ACTIVITY DETECTION

Y Todi

Contents

PREFACE

The project uses data provided by a paper titled “Analysis of human behavior recognition algorithms based on acceleration data” presented in IEEE International Conference of Robotics and Automation. While the public dataset contains 14 activities and 16 volunteers, our focus will be limited to three activities and 3 volunteers- 1 female and 2 males. The activities of interest would be- Climbing stairs, Descending stairs and walking.

PART 1- SPECTROGRAM

1.1 Introduction

A spectrogram is a visual representation of a signal that allows time-frequency analysis. It is typically a graph of spectral power density of a signal at different frequencies against time.

The spectral characteristic observed in a spectrogram are:

- Spectral Resolution
  Spectral resolution is the ability of a sensor or filter to resolve the energy received in a spectral bandwidth into its components, or to define fine wavelength intervals. A narrow band spectrogram focuses on changes in spectral resolution.

- Temporal Resolution
  Temporal resolution is the smallest step with which a change in frequency can be detected. A wide band spectrogram focuses on changes in temporal resolution.

A spectrogram depends upon a number of parameters such as:

- Type of windowing

  Windowing is a process of shaping a signal before its spectral analysis. There are several types of windows which include- Hamming, Hann, Kaiser, Flat top weighted window, Chebyshev window and many more.

- Frequency (f)

  This is the cyclical frequency at which the spectrogram is plotted.

- Number of frequency points per segment (n)

  By default, a spectrogram is divided into 8 segments with equal number of frequency points. The number of frequency points per segment decides the number of segments.

- Overlap

  Overlap is the amount of overlap between adjoining segments. Large overlap can result in aliasing of the signal.

- Sampling frequency

  The sampling frequency is the rate art which the samples are recorded by the sensor, i.e. the number of samples per unit time.

1.2 Windowing

Windowing reduces the discontinuities at the boundaries of each finite segment. Windowing consists of multiplying the time record by a finite-length window with an amplitude that varies smoothly toward zero at the edges. This makes the endpoints of the waveform meet and, therefore, results in a continuous waveform without sharp transitions, producing a well shaped signal.

For spectral estimate, it is important to know the frequency content of the signal. This is because the frequency characteristic of a window is a continuous spectrum with a main lobe and several side lobes. The main lobe is centered at each frequency component of the time-domain signal, and the side lobes approach zero. The height of the side lobes indicates the affect the windowing function has on frequencies around main lobes. Applying a window minimizes the effect of spectral. The main lobe is responsible for smoothing the estimated spectral shape whereas the side lobe is responsible for spectral leakage.

1.2.1 Kaiser Window

Kaiser window has two parameters:

- "n" (Number of frequency points per segment)

- β (Beta value)

Beta value provides a trade-off between side lobe level and main lobe level. Larger beta value gives lower side lobe levels but at the price of wider main lobe. Widening the main lobe generally reduces the temporal resolution when window is used for spectral analysis.

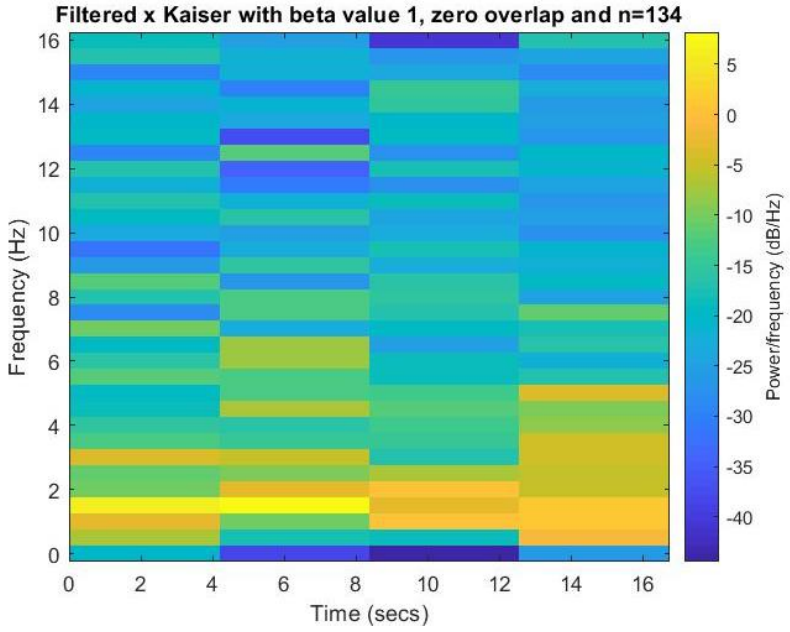
X-acceleration of the tri-axial accelerometer data is plotted. All the data used has a sampling frequency of 32Hz.

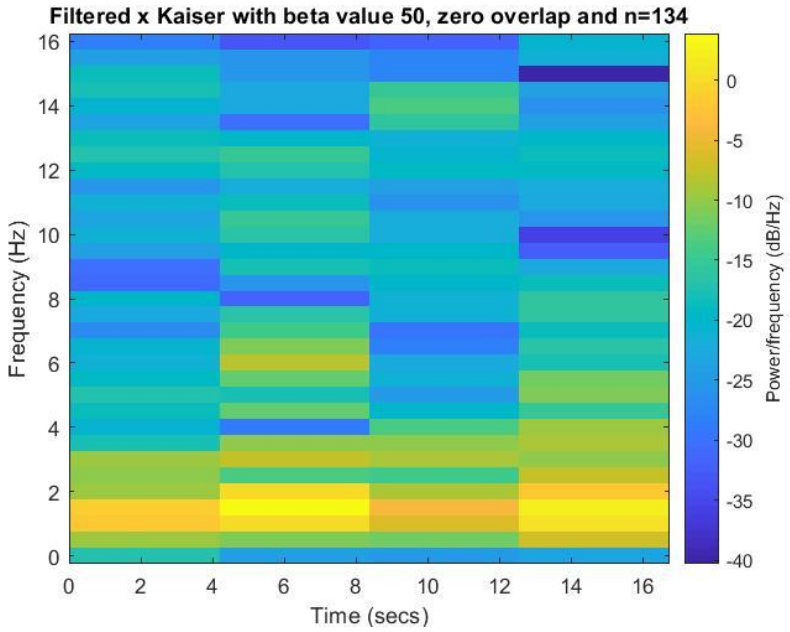
The data has been plotted in Spectrogram by varying the following while keeping other parameters constant :

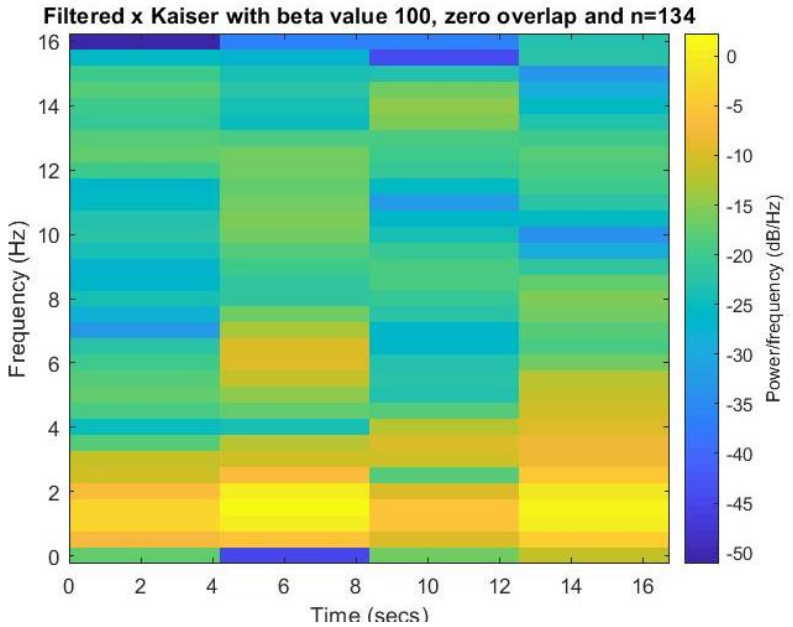
1) Beta value while all other parameters
2) Number of frequency points per segment
3) Percentage Overlap
4) Frequency

1) Spectrograph of Filtered X-acceleration with varying Beta and constant n

Parameter	Value
Number of frequency points per segment (n)	134
Beta (β)	1, 50, 100
Frequency(f)	64 Hz
Overlap	0





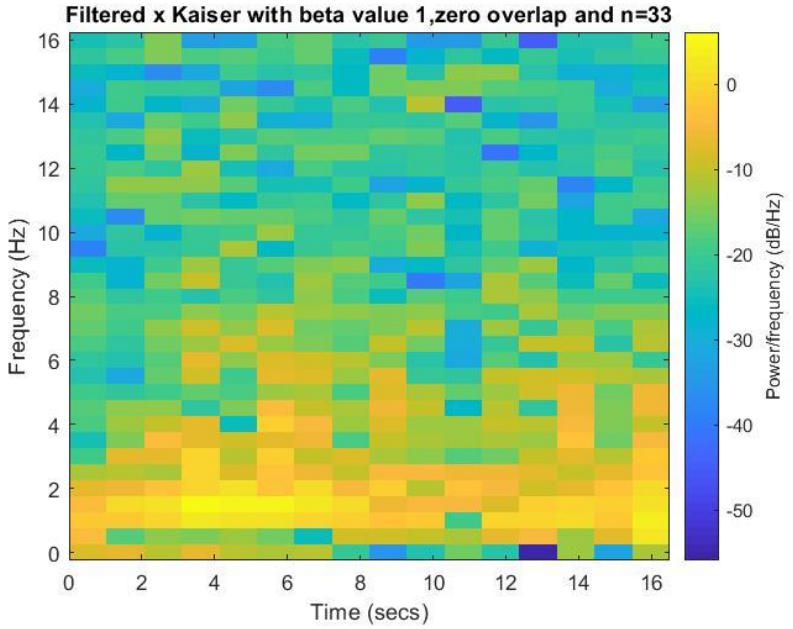


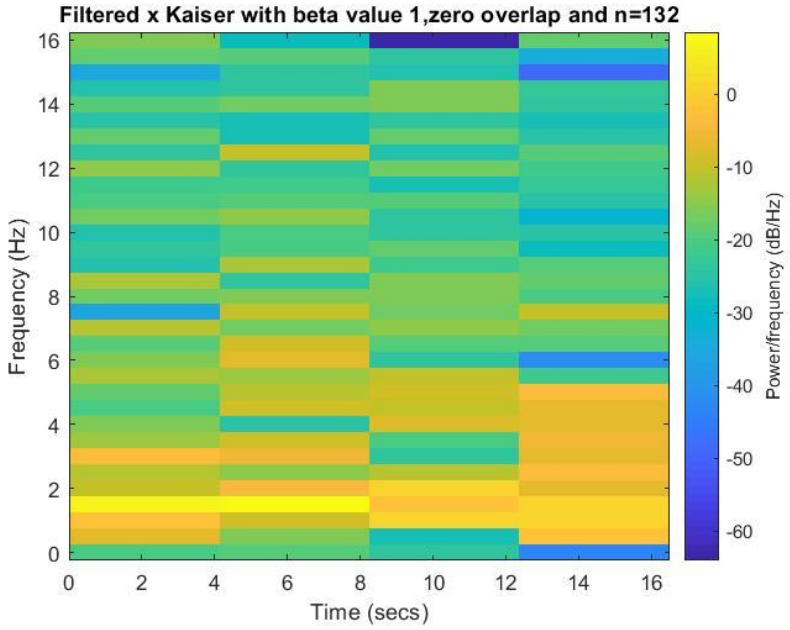
Interpretation:

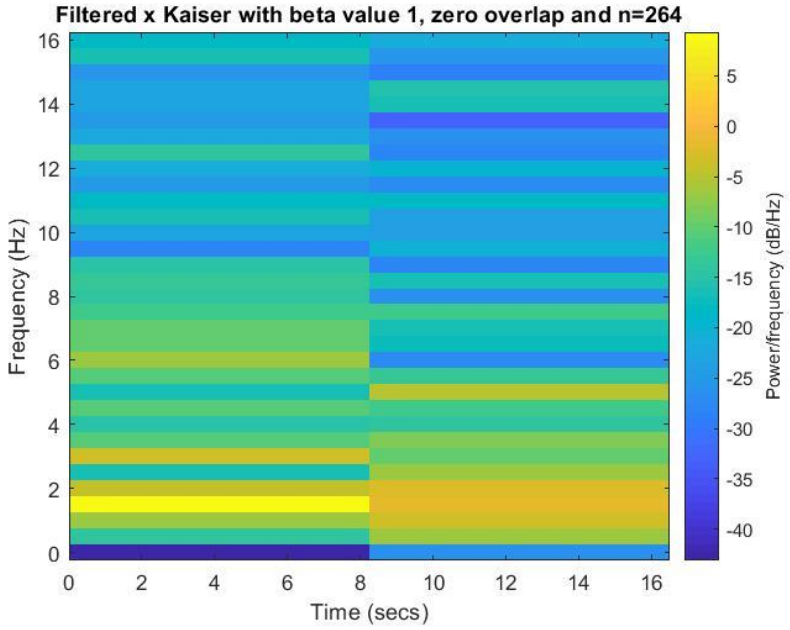
From the above Spectrographs, it can be observed that with increasing Beta value, the Frequency resolution decreases. Spectrograph with beta value=100 has much wider frequency resolution as compared to beta value of 1 where the frequency is resolved into thin bands.

2) Spectrograph of Filtered X-acceleration with varying Beta and constant n

Parameter	Value
Number of frequency points per segment (n)	33, 132, 264
Beta (β)	1
Frequency(f)	64Hz
Overlap	0





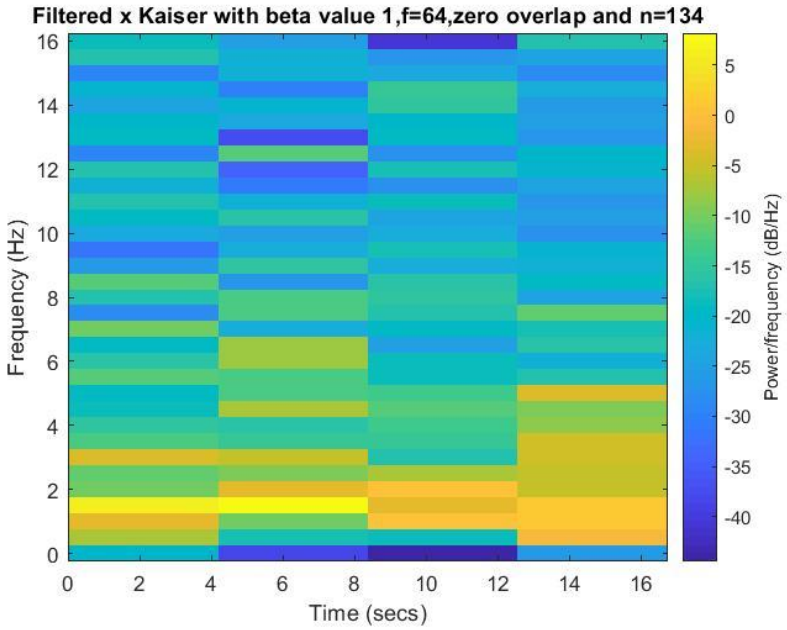


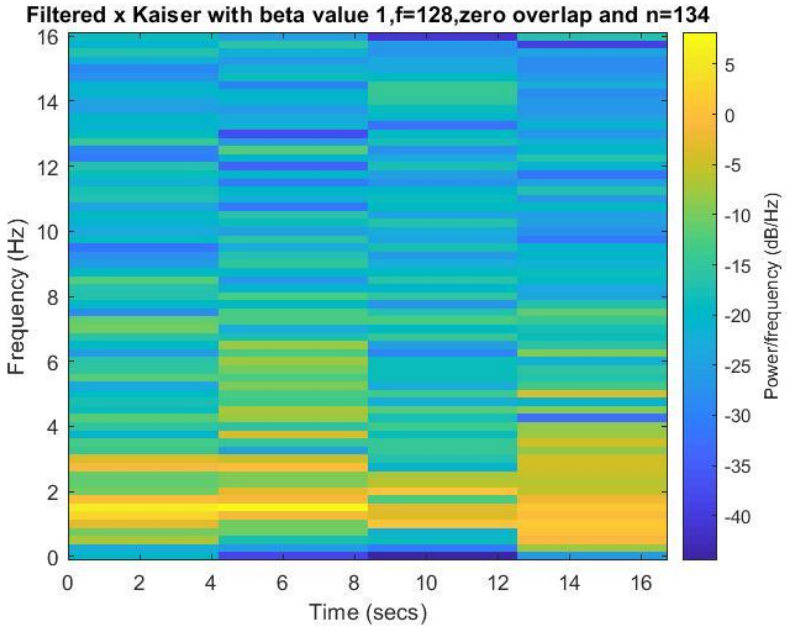
Interpretation:

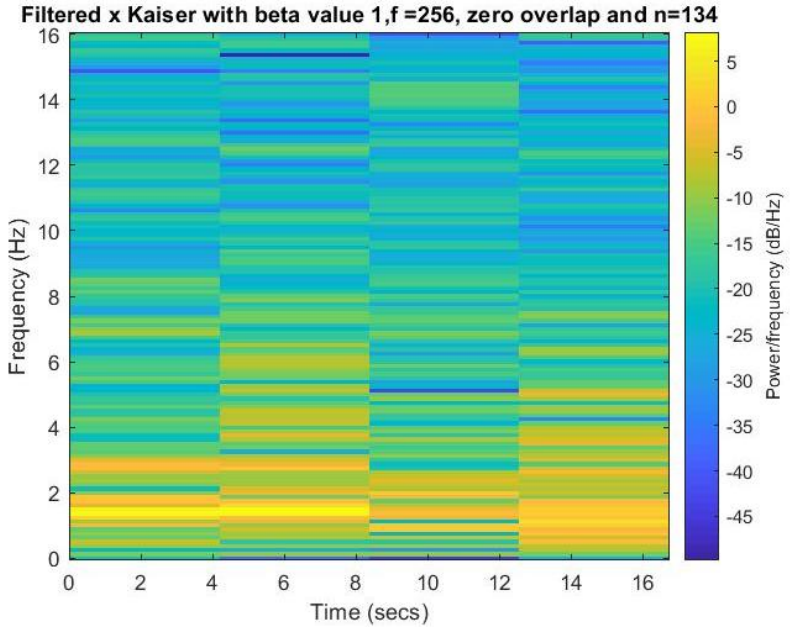
From the above Spectrographs, it can be observed that with increasing the number of frequency points per segment, i.e. by decreasing the number of segments, the spectral resolution decreases. Spectrograph with n=264 has much longer bands, encompassing many different spectral densities into one whereas the same is well resolved for n=33. The temporal resolution seems to also decrease with the increasing n.

3) Spectrograph of Filtered X-acceleration with varying Frequency(f) and constant Beta, n

Parameter	Value
Number of frequency points per segment (n)	134
Beta (β)	1
Frequency(f)	64, 128, 256
Overlap	0





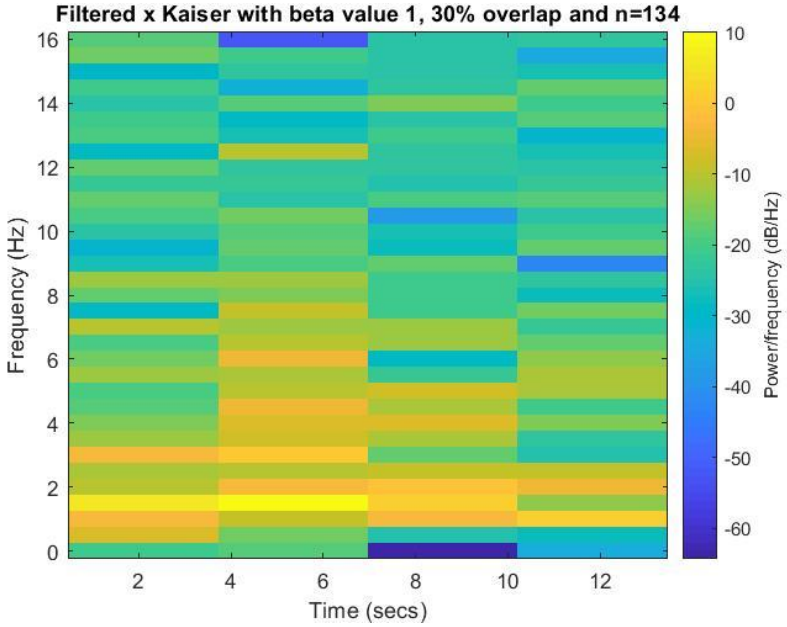


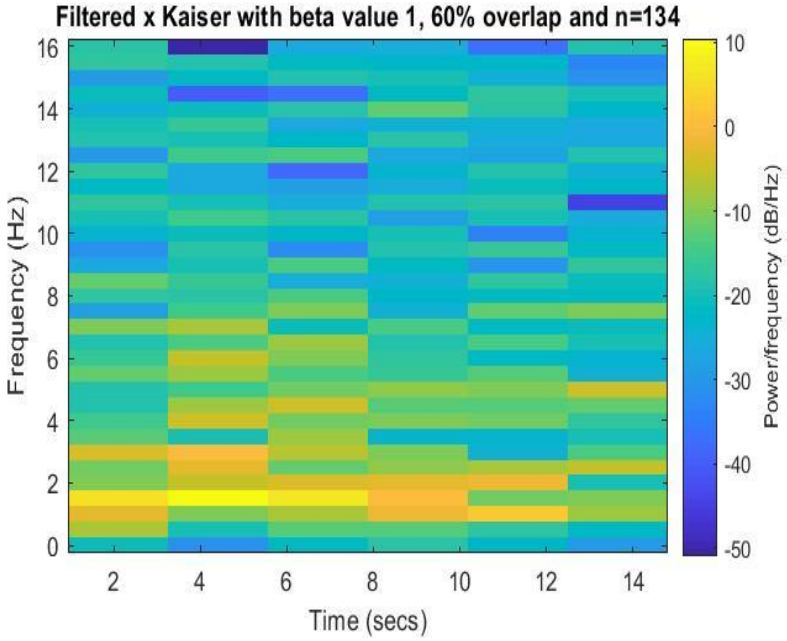
Interpretation:

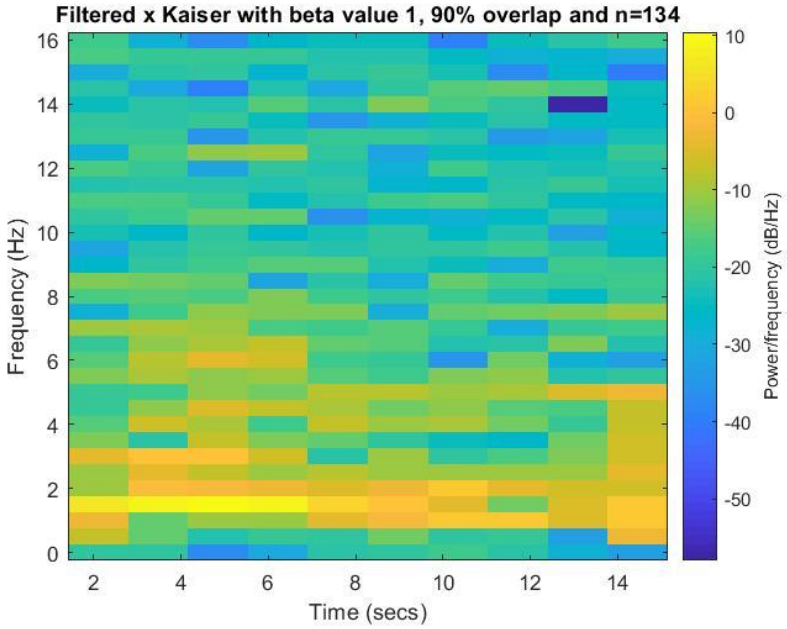
From the above Spectrographs, it can be observed that with increasing the frequency, the Temporal resolution increases, i.e. there are more number of steps of frequency into which the spectral power densities are resolved. Spectrograph with f=256 Hz has many fine bands along the vertical axis providing good temporal resolution as compared to f at 64 Hz. The spectral resolution too seems to have improved with increase in frequency.

4) Spectrograph of Filtered X-acceleration with varying Overlap and constant Beta, n, f.

Parameter	Value
Number of frequency points per segment (n)	134
Beta (β)	1
Frequency(f)	64
Overlap	30, 60, 90%







Interpretation:

From the above Spectrographs, it can be observed that with increasing the overlap, the effect is same as reducing the number of frequency points per segment. The spectral resolution increases with the increasing overlap and there is some enhancement of the temporal resolution..

1.2.2 Hamming Window

Hamming window has one parameter:

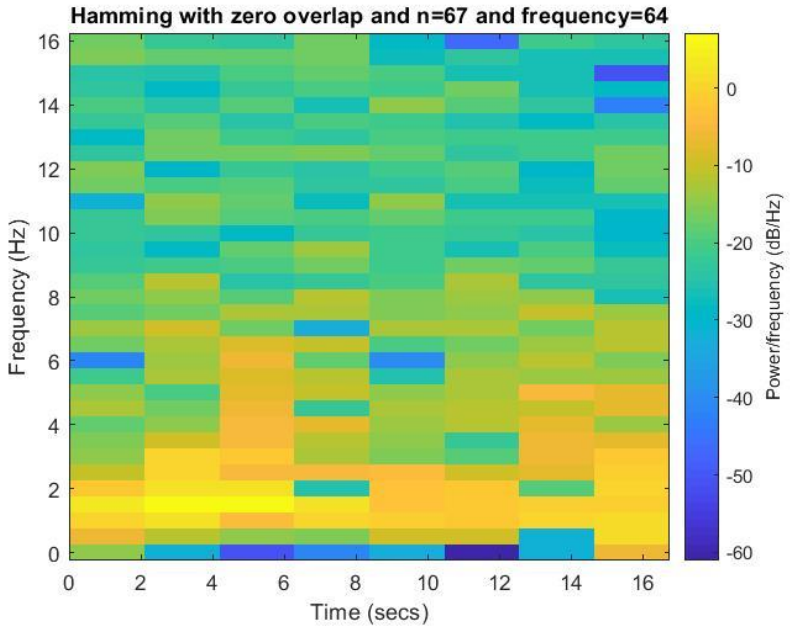
- "n" (Number of frequency points per segment)

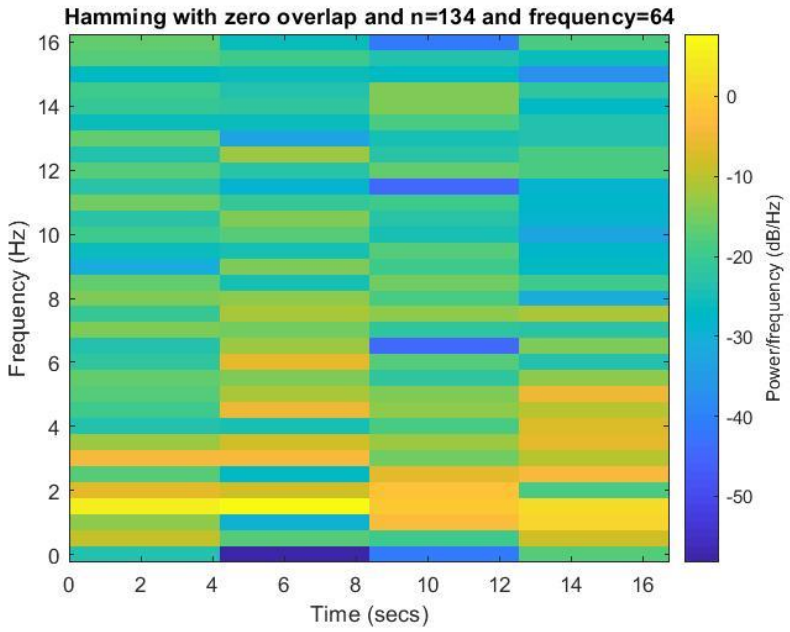
The data has been plotted in Spectrogram by varying the following while keeping other parameters constant :

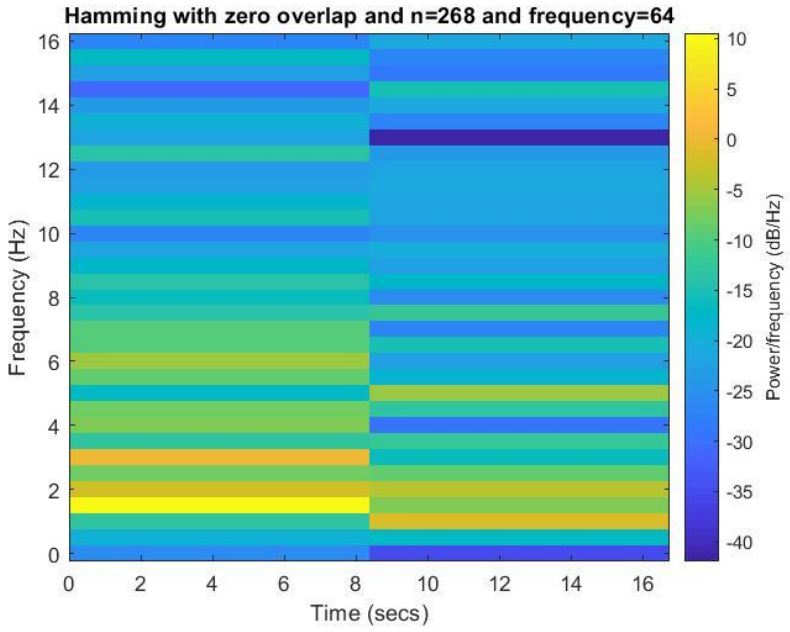
1) Number of frequency points in a segment
2) Frequency

1) <u>Spectrograph of Filtered X-acceleration with varying "n"</u>

Parameter	Value
Number of frequency points per segment (n)	67, 134,268
Frequency(f)	64
Overlap	0





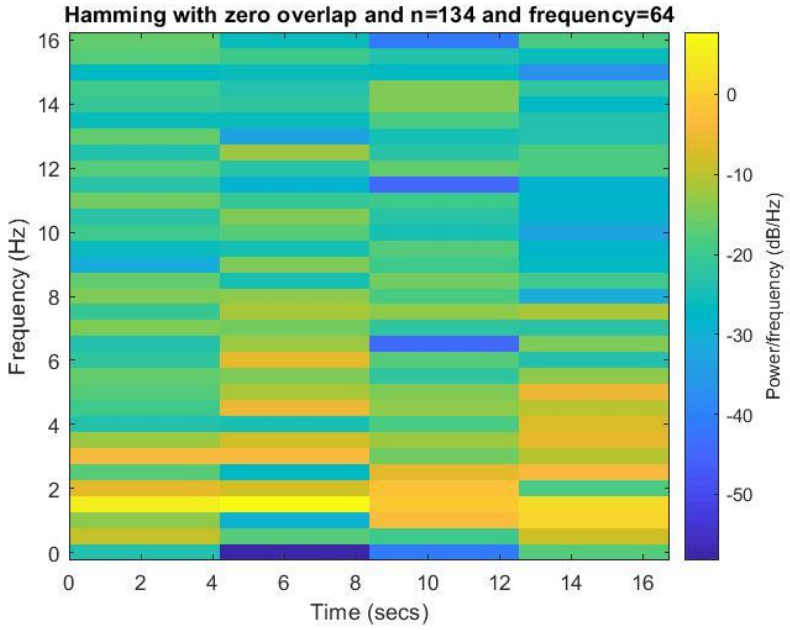


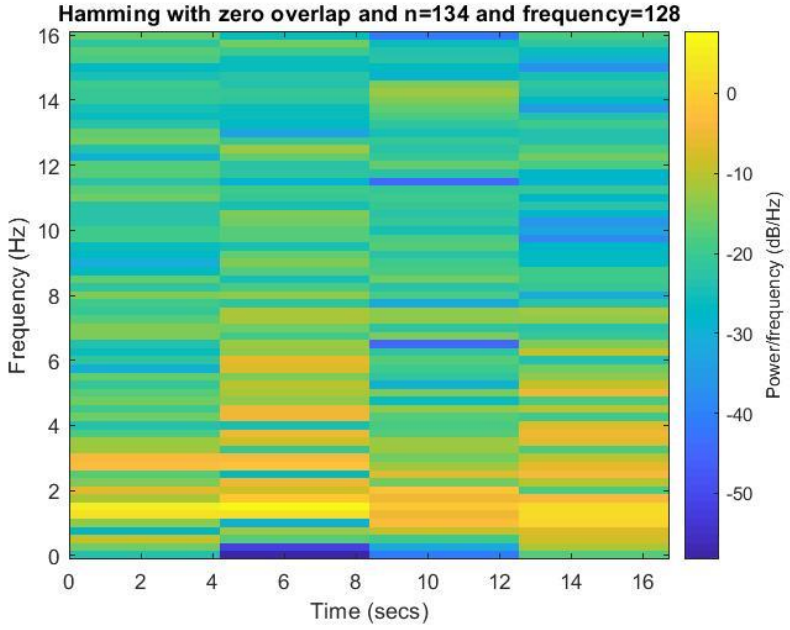
Interpretation:

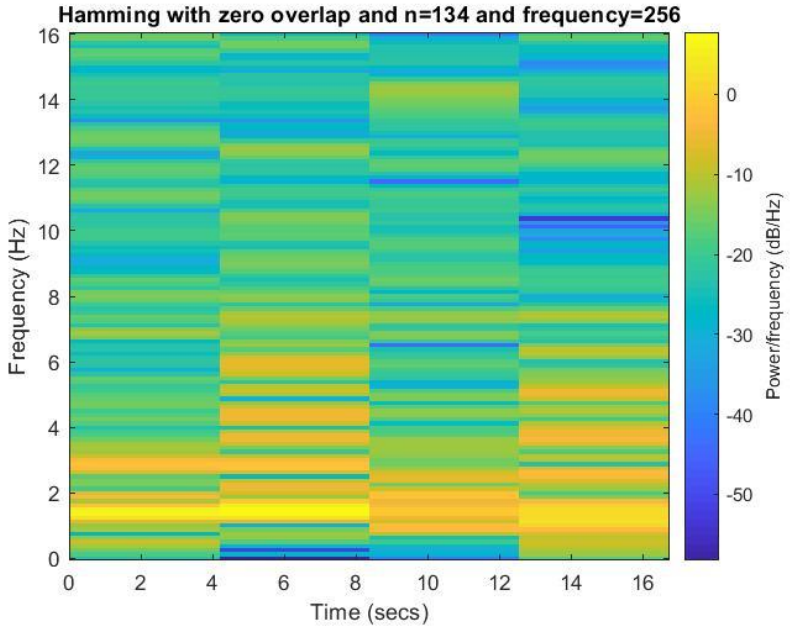
From the above Spectrographs, it can be observed that with increasing the number of frequency points per segment, i.e. by decreasing the number of segments, the spectral resolution decreases. Spectrograph with n=268 has much longer bands, encompassing many different spectral densities into one whereas the same is well resolved for n=67. The temporal resolution seems to also decrease with the increasing n.

2) Spectrograph of Filtered X-acceleration with varying frequency

Parameter	Value
Number of frequency points per segment (n)	67
Frequency(f)	64, 128, 256 Hz
Overlap	0







Interpretation:

From the above Spectrographs, it can be observed that with increasing the frequency, the Temporal resolution increases, i.e. there are more number of steps of frequency into which the spectral power densities are resolved. Spectrograph with f=256 Hz has many fine bands along the vertical axis providing good temporal resolution as compared to f at 64 Hz. The spectral resolution too seems to have improved with increase in frequency.

1.2.3 Flat Top Weighted Window

Flat top Weighted window has two parameters:

- "n" (Number of frequency points per segment)
- Flag- "periodic" or "symmetric"

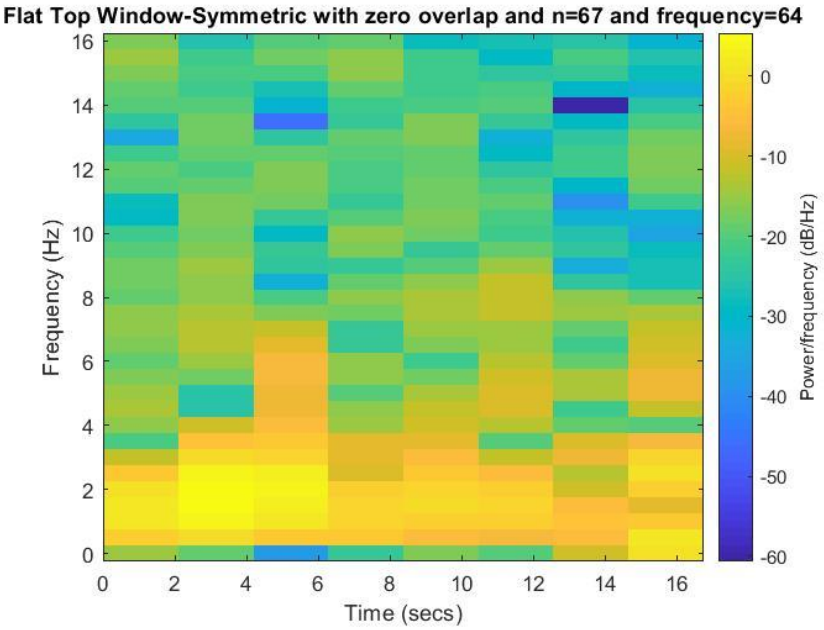
The "periodic" flag is not used for filter design. Also, the function spectrogram performs Short-time Fourier analysis which is not appropriate for this parameter. Generally, "symmetric" is used for digital filter design using windows.

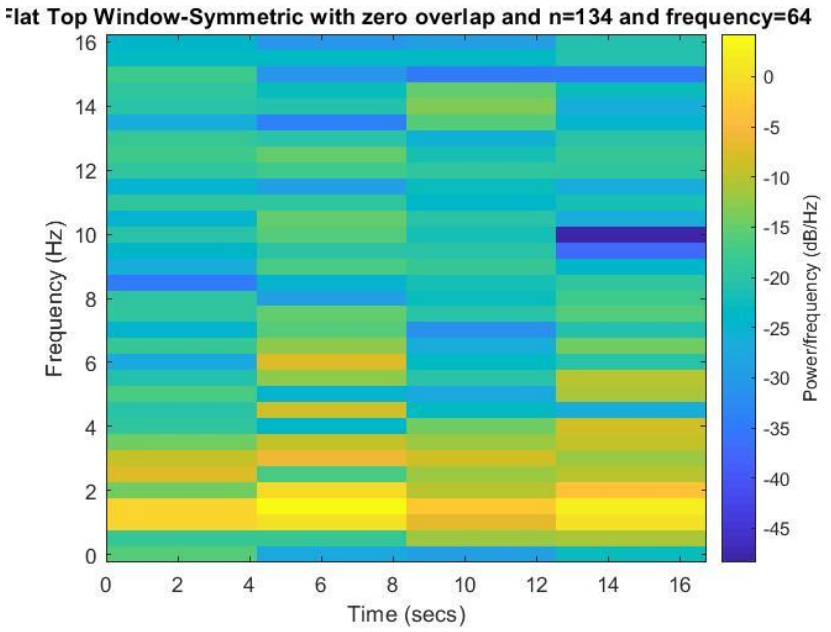
The data has been plotted in Spectrogram by varying the following while keeping other parameters constant :

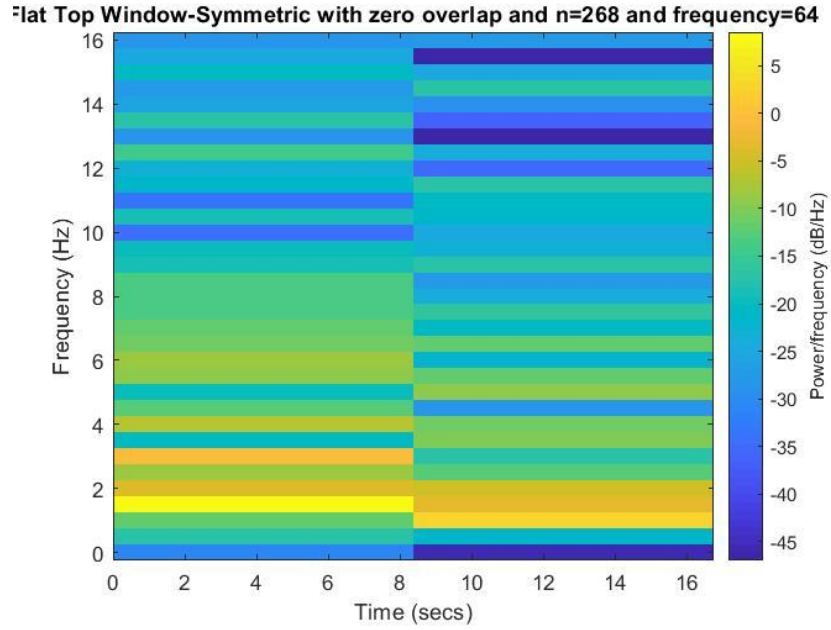
1) Number of frequency points in a segment

1) <u>Spectrograph of Filtered X-acceleration with varying "n"</u>

Parameter	Value
Number of frequency points per segment (n)	67, 134,268
Frequency(f)	64
Overlap	0







Interpretation:

From the above Spectrographs, it can be observed that with increasing the number of frequency points per segment, i.e. by decreasing the number of segments, the spectral resolution increases up to a point and then begins to decrease. Spectrograph with n=268 has much longer bands, encompassing many different spectral densities into one whereas the same is well resolved for n=67. The temporal resolution seems to also decrease with the increasing up to a point and then increases.

PART 2-FILTER

2.1 Bandpass Filter Design

A bandpass filter is combination of high-pass and lowpass filter into a single filter. Signals in the pass band of the filter are amplified by a gain whereas signals that are not in the pass band are attenuated or significantly reduced in magnitude.

Bandpass filter design parameters:

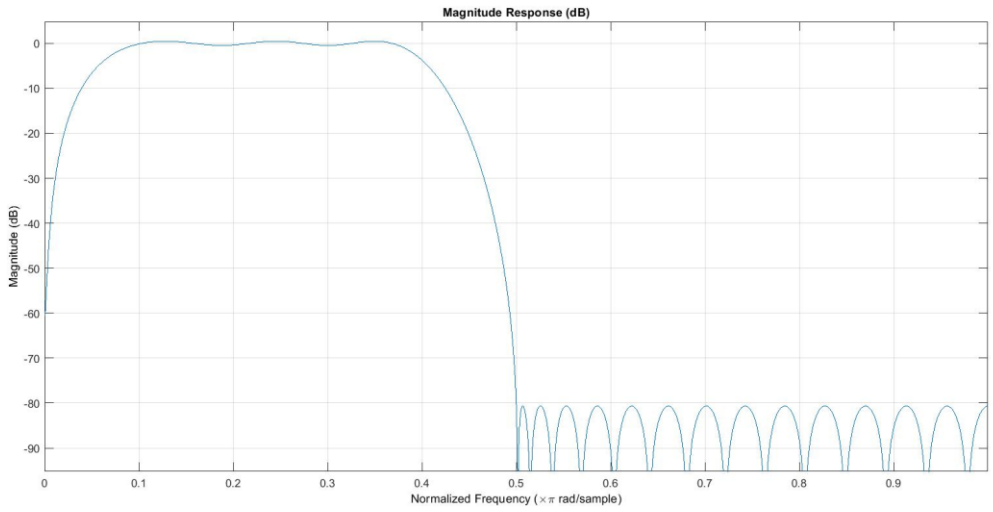
Parameter	Value
First Stop Band Frequency	0 Hz
First Pass Band Frequency	1.5 Hz
Second Pass Band Frequency	6 Hz
Second Stop Band Frequency	8 Hz
Density factor	20
Sampling Frequency	32 Hz

The filter designed is of the order 42.

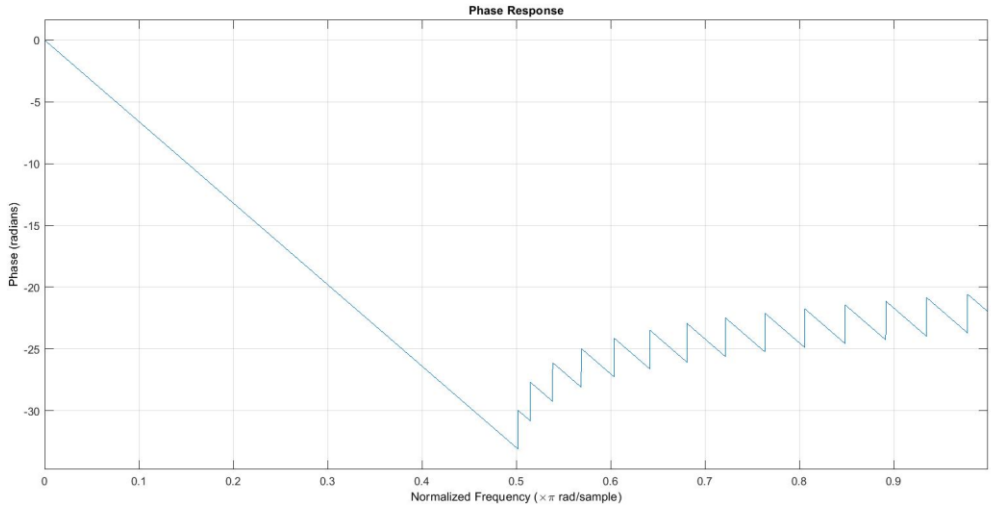
From observation, the spectrographs plotted in Part-1 show that the frequency lies on the range of 0 to 16 Hz. The average frequency of human movement while walking lies between 0.8 to 2Hz. The range of frequency of human movement is 1 to 8 Hz, 8Hz being very agile and a rare occurrence. Considering the age group of the data set and the fact that walking, climbing and descending stairs have varied frequency where climbing stairs is the slowest and descending stairs is the fastest, the range for the passband frequency chosen was 1.5 to 6 Hz. From the spectrographs plotted, it can also be observed that above 8 Hz, the power/ frequency is very low and is mostly likely, is not the data of our interest indicating human activities of walking, climbing or descending stairs.

The MATLAB code has been provided in the Appendix.

Magnitude response of the Bandpass Filter



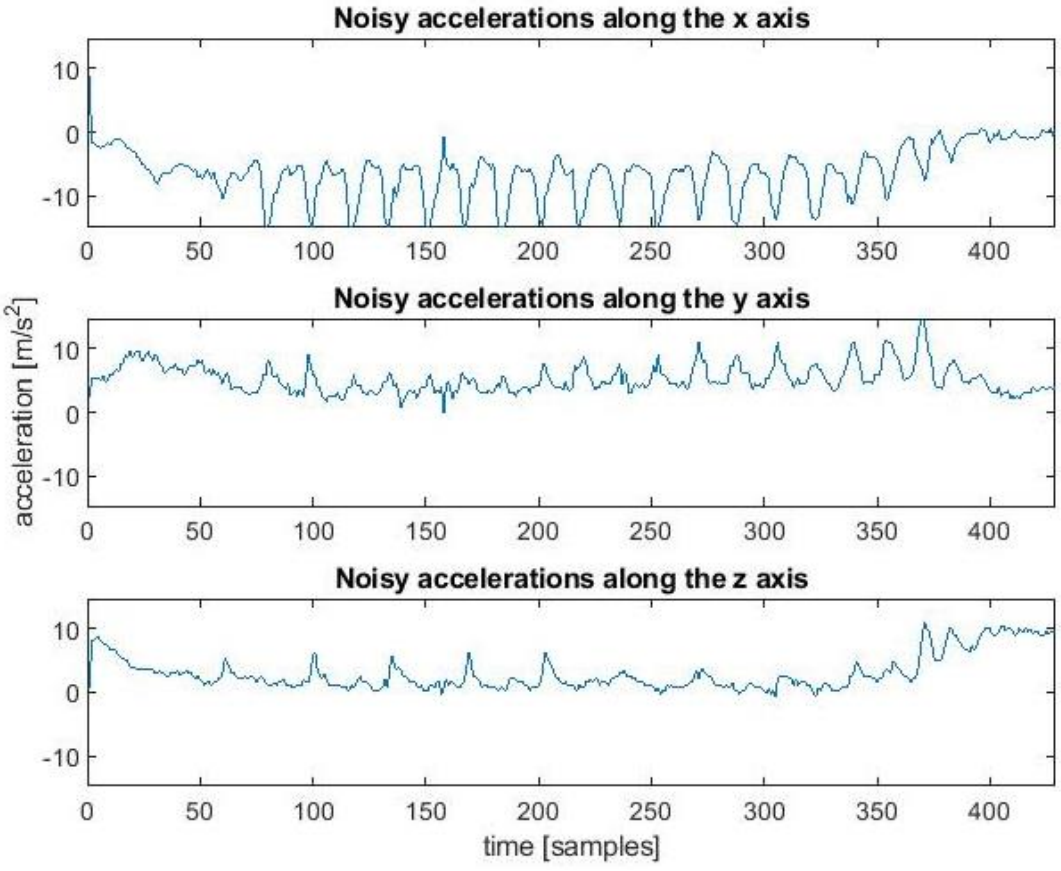
Phase response of the Bandpass Filter

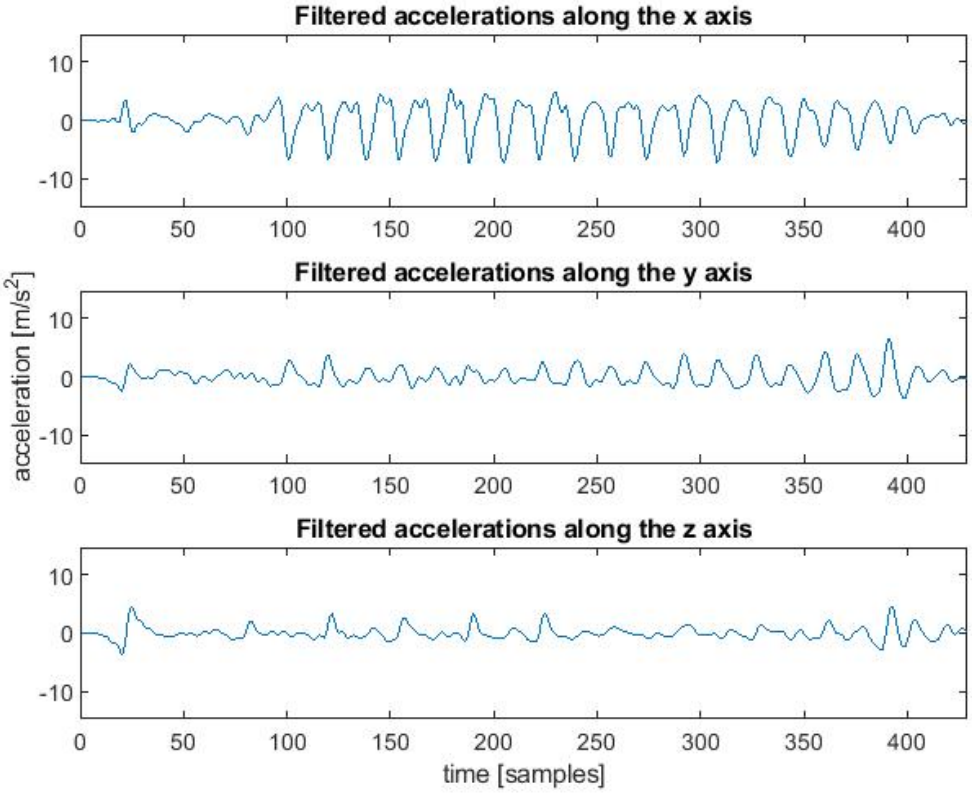


2.2 Raw and Processed Data

Example 1

The raw data and the data after filtering, i.e. processed data is plotted for a male volunteer while descending stairs.

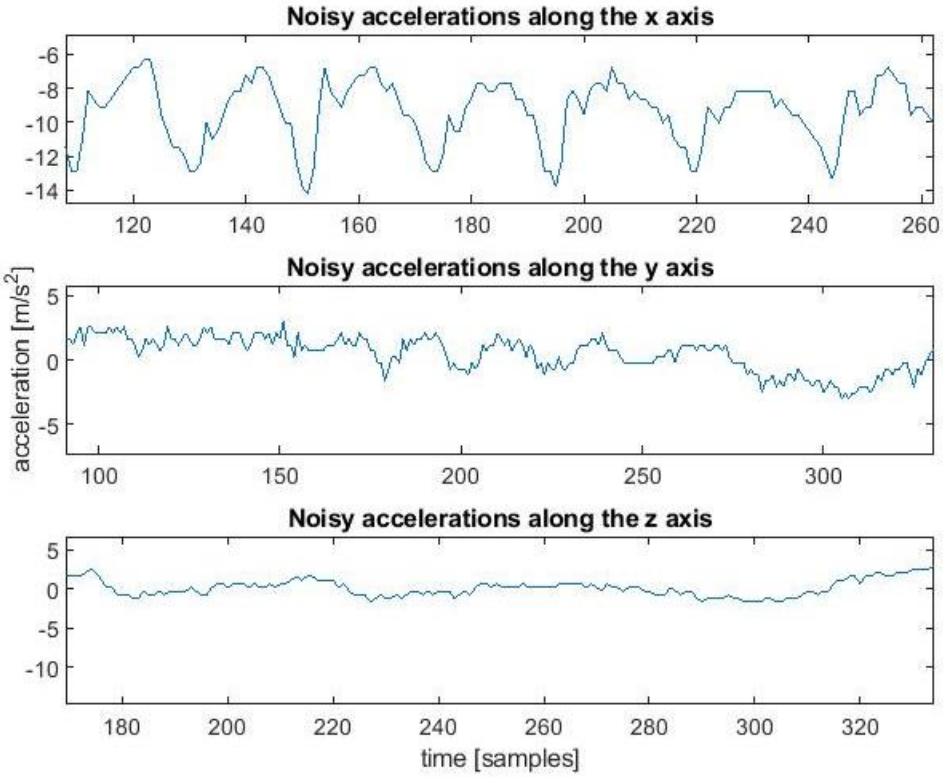


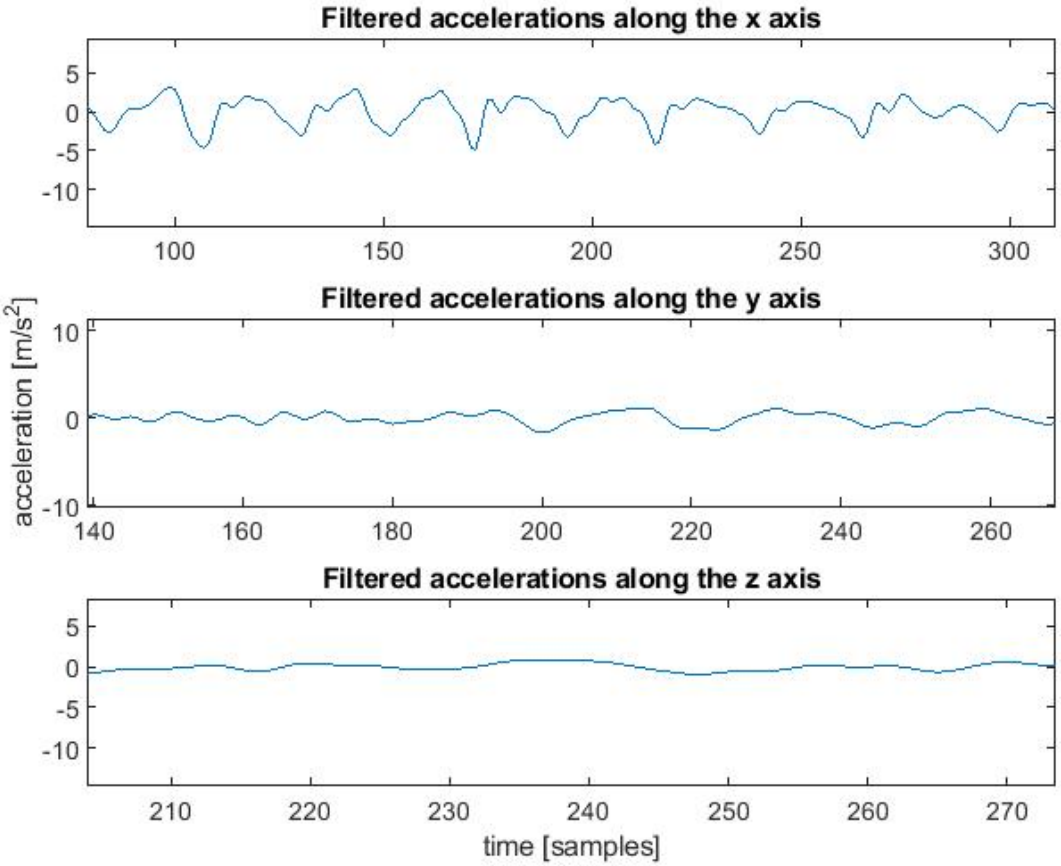


It can be observed that the noise has been reduced, the signal is far more smooth with less random spikes.

Example 2

The raw data and the data after filtering , i.e. processed data is plotted for a female volunteer while climbing stairs.




Filtered accelerations along the x axis
5
0
-5
-10
100
150
200
250
300
Filtered accelerations along the y axis
acceleration [m/s^2]
10
0
-10
140
160
180
200
220
240
260
Filtered accelerations along the z axis
5
0
-5
-10
210
220
230
240
250
260
270
time [samples]

PART 3- CLASSIFY

3.1 Structure of the Classifier

The type of classifier used is Bagged Tree, with the learning method as ensemble.

Bagging, also known as Bootstrap Aggregation is an ensemble method which has proved to be strong and powerful for machine learning. It is typically used for classification and regression problems.

An ensemble model is a technique that combines predictions from multiple machine learning algorithms together to make accurate predictions than any individual model.

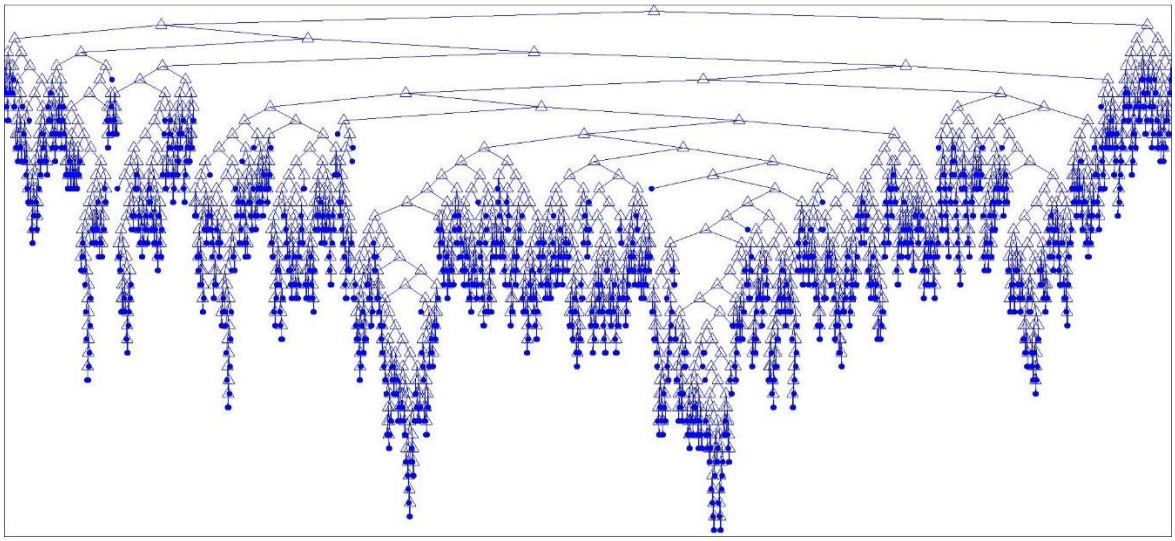
Conventional decision tree algorithm is a high variance algorithm and is data sensitive, i.e. the resulting decision tree could change with minimum changes to training data. This, the concept of bagging applied to trees become useful which allows to reduce the variance on a high variance algorithm. Bagging applied to decision trees removes the concern of overfitting data.

In a bagged tree classifier, the entire population P of size N is divided into B samples of size n.

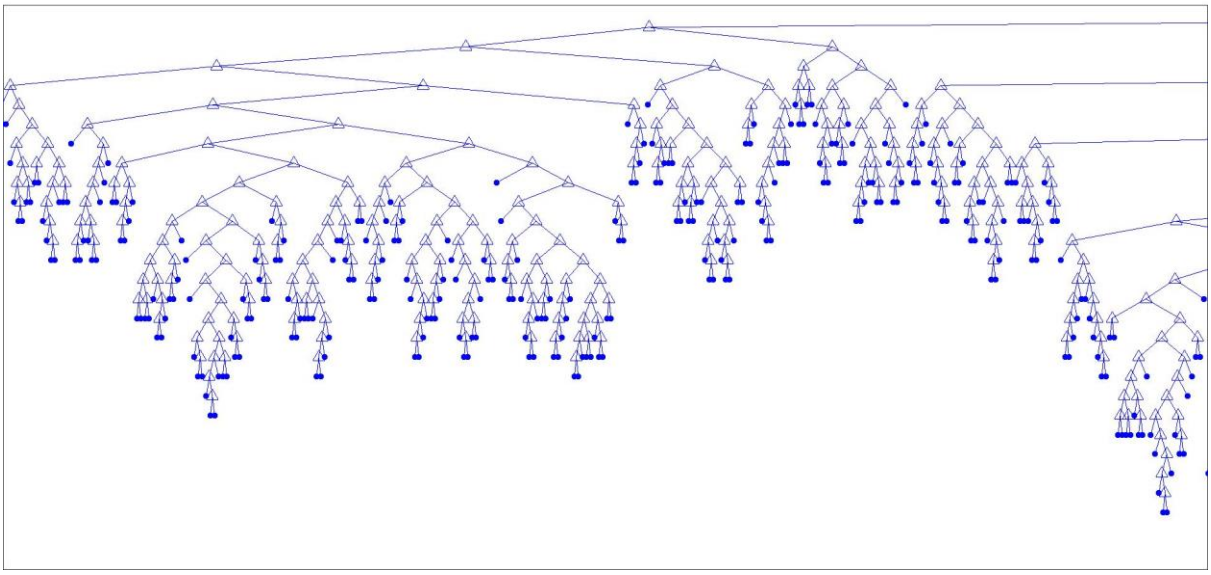
1. Create Multiple Data Sets:
   - Bagging generates B new training sets of size n by sampling from P uniformly and with replacement.
   - A construction tree is created.

2. Build Multiple Classifiers:
   - Classifiers are built on each data set.
   - The B models are fitted by assigning a class to each terminal node, and the class attached to each case is stored, couple with predictor values for each observation.

3. Combine Classifiers:
   - The predictions of all the classifiers are combined using voting for classification that leads to a robust model.



Architecture of the Classifier



Structure of part of main tree

3.2 Procedure

Procedure for training the classifier was as follows:

1) The noisy data provided was first filtered using a High-pass Filter. This was done to remove the DC component from the noisy data.
2) Filtered data was further smoothened using median filtering.
3) From the processed data, approximately 8000 observations were chosen randomly. This was done by exporting all the processed data to excel and choosing them through random selection.
4) This selected data was trained on a Classifier in MATLAB, using Structure type as Bagged Tree and learner method as Ensemble.
5) The Validation process was chosen was as Cross Fold- with 5 folds.
6) The model was trained across many structure which included Decision Trees, Support Vector machines, Ensemble classifiers and Nearest Neighbor classifier.
7) The model with highest accuracy was chosen- Bagged Tree using Ensemble.
8) The number of learners chosen for the model in Step 7 was 30.

3.3 Evaluation

3.3.1 Performance on Trained data

The table below shows that the accuracy of the classifier on trained data set was 70.4%

Current Model

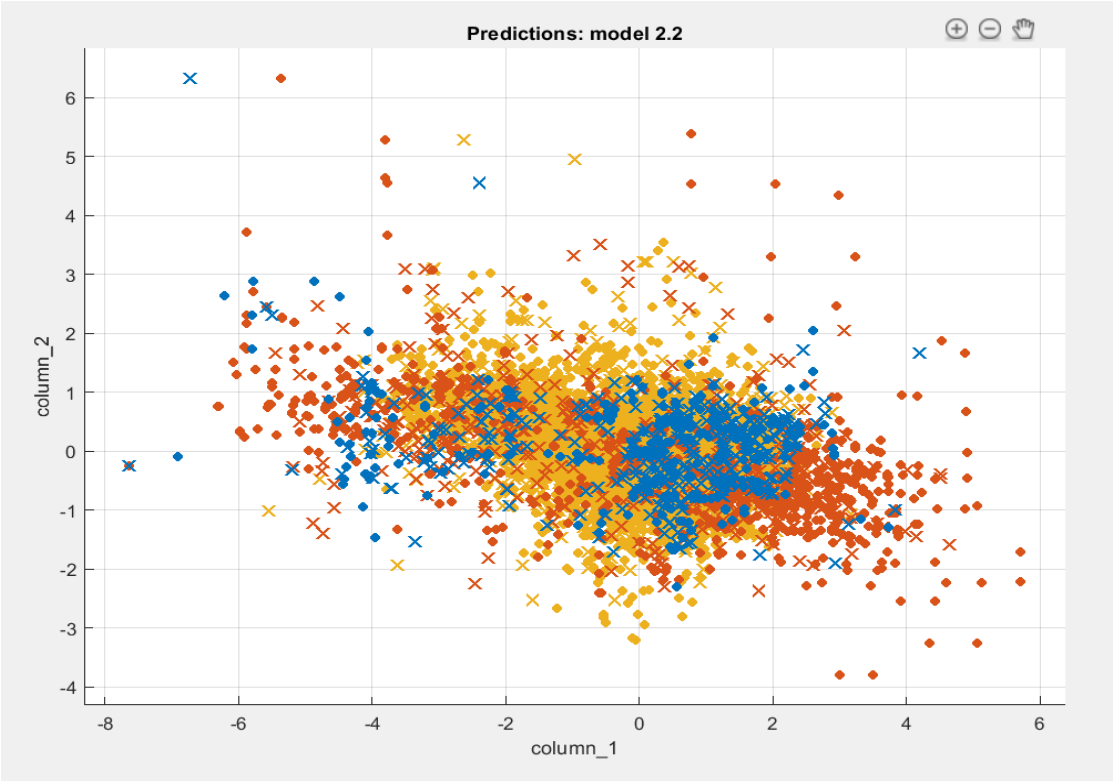
Results

Accuracy	70.4%
Prediction speed	~33000 obs/sec
Training time	7.5712 sec

Model Type

Preset: Bagged Trees
Ensemble method: Bag
Learner type: Decision tree
Number of learners: 30

The graph shows the performance of the trained classifier on the trained data.

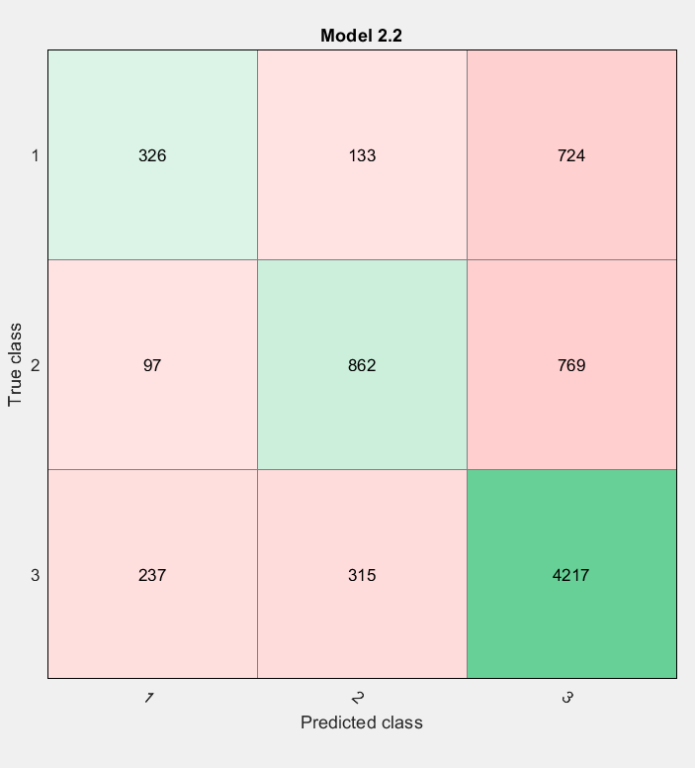


Correct

Incorrect

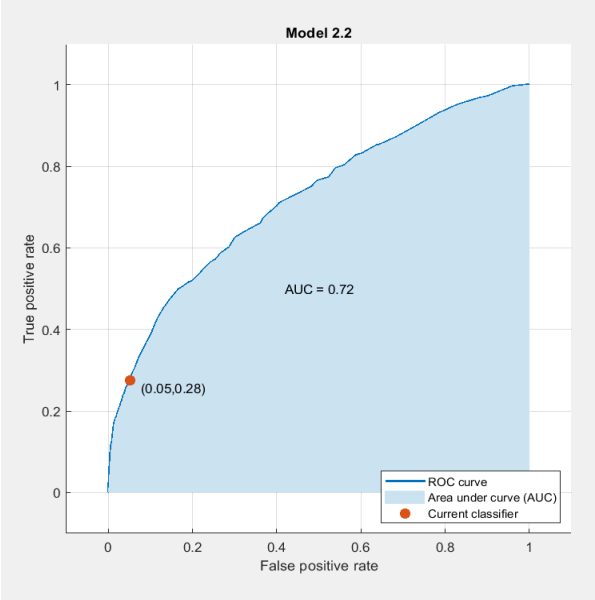
	Climbing Stairs
	Descending Stairs
	Walking

The training data comprised of approximately 8000 observations. The confusion matrix of the trained data is shown below:



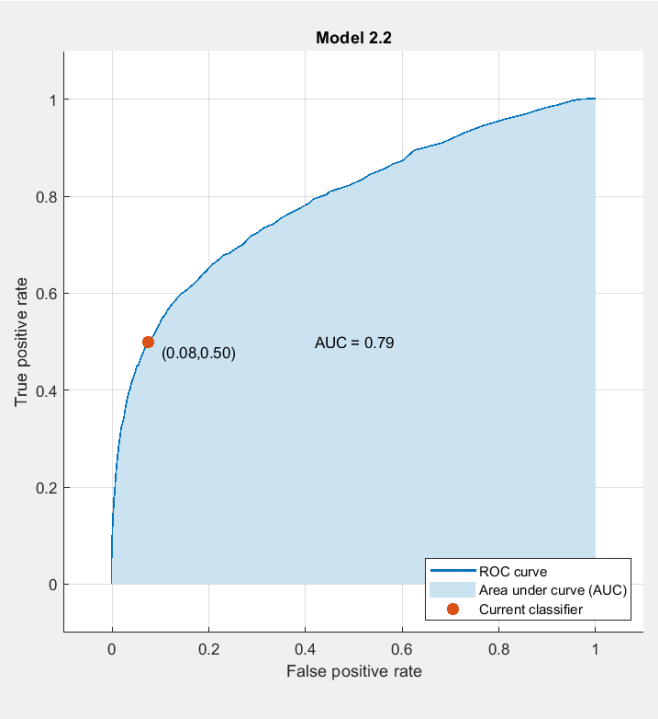
The Receiver Operating Curve (ROC) for different classes are shown below. ROC is plotted between the True positive rate and the False negative rate. Area under the curve marks the accuracy performance of the classifier

1. Climbing Stairs



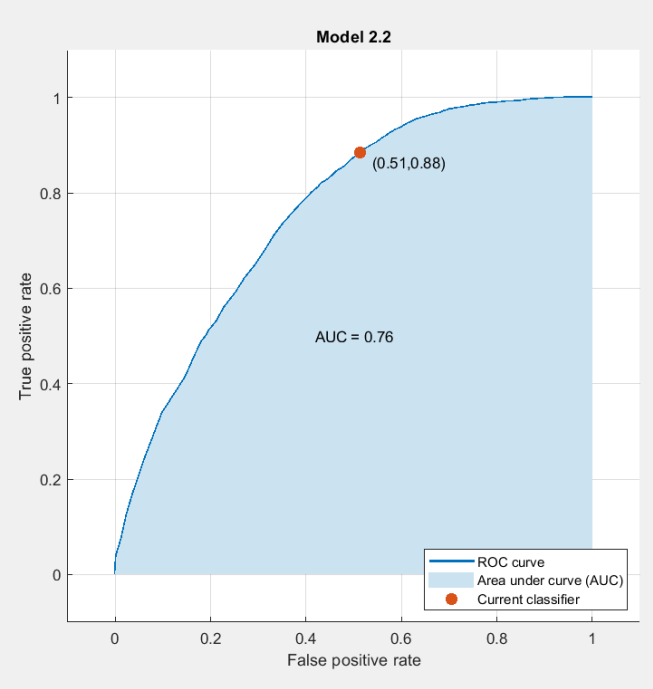
The area under the curve is 72%.

2. Descending Stairs



The area under the curve is 79%

3.Walking



The area under the curve is 76%.

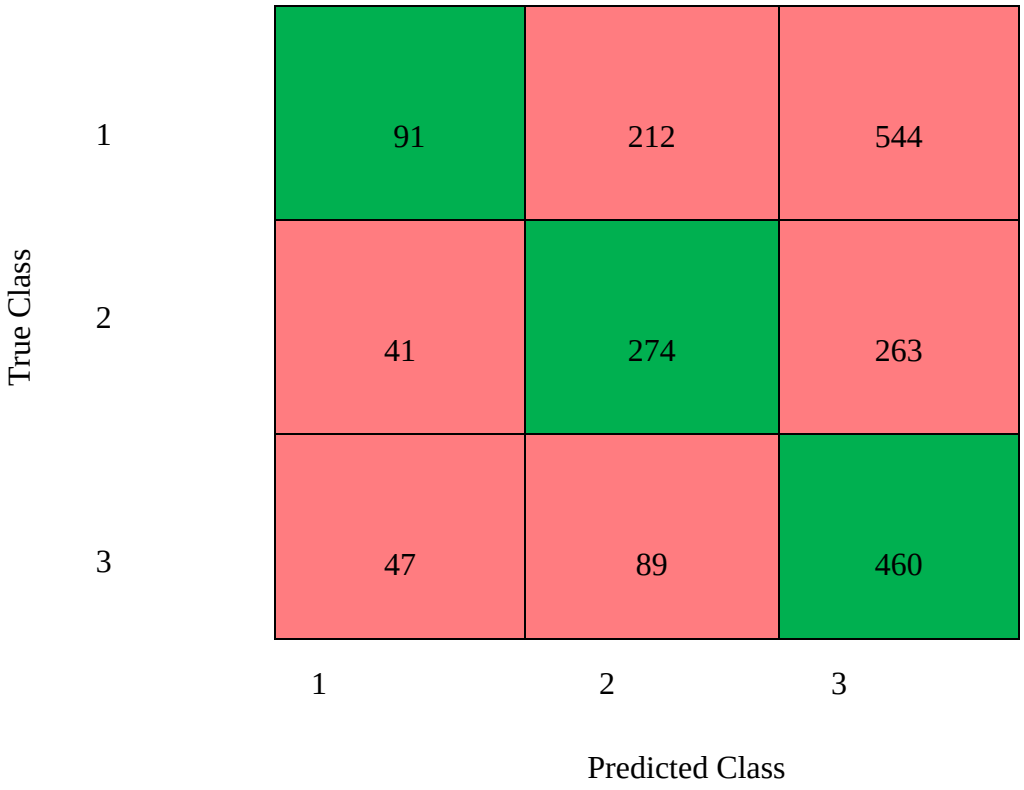
3.3.2 Performance on Test Data

Test data comprised of approximately 2000 observations. The confusion matrix of the test data is shown below:

Class 1- Climbing Stairs

Class 2- Descending Stairs

Class 3- Walking



The prediction accuracy on test data is about 41%.

PART 4- ANALYSIS

4.1 False Negatives

False Negative is the Type II error where it is shown that there is absence of a condition whereas it is present.

True positive Rates and False Negative Rates are shown in the matrix below:

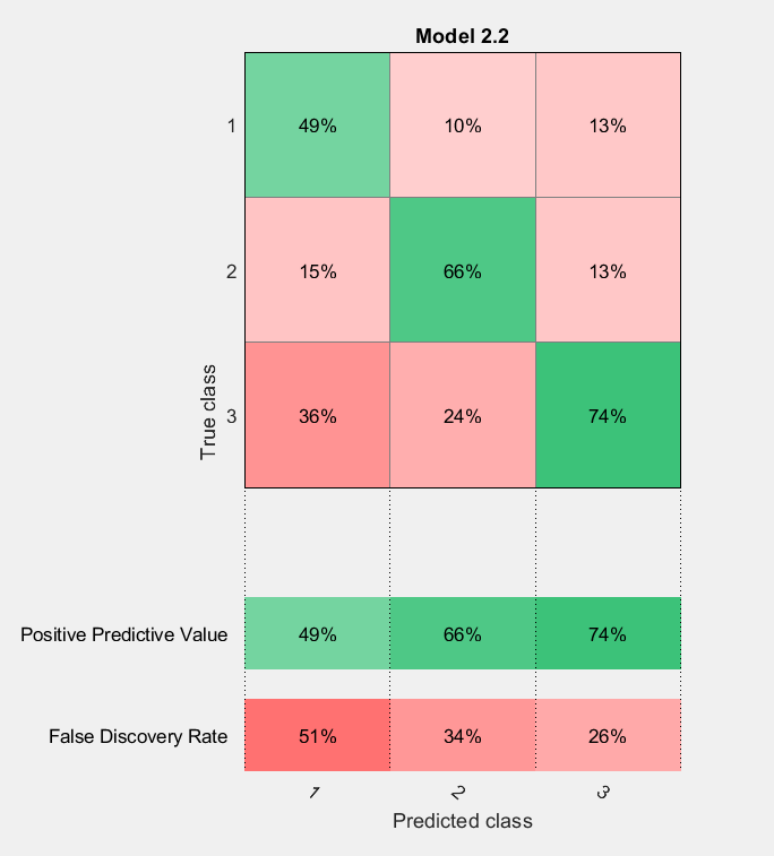
It can be observed that the positive detection for Walking shown by "3" is the highest whereas the positive detection is lowest for Climbing stairs given by "1." Descending Stairs is represented by "2." In other words, the false negative is lowest for Class 3- Walking and highest for Class 1- Climbing Stairs.

It can also be said that there is a difficulty in distinguishing between Climbing Stairs and walking as well as Descending Stairs and walking while Climbing and descending stairs are well resolved. The False negative rate is 72% for Climbing stairs, 50% for Descending stairs and 12% for walking.

4.2 False Positive

False Positive is the Type I error where it is shown that a condition is present whereas it is not.

Positive Predictive rates and False Positive rates are shown in the matrix below:



It can be observed that the positive detection for Walking shown by "3" is the highest whereas the positive detection is lowest for Climbing stairs given by "1." Descending Stairs is represented by "2." In other words, the false positive is lowest for Class 3- Walking and highest for Class 1- Climbing Stairs.

It can also be said that there is a difficulty in distinguishing between Climbing Stairs and walking as well as Descending Stairs and walking while Climbing and descending stairs are well resolved. The False positive rate is 51% for Climbing stairs, 34% for Descending stairs and 26% for walking.

4.3 Classifier Tuning

Ensemble is a method in which many weak learners are used and trained to generate a single strong learner. Thus, one of the ways to tune the classifier used (Bagged Ensemble Classifier)is to change the number of learners.

Decreasing Learners

By decreasing the number of learners, the false negative and false positives under go a shift. However, there is a compromise made on the overall prediction accuracy of the model. The overall accuracy of the prediction model has reduced from 70.4% to 67.3%.

Model 8: Trained

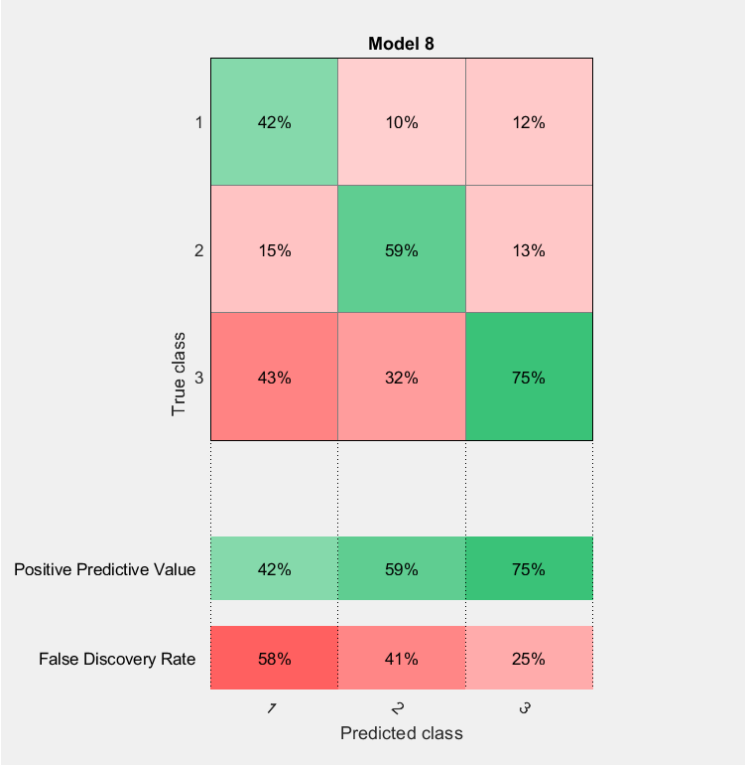
Results

Accuracy	67.3%
Prediction speed	~93000 obs/sec
Training time	3.9424 sec

Model Type

Preset: Bagged Trees
Ensemble method: Bag
Learner type: Decision tree
Number of learners: 8

For learners=8, the false negatives and false positives are shown in the confusion matrix below:



It can be observed that with a reduction in the number of learners, the false negative rates have decreased for Climbing Stairs by 6%, Descending stairs by 2% and increased for walking by 7%. Also, the false positive rates have increased by 7% for Climbing Stairs, 7% for Descending stairs and decreased by 1% for Walking.

Increasing Learners

By increasing the number of learners, the false negative and false positives under go a shift. Moreover, there is a improvement on the overall prediction accuracy of the model. The overall accuracy of the prediction model has increased from 70.4% to 71%.

Model 10: Trained

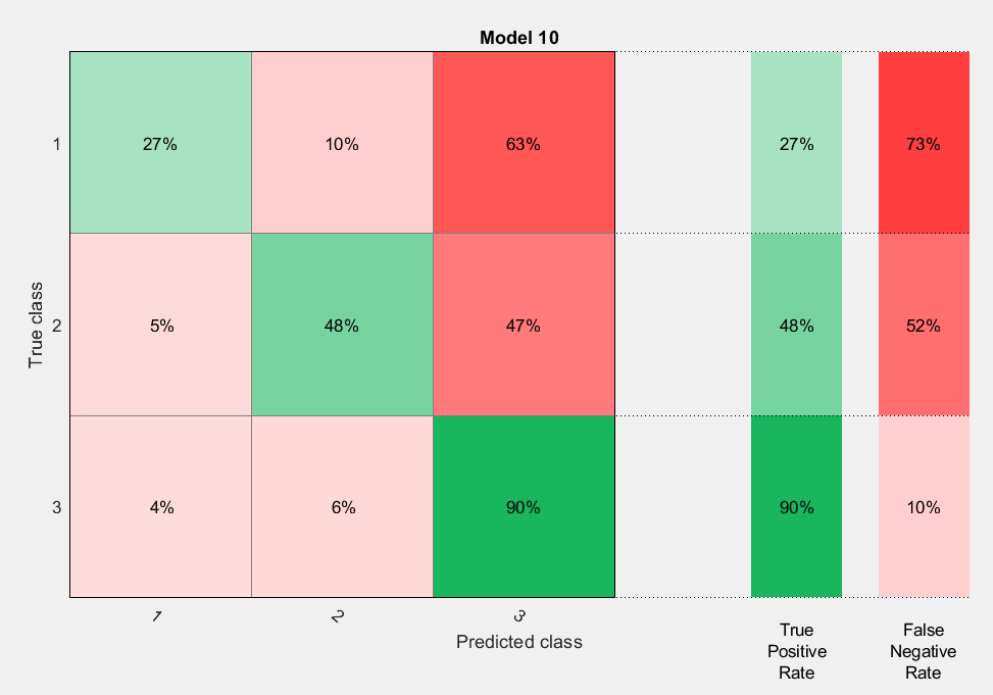
Results

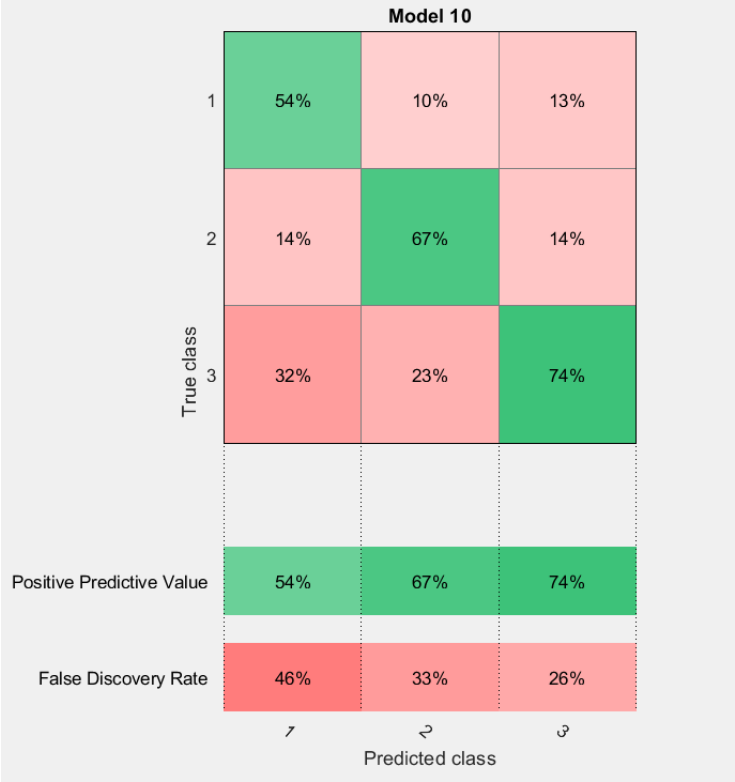
Accuracy	71.0%
Prediction speed	~9000 obs/sec
Training time	28.215 sec

Model Type

Preset: Bagged Trees
Ensemble method: Bag
Learner type: Decision tree
Number of learners: 90

For learners=90, the false negatives and false positives are shown in the confusion matrix below:





It can be observed that with an increase in the number of learners, the false negative rates have increased for Climbing Stairs by 1%, Descending stairs by 2% and walking by 2%. Also, the false

positive rates have decreased by 5% for Climbing Stairs, 1% for Descending stairs and there has been no significant change for Walking.

Also, after a point, increasing the number of learners neither does cause a shift between false positive and false negative nor does affect the accuracy. Thus, there is a limit to which the shift can be made and to the highest prediction accuracy that the model can achieve.

Thus, it can be concluded that by increasing the number of learners, there has been a shift from false positive to false negative rates whereas decreasing the number of learners results in a shift from false negative rates to false positive rates. However, the trade-off during this shift is the model prediction accuracy which increased for the former case and decreased for the latter case. This is as expected since increasing the number of learners should make the accuracy of our prediction model stronger.

APPENDIX

A. MATLAB Code

1. Code for generating Spectrographs using Kaiser window

```
folder = 'Climb_stairs/';
files = dir([folder,' Accelerometer-2011-04-11-11-58-30-
climb_stairs-f1.txt]);
numFiles = length(files);
dataFiles = zeros(1,numFiles);
noisy_x = [];
noisy_y = [];
noisy_z = [];
for i=1:1:numFiles
    dataFiles(i) = fopen([folder files(i).name],'r');
    data = fscanf(dataFiles(i),'%d\t%d\t%d\n',[3,inf]);

    % Fix the array sizes for data vectors of differing
lengths
    noisy_x = padarray(noisy_x, [0,max(size(data,2)-
size(noisy_x,2),0)],0,'post');
    noisy_y = padarray(noisy_y, [0,max(size(data,2)-
size(noisy_y,2),0)],0,'post');
    noisy_z = padarray(noisy_z, [0,max(size(data,2)-
size(noisy_z,2),0)],0,'post');

    % CONVERT THE ACCELEROMETER DATA INTO REAL ACCELERATION
VALUES
    % mapping from [0..63] to [-14.709..+14.709]
    noisy_x(i,1:size(data,2)) = -14.709 +
(data(1,:)/63)*(2*14.709);
    noisy_y(i,1:size(data,2)) = -14.709 +
(data(2,:)/63)*(2*14.709);
    noisy_z(i,1:size(data,2)) = -14.709 +
(data(3,:)/63)*(2*14.709);
end
noisy_x = transpose(noisy_x);
noisy_y = transpose(noisy_y);
noisy_z = transpose(noisy_z);

Fs = 32;  % Sampling Frequency

Fstop = 0;                % Stopband Frequency
Fpass = 2;                % Passband Frequency
```

```
Dstop = 0.0001;           % Stopband Attenuation
Dpass = 0.057501127785;   % Passband Ripple
dens  = 20;               % Density Factor

% Calculate the order from the parameters using FIRPMORD.
[N, Fo, Ao, W] = firpmord([Fstop, Fpass]/(Fs/2), [0 1],
[Dstop, Dpass]);

% Calculate the coefficients using the FIRPM function.
b  = firpm(N, Fo, Ao, W, {dens});
Hd = dfilt.dffir(b);

filtered_x=filter(Hd,noisy_x);

filtered_y=filter(Hd,noisy_y);

filtered_z=filter(Hd,noisy_z);

%%%%% %%%%%% %%%%%%%%
n1=134;
b1=1;
b2=50;
b3=100;

%Code to plot spectrographs using kaiser window keeping
number of frequency points same
%and varying beta
figure(1),
spectrogram(filtered_x,kaiser(n1,b1),0,64,32,'yaxis')
title('Filtered x Kaiser with beta value 1, zero overlap
and n=134')
figure(2),
spectrogram(filtered_x,kaiser(n1,b2),0,64,32,'yaxis')
title('Filtered x Kaiser with beta value 50, zero overlap
and n=134')
figure(3),
spectrogram(filtered_x,kaiser(n1,b3),0,64,32,'yaxis')
title('Filtered x Kaiser with beta value 100, zero overlap
and n=134')

%%
%%%%%%%%%%%%%%

n11=134;
b11=1;
```

```
%Code to plot spectrographs using kaiser window varying
number of frequency points same
%and keeping beta constant
figure(1),
spectrogram(filtered_x,kaiser(n11,b11),0,64,32,'yaxis')
title('Filtered x Kaiser with beta value 1,f=64,zero
overlap and n=134')
figure(2),
spectrogram(filtered_y,kaiser(n11,b11),0,64,32,'yaxis')
title('Filtered y Kaiser with beta value 1,f=64,zero
overlap and n=134')
figure(3),
spectrogram(filtered_z,kaiser(n11,b11),0,64,32,'yaxis')
title('Filtered z Kaiser with beta value 1,f =64, zero
overlap and n=134')
%%
n2=134;
b2=1;
%%%%%%%%%%%%%%%%%%%
%Code to plot spectrographs using kaiser window varying
number of frequency points same
%and keeping beta constant
%figure(1),
%spectrogram(filtered_x,kaiser(n2,b1),0,64,32,'yaxis')
%title('Filtered x Kaiser with beta value 1,zero overlap
and n=33')
%figure(2),
%spectrogram(filtered_x,kaiser(n2,b1),0,64,32,'yaxis')
%title('Filtered x Kaiser with beta value 1,zero overlap
and n=132')
%figure(3),
%spectrogram(filtered_x,kaiser(n3,b1),0,64,32,'yaxis')
%title('Filtered x Kaiser with beta value 1, zero overlap
and n=264')

%%
%Code to plot spectrographs using kaiser window keeping
number of frequency points same
%and varying beta
%figure(1),
%spectrogram(filtered_x,kaiser(n1,b1),30,64,32,'yaxis')
%title('Filtered x Kaiser with beta value 1, 30% overlap
and n=134')
%figure(2),
%spectrogram(filtered_x,kaiser(n1,b1),60,64,32,'yaxis')
%title('Filtered x Kaiser with beta value 1, 60% overlap
and n=134')
```

```
%figure(3),
%spectrogram(filtered_x,kaiser(n1,b1),90,64,32,'yaxis')
%title('Filtered x Kaiser with beta value 1, 90% overlap
and n=134')
```

2. Code for generating Spectrographs using Kaiser window

```
folder = 'Climb_stairs/';
files = dir([folder,' Accelerometer-2011-04-11-11-58-30-
climb_stairs-f1.txt
]);
numFiles = length(files);
dataFiles = zeros(1,numFiles);
noisy_x = [];
noisy_y = [];
noisy_z = [];
for i=1:1:numFiles
    dataFiles(i) = fopen([folder files(i).name],'r');
    data = fscanf(dataFiles(i),'%d\t%d\t%d\n',[3,inf]);

    % Fix the array sizes for data vectors of differing
lengths
    noisy_x = padarray(noisy_x, [0,max(size(data,2)-
size(noisy_x,2),0)],0,'post');
    noisy_y = padarray(noisy_y, [0,max(size(data,2)-
size(noisy_y,2),0)],0,'post');
    noisy_z = padarray(noisy_z, [0,max(size(data,2)-
size(noisy_z,2),0)],0,'post');

    % CONVERT THE ACCELEROMETER DATA INTO REAL ACCELERATION
VALUES
    % mapping from [0..63] to [-14.709..+14.709]
    noisy_x(i,1:size(data,2)) = -14.709 +
(data(1,:)/63)*(2*14.709);
    noisy_y(i,1:size(data,2)) = -14.709 +
(data(2,:)/63)*(2*14.709);
    noisy_z(i,1:size(data,2)) = -14.709 +
(data(3,:)/63)*(2*14.709);
end
noisy_x = transpose(noisy_x);
noisy_y = transpose(noisy_y);
noisy_z = transpose(noisy_z);

%Fs = 32;  % Sampling Frequency
```

```
%Fstop = 0;                % Stopband Frequency
%Fpass = 2;                % Passband Frequency
%Dstop = 0.0001;           % Stopband Attenuation
%Dpass = 0.057501127785;   % Passband Ripple
%dens  = 20;               % Density Factor

% Calculate the order from the parameters using FIRPMORD.
%[N, Fo, Ao, W] = firpmord([Fstop, Fpass]/(Fs/2), [0 1],
[Dstop, Dpass]);

% Calculate the coefficients using the FIRPM function.
%b  = firpm(N, Fo, Ao, W, {dens});
%Hd = dfilt.dffir(b);

%filtered_x=filter(Hd,noisy_x);

%filtered_y=filter(Hd,noisy_y);

%filtered_z=filter(Hd,noisy_z);


%%%%%%%%%%%%%%%%%%%%%%%%

%Code to plot spectrographs using kaiser window keeping
number of frequency points same
%and varying beta

n1=67;
n2=134;
n3=268;
%%
%figure(1),
%spectrogram(filtered_x,hamming(n1),0,64,32,'yaxis')
%title('Hamming with zero overlap and n=67 and
frequency=64')
%figure(2),
%spectrogram(filtered_x,hamming(n2),0,64,32,'yaxis')
%title('Hamming with zero overlap and n=134 and
frequency=64')
%figure(3),
%spectrogram(filtered_x,hamming(n3),0,64,32,'yaxis')
%title('Hamming with zero overlap and n=268 and
frequency=64')
```

3. Code for generating Spectrographs using Flat Top Weighted window

```
folder = 'Climb_stairs/';
files = dir([folder,'Accelerometer-2011-04-11-11-58-30-
climb_stairs-f1.txt']);
numFiles = length(files);
dataFiles = zeros(1,numFiles);
noisy_x = [];
noisy_y = [];
noisy_z = [];
for i=1:1:numFiles
    dataFiles(i) = fopen([folder files(i).name],'r');
    data = fscanf(dataFiles(i),'%d\t%d\t%d\n',[3,inf]);

    % Fix the array sizes for data vectors of differing
lengths
    noisy_x = padarray(noisy_x, [0,max(size(data,2)-
size(noisy_x,2),0)],0,'post');
    noisy_y = padarray(noisy_y, [0,max(size(data,2)-
size(noisy_y,2),0)],0,'post');
    noisy_z = padarray(noisy_z, [0,max(size(data,2)-
size(noisy_z,2),0)],0,'post');

    % CONVERT THE ACCELEROMETER DATA INTO REAL ACCELERATION
VALUES
    % mapping from [0..63] to [-14.709..+14.709]
    noisy_x(i,1:size(data,2)) = -14.709 +
(data(1,:)/63)*(2*14.709);
    noisy_y(i,1:size(data,2)) = -14.709 +
(data(2,:)/63)*(2*14.709);
    noisy_z(i,1:size(data,2)) = -14.709 +
(data(3,:)/63)*(2*14.709);
end
```

```
noisy_x = transpose(noisy_x);
noisy_y = transpose(noisy_y);
noisy_z = transpose(noisy_z);

Fs = 32;  % Sampling Frequency

Fstop = 0;               % Stopband Frequency
Fpass = 2;               % Passband Frequency
Dstop = 0.0001;          % Stopband Attenuation
Dpass = 0.057501127785;  % Passband Ripple
dens  = 20;              % Density Factor

% Calculate the order from the parameters using FIRPMORD.
[N, Fo, Ao, W] = firpmord([Fstop, Fpass]/(Fs/2), [0 1],
[Dstop, Dpass]);

% Calculate the coefficients using the FIRPM function.
b  = firpm(N, Fo, Ao, W, {dens});
Hd = dfilt.dffir(b);

filtered_x=filter(Hd,noisy_x);

filtered_y=filter(Hd,noisy_y);

filtered_z=filter(Hd,noisy_z);


%%%%%%%%%%

n2=134;
n3=268;
n1=67;
%We only use symmetric. Periodic is not for stft
figure(1),
spectrogram(filtered_x,flattopwin(n1,'symmetric'),0,64,32,'
yaxis')
title('Flat Top Window-Symmetric with zero overlap and n=67
and frequency=64')
figure(2),
spectrogram(filtered_x,flattopwin(n2,'symmetric'),0,64,32,'
yaxis')
title('Flat Top Window-Symmetric with zero overlap and
n=134 and frequency=64')
%figure(3),
spectrogram(filtered_x,flattopwin(n3,'symmetric'),0,64,32,'
yaxis')
```

```
%title('Flat Top Window-Symmetric with zero overlap and
n=268 and frequency=64')
figure(3),
spectrogram(x_set,flattopwin(n2,'symmetric'),0,64,32,'yaxis
')
title('Flat Top Window with zero overlap and n=264 and
frequency=64')
figure(3),
spectrogram(x_set,flattopwin(n3,'symmetric'),0,64,32,'yaxis
')
title('Flat Top Window with zero overlap and n=66 and
frequency=64')
```

4. Code for generating Bandpass Filter

```
function Hd = mymodel_bandpassfilter
%MYMODEL_BANDPASSFILTER Returns a discrete-time filter
object.


% Equiripple Bandpass filter designed using the FIRPM
function.

% All frequency values are in Hz.
Fs = 32;  % Sampling Frequency

Fstop1 = 0;               % First Stopband Frequency
Fpass1 = 1.5;             % First Passband Frequency
Fpass2 = 6;               % Second Passband Frequency
Fstop2 = 8;               % Second Stopband Frequency
Dstop1 = 0.001;           % First Stopband Attenuation
Dpass  = 0.057501127785;  % Passband Ripple
Dstop2 = 0.0001;          % Second Stopband Attenuation
dens   = 20;              % Density Factor

% Calculate the order from the parameters using FIRPMORD.
[N, Fo, Ao, W] = firpmord([Fstop1 Fpass1 Fpass2
Fstop2]/(Fs/2), [0 1 ...
                          0], [Dstop1 Dpass Dstop2]);

% Calculate the coefficients using the FIRPM function.
```

```
b  = firpm(N, Fo, Ao, W, {dens});
Hd = dfilt.dffir(b);

% [EOF]
fvtool(Hd,'Analysis','freq')
```

Code for processing raw data using Bandpass Filter

```
folder = 'Climb_stairs/';
files = dir([folder,'Accelerometer-2011-05-30-21-53-35-
descend_stairs-m2.txt']);
numFiles = length(files);
dataFiles = zeros(1,numFiles);
noisy_x = [];
noisy_y = [];
noisy_z = [];
for i=1:1:numFiles
    dataFiles(i) = fopen([folder files(i).name],'r');
    data = fscanf(dataFiles(i),'%d\t%d\t%d\n',[3,inf]);

    % Fix the array sizes for data vectors of differing
lengths
    noisy_x = padarray(noisy_x, [0,max(size(data,2)-
size(noisy_x,2),0)],0,'post');
    noisy_y = padarray(noisy_y, [0,max(size(data,2)-
size(noisy_y,2),0)],0,'post');
    noisy_z = padarray(noisy_z, [0,max(size(data,2)-
size(noisy_z,2),0)],0,'post');

    % CONVERT THE ACCELEROMETER DATA INTO REAL ACCELERATION
VALUES
    % mapping from [0..63] to [-14.709..+14.709]
    noisy_x(i,1:size(data,2)) = -14.709 +
(data(1,:)/63)*(2*14.709);
    noisy_y(i,1:size(data,2)) = -14.709 +
(data(2,:)/63)*(2*14.709);
    noisy_z(i,1:size(data,2)) = -14.709 +
(data(3,:)/63)*(2*14.709);
end
noisy_x = transpose(noisy_x);
noisy_y = transpose(noisy_y);
noisy_z = transpose(noisy_z);

%Fs = 32;  % Sampling Frequency
```

```
%Fstop = 0;                % Stopband Frequency
%Fpass = 2;                % Passband Frequency
%Dstop = 0.0001;           % Stopband Attenuation
%Dpass = 0.057501127785;   % Passband Ripple
%dens  = 20;               % Density Factor

% Calculate the order from the parameters using FIRPMORD.
%[N, Fo, Ao, W] = firpmord([Fstop, Fpass]/(Fs/2), [0 1],
[Dstop, Dpass]);

% Calculate the coefficients using the FIRPM function.
%b  = firpm(N, Fo, Ao, W, {dens});
%Hd = dfilt.dffir(b);

%filtered_x=filter(Hd,noisy_x);

%filtered_y=filter(Hd,noisy_y);

%filtered_z=filter(Hd,noisy_z);
numSamples = length(noisy_x(:,1));

% DISPLAY THE RESULTS
time = 1:1:numSamples;

Fs = 32;  % Sampling Frequency

Fstop1 = 0;               % First Stopband Frequency
Fpass1 = 1.5;             % First Passband Frequency
Fpass2 = 6;               % Second Passband Frequency
Fstop2 = 8;               % Second Stopband Frequency
Dstop1 = 0.001;           % First Stopband Attenuation
Dpass  = 0.057501127785;  % Passband Ripple
Dstop2 = 0.0001;          % Second Stopband Attenuation
dens   = 20;              % Density Factor

% Calculate the order from the parameters using FIRPMORD.
[N, Fo, Ao, W] = firpmord([Fstop1 Fpass1 Fpass2
Fstop2]/(Fs/2), [0 1 ...
                          0], [Dstop1 Dpass Dstop2]);

% Calculate the coefficients using the FIRPM function.
b1  = firpm(N, Fo, Ao, W, {dens});
Hd2 = dfilt.dffir(b1);

%%
bandx=filter(Hd2,noisy_x);
bandy=filter(Hd2,noisy_y);
```

```
bandz=filter(Hd2,noisy_z);

figure,
    subplot(3,1,1)
    plot(time,noisy_x,'-')
    axis([0 numSamples -14.709 +14.709])
    title('Noisy accelerations along the x axis')
    subplot(3,1,2)
    plot(time,noisy_y,'-')
    axis([0 numSamples -14.709 +14.709])
    ylabel('acceleration [m/s^2]')
    title('Noisy accelerations along the y axis')
    subplot(3,1,3)
    plot(time,noisy_z,'-')
    axis([0 numSamples -14.709 +14.709])
    xlabel('time [samples]')
    title('Noisy accelerations along the z axis')
% clean signal
figure,
    subplot(3,1,1)
    plot(time,bandx,'-')
    axis([0 numSamples -14.709 +14.709])
    title('Filtered accelerations along the x axis')
    subplot(3,1,2)
    plot(time,bandy,'-')
    axis([0 numSamples -14.709 +14.709])
    ylabel('acceleration [m/s^2]')
    title('Filtered accelerations along the y axis');
    subplot(3,1,3)
    plot(time,bandz,'-')
    axis([0 numSamples -14.709 +14.709])
    xlabel('time [samples]')
    title('Filtered accelerations along the z axis')
```

5. Code for Training the classifier and performing testing on new data set

```
%function [trainedClassifier, validationAccuracy] =
trainClassifier(trainingData)
% [trainedClassifier, validationAccuracy] =
trainClassifier(trainingData)
% returns a trained classifier and its accuracy. This code
recreates the
% classification model trained in Classification Learner
app. Use the
% generated code to automate training the same model with
new data, or to
% learn how to programmatically train models.
%
%  Input:
%      trainingData: a matrix with the same number of
columns and data type
%       as imported into the app.
%
%  Output:
%      trainedClassifier: a struct containing the trained
classifier. The
%       struct contains various fields with information
about the trained
%       classifier.
%
%      trainedClassifier.predictFcn: a function to make
predictions on new
%       data.
%
%      validationAccuracy: a double containing the accuracy
in percent. In
%       the app, the History list displays this overall
accuracy score for
%       each model.
%
% Use the code to train the model with new data. To retrain
your
% classifier, call the function from the command line with
your original
% data or new data as the input argument trainingData.
%
% For example, to retrain a classifier trained with the
original data set
% T, enter:
%   [trainedClassifier, validationAccuracy] =
trainClassifier(T)
%
```

```
% To make predictions with the returned 'trainedClassifier'
on new data T2,
% use
%   yfit = trainedClassifier.predictFcn(T2)
%
% T2 must be a matrix containing only the predictor columns
used for
% training. For details, enter:
%   trainedClassifier.HowToPredict

% Auto-generated by MATLAB on 23-Nov-2018 02:51:05


% Extract predictors and response
% This code processes the data into the right shape for
training the
% model.
% Convert input to table


inputTable = array2table(xx, 'VariableNames', {'column_1',
'column_2', 'column_3', 'column_4'});

predictorNames = {'column_1', 'column_2', 'column_3'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_4;
isCategoricalPredictor = [false, false, false];

% Train a classifier
% This code specifies all the classifier options and trains
the classifier.
t = templateTree(...
    'MaxNumSplits', 7679);
Md1 = fitcensemble(...
    predictors, ...
    response, ...
    'Method', 'Bag', ...
    'NumLearningCycles', 30, ...
    'Learners', t, ...
    'ClassNames', [1; 2; 3]);

% Create the result struct with predict function
predictorExtractionFcn = @(x) array2table(x,
'VariableNames', predictorNames);
ensemblePredictFcn = @(x) predict(Md1, x);
trainedClassifier.predictFcn = @(x)
ensemblePredictFcn(predictorExtractionFcn(x));
```

```
% Add additional fields to the result struct
trainedClassifier.ClassificationEnsemble = Md1;
trainedClassifier.About = 'This struct is a trained model
exported from Classification Learner R2018a.';
trainedClassifier.HowToPredict = sprintf('To make
predictions on a new predictor column matrix, X, use: \n
yfit = c.predictFcn(X) \nreplacing ''c'' with the name of
the variable that is this struct, e.g. ''trainedModel''. \n
\nX must contain exactly 3 columns because this model was
trained using 3 predictors. \nX must contain only predictor
columns in exactly the same order and format as your
training \ndata. Do not include the response column or any
columns you did not import into the app. \n \nFor more
information, see <a href="matlab:helpview(fullfile(docroot,
''stats'', ''stats.map''),
''appclassification_exportmodeltoworkspace'')">How to
predict using an exported model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for
training the
% model.
% Convert input to table
inputTable = array2table(xx, 'VariableNames', {'column_1',
'column_2', 'column_3', 'column_4'});

predictorNames = {'column_1', 'column_2', 'column_3'};
predictors = inputTable(:, predictorNames);
response = inputTable.column_4;
isCategoricalPredictor = [false, false, false];

% Perform cross-validation
partitionedModel =
crossval(trainedClassifier.ClassificationEnsemble, 'KFold',
5);

% Compute validation predictions
[validationPredictions, validationScores] =
kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel,
'LossFun', 'ClassifError');
%view(classificationEnsemble.Trees{1},'Mode','Graph')
%Tree10 = Md1.Trees{10};
%view(Md1.Trained{10},'Mode','graph');
```

```
%view(Tree10,'Mode','graph');
yfit = trainedClassifier.predictFcn(T2);
C=confusionmat(ytrue,yfit)
plotconfusion(ytrue,yfit)
```