsize]) for i in range(0, len(x)-fftsize, hop)])

stft_signal = stft(m)
energy_signal = []

# calculating the energy
for i, amp in enumerate(stft_signal):
    energy_signal.append(np.sum(np.power(abs(stft_signal[i]),2)))
```

All the features were copied to a separate file –

```
for i ,val in enumerate(valmean):
    saveFile = open ('features_unormal_final_test_run.csv','a')
    saveFile.write(str(valmean[i]) + ',' +str(valmax[i]) + ',' + str(valmin[i]) + ',' + str(valstd[i]) + ',' + str(valenergy[i])+','+
str(energy_signal[i]) + ',' + str(1) )
    saveFile.write('\n')
    saveFile.close()
```

1.145282	1.577376	0.666123	0.224153	348.6503	21863.19	0
1.059249	1.647668	0.713404	0.198729	297.3442	23098.85	0
1.076977	1.615515	0.627315	0.243543	312.1132	24283.55	0
1.486827	1.88757	1.132376	0.194139	575.5764	44653.59	0
1.091722	1.55023	0.802237	0.190082	314.3651	29009.82	0
1.055898	1.492317	0.652908	0.19977	295.636	26837.93	0
1.056892	1.533013	0.610314	0.252298	302.2526	22558.87	0
1.142246	1.645297	0.65486	0.219142	346.3036	23178.34	0
2.175451	5.864994	0.484133	1.402867	1715.359	133242.3	1
1.059487	1.711376	0.620621	0.268249	305.7845	27018.37	0
0.976772	2.658838	0.079854	0.780251	400.0961	22882.57	1
1.070312	1.527141	0.675023	0.200795	303.587	26372.99	0
1.054539	1.598773	0.68994	0.215839	296.6116	25573.36	0
1.033147	1.855121	0.678222	0.23647	287.5678	28751.41	0
1.268919	2.265656	0.740491	0.316093	437.7779	32496.3	0
1.086451	2.173898	0.380706	0.530583	374.2452	27031.52	1

As the scale is different for different feature, the values were normalized to [0 to 1] interval using –

X_normalise = (X – min(X(:))) / (max(X(:) – min(X(:)))

The normalized data is as follows -

	A	B	C	D	E	F	G
915	0.879213	0.729799	0.822165	0.604576	0.787205	0.722129	1
916	0.045477	0.035346	0.662095	0.047828	0.005965	0.018058	0
917	0.051554	0.035152	0.629824	0.044913	0.008408	0.018325	0
918	0.124049	0.107504	0.555018	0.117843	0.046276	0.023116	0
919	0.061259	0.084941	0.44164	0.130258	0.018432	0.032133	0
920	0.068719	0.082638	0.552694	0.104595	0.019445	0.041587	0
921	0.054497	0.063777	0.552475	0.084467	0.011802	0.020563	0
922	0.078766	0.110733	0.534058	0.115112	0.024784	0.02552	0
923	0.05451	0.060429	0.528441	0.089951	0.012183	0.03228	0
924	0.063298	0.272217	0.032514	0.358137	0.057617	0.032841	1
925	0.162663	0.113256	0.551151	0.114579	0.065556	0.085492	0
926	0.181126	0.192098	0.554079	0.169535	0.08105	0.056708	0
927	0.232293	0.395121	0.41633	0.435757	0.164477	0.143173	1
928	0.075386	0.084449	0.533763	0.098289	0.021912	0.036188	0
929	0.058803	0.053921	0.508243	0.089077	0.013966	0.035082	0
930	0.045236	0.070174	0.63705	0.059419	0.006409	0.017725	0

Data division :

The data was divided into 2 sets

1) Training data : consisting of feature vector extracted by running a sample window of size 256 with 50% overlap . This data was made from the accelerometer reading of 4 people out of the 5.
2) Testing data : consisting of feature vector extracted by running a sample window of size 256 with 50% overlap. This data was made from the accelerometer reading of the 5^{th} person.

Classifier :

K- nearest neighbor algorithm was used to classify the data in walking or running . K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If K = 1, then the case is simply assigned to the class of its nearest neighbor.

Distance – Euclidean distance : Sum($X_i - Y_i$) where Xi and Yi are the feature vectors.

Code :

```
data_path = 'C:/Python27/features_normal_final.csv'


file_data = np.loadtxt(data_path,delimiter=',')
data =[]
```

```
final_data= []
target_data = []

for i ,val in enumerate(file_data):
   data = [file_data[i][0],file_data[i][1],file_data[i][2],file_data[i][3],file_data[i][4],file_data[i][5]]
   final_data.append(data)
   target_data.append(file_data[i][6])

neigh = KNeighborsClassifier(n_neighbors=1)
neigh.fit(final_data, target_data)
```

Prediction :

After training the KNN classifier with the training data , The test data which used for analyzing the validity of the process .

Predictions were made for different values of K and the best solution was received at K = 1

For Test data contain the samples of the person running –

Number of sample – 149
Correctly predicted – 141
Incorrectly predicted – 8
Error – approx 5%

For Test data contain the samples of the person walking –

Number of sample – 279
Correctly predicted – 268
Incorrectly predicted – 11
Error – approx 4%

Full Code (Also attached in the mail) :

Feature .py - For smoothing and Feature Extraction

```
import siganalysis as sa
import numpy as np
from numpy import convolve
import matplotlib.pyplot as plt
from scipy.interpolate import spline
import math
from scipy.fftpack import *
import scipy

#time, x, y, z = np.loadtxt('E:/IIIT delhi/Activity/yatharth running.csv',delimiter =',',unpack =True)
#time, x, y, z = np.loadtxt('E:/IIIT delhi/Activity/shrey running.csv',delimiter =',',unpack =True)
```

```
#time, x, y, z = np.loadtxt('E:\IIIT delhi\Activity\surya running.csv',delimiter =',',unpack =True)
#time, x, y, z = np.loadtxt('E:/IIIT delhi/Activity/sulabh running.csv',delimiter =',',unpack =True)

#time, x, y, z = np.loadtxt('E:/IIIT delhi/Activity/yatharth walking.csv',delimiter =',',unpack =True)
#time, x, y, z = np.loadtxt('E:/IIIT delhi/Activity/shrey walking.csv',delimiter =',',unpack =True)
#time, x, y, z = np.loadtxt('E:/IIIT delhi/Activity/sulabh walking.csv',delimiter =',',unpack =True)
#time, x, y, z = np.loadtxt('E:\IIIT delhi\Activity\surya walking.csv',delimiter =',',unpack =True)

time, x, y, z = np.loadtxt('E:/IIIT delhi/Activity/sushant Walking.csv',delimiter =',',unpack =True)
#time, x, y, z = np.loadtxt('E:/IIIT delhi/Activity/sushant walking.csv',delimiter =',',unpack =True)




mpre = x*x+y*y+z*z
m = np.sqrt(mpre)
m_smooth = np.zeros(len(m))
m_smooth = m;


for i ,val in enumerate(m_smooth[2 : (len(m_smooth) -2)]):
   m_smooth[i] = (m_smooth[i-2] + m_smooth[i-1] +m_smooth[i] + m_smooth[i+1] + m_smooth[i+2])/5
   m = m_smooth

#plt.plot(m)
#plt.show()


def stft(x, fftsize=256, overlap=2):
   hop = fftsize / overlap
   w = scipy.hanning(fftsize+1)[:-1]

   return np.array([np.fft.rfft(w*x[i:i+fftsize]) for i in range(0, len(x)-fftsize, hop)])

stft_signal = stft(m)
energy_signal = []

for i, amp in enumerate(stft_signal):
   energy_signal.append(np.sum(np.power(abs(stft_signal[i]),2)))




def feature(x, fftsize=256, overlap=2):
   meanamp = []
   maxamp = []
   minamp = []
   stdamp = []
   energyamp = []
   hop = fftsize / overlap
   for i in range(0, len(x)-fftsize, hop):
      meanamp.append(np.array(np.mean(x[i:i+fftsize])))
      maxamp.append(np.array(np.max(x[i:i+fftsize])))
```

```
        minamp.append(np.array(np.min(x[i:i+fftsize])))
        stdamp.append(np.array(np.std(x[i:i+fftsize])))
        energyamp.append(np.array(np.sum(np.power(x[i:i+fftsize],2))))


    return meanamp ,maxamp ,minamp,stdamp,energyamp



valmean ,valmax,valmin,valstd,valenergy= feature(m)
diff = np.subtract(valmax,valmin)

plt.plot(energy_signal)
plt.xlabel('Freq')
plt.ylabel('Energy')
plt.title('Walking')
plt.show()



for i ,val in enumerate(valmean):
    saveFile = open ('features_unormal_final_test_run.csv','a')
    saveFile.write(str(valmean[i]) + ',' +str(valmax[i]) + ',' + str(valmin[i]) + ',' + str(valstd[i]) + ',' + str(valenergy[i])+','+
str(energy_signal[i]) + ',' + str(1) )
    saveFile.write('\n')
```

Normal_feature.py – for normalizing the feature vector

```
import siganalysis as sa
import numpy as np
from numpy import convolve
import matplotlib.pyplot as plt
from scipy.interpolate import spline
import math
from scipy.fftpack import *
import scipy



data_path = 'C:/Python27/features_unnormal_final.csv'



file_data = np.loadtxt(data_path,delimiter=',')

data =[]
final_data= []
target_data = []

valmean_nor = ((file_data[:,0]) - min(file_data[:,0]))/(max(file_data[:,0]) - min(file_data[:,0]))
```

```
valmax_nor = ((file_data[:,1]) - min(file_data[:,1]))/(max(file_data[:,1]) - min(file_data[:,1]))
valmin_nor = ((file_data[:,2]) - min(file_data[:,2]))/(max(file_data[:,2]) - min(file_data[:,2]))
valstd_nor = ((file_data[:,3]) - min(file_data[:,3]))/(max(file_data[:,3]) - min(file_data[:,3]))
valenergy_nor = ((file_data[:,4]) - min(file_data[:,4]))/(max(file_data[:,4]) - min(file_data[:,4]))
energy_signal_nor = ((file_data[:,5]) - min(file_data[:,5]))/(max(file_data[:,5]) - min(file_data[:,5]))


for i ,val in enumerate(valmean_nor):
   saveFile = open ('features_normal_final.csv','a')
   saveFile.write(str(valmean_nor[i]) + ',' +str(valmax_nor[i]) + ',' + str(valmin_nor[i]) + ',' + str(valstd_nor[i]) + ',' +
str(valenergy_nor[i])+','+ str(energy_signal_nor[i]) + ',' + str(file_data[i][6]))
   saveFile.write('\n')
   saveFile.close()
```

Knn.py - for applying the KNN algorithm and prediction

```
import sklearn
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
import siganalysis as sa
import numpy as np
from numpy import convolve
import matplotlib.pyplot as plt
from scipy.interpolate import spline
import math
from scipy.fftpack import *
import scipy

#knn = neighbors.KNeighborsClassifier()




data_path = 'C:/Python27/features_normal_final.csv'



file_data = np.loadtxt(data_path,delimiter=',')
data =[]
final_data= []
target_data = []


for i ,val in enumerate(file_data):
   data = [file_data[i][0],file_data[i][1],file_data[i][2],file_data[i][3],file_data[i][4],file_data[i][5]]
   final_data.append(data)
   target_data.append(file_data[i][6])


neigh = KNeighborsClassifier(n_neighbors=1)
neigh.fit(final_data, target_data)
```

```
#print(neigh.predict([[1.003974259,0.177428076,1.518556173,0.738015228,0.780540945,100.2639644]]))

data_path1 = 'C:/Python27/features_normal_final_test_walk.csv'



file_data1 = np.loadtxt(data_path1,delimiter=',')
data1 =[]
final_data1= []
target_data1 = []
count0 = 0
count1 = 0

for i ,val in enumerate(file_data1):
   data1 = [file_data1[i][0],file_data1[i][1],file_data1[i][2],file_data1[i][3],file_data1[i][4],file_data1[i][5]]
   predict = neigh.predict([data1])
   print predict
   if predict == file_data1[i][6]:
      count0 =count0 + 1
   else:
      count1 = count1 + 1



"""
for i in predict:

   if i == '[ 0.]':
      count0 =count0 + 1
   else:
      count1 = count1 + 1"""

print count0
print count1
```

References:

[1] https://www.truststc.org/education/reu/10/Papers/DasGreenPerezMurphy_Paper.pdf

[2]How an Accelerometer work https://www.youtube.com/watch?v=KZVgKu6v808

[3] http://compepi.cs.uiowa.edu/uploads/Wiki/ScrubScrubRevolution/ravi05.pdf

[4] http://www.ijimai.org/journal/sites/default/files/IJIMAI20121_5_5.pdf

[5] http://www.icephd.org/sites/default/files/IWAAL2012.pdf

[6] http://www.mdpi.com/1424-8220/10/2/1154

[7] http://www.cis.fordham.edu/wisdm/includes/files/sensorKDD-2010.pdf