MINI-PROJECT

Adheesh Chatterjee **24th Nov 2018**
116236935

SPECTROGRAM

Spectrogram (X,WINDOW,NOVERLAP,NFFT,Fs)

X specifies the vector for which the short-time Fourier transform of the signal is calculated

WINDOW is a vector which divides X into segments of the same length as WINDOW, and then windows each segment with the vector specified in WINDOW. If WINDOW is an integer, the function divides X into segments of length equal to that integer value and windows each segment with a Hamming window. If WINDOW is not specified, the default is used.

NOVERLAP specifies the samples of overlap between adjoining segments. If NOVERLAP is not specified, the default value is used to obtain a 50% overlap.

NFFT specifies the number of frequency points used to calculate the discrete Fourier transforms. If NFFT is not specified, the default NFFT is used.

Fs specifies the sample rate in Hz. If Fs is specified as empty, it defaults to 1 Hz. If it is not specified, normalized frequency is used.

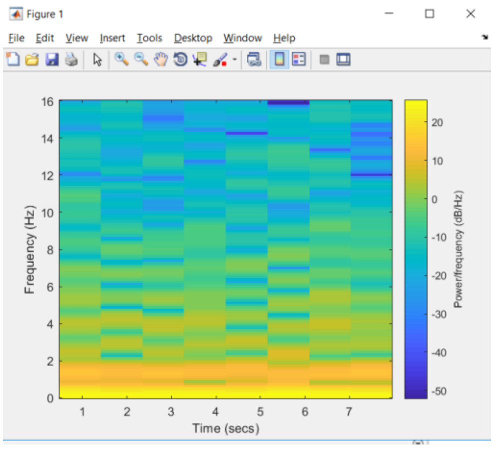
CODE

Spectrogram (X,[ ],[ ],[ ],32)

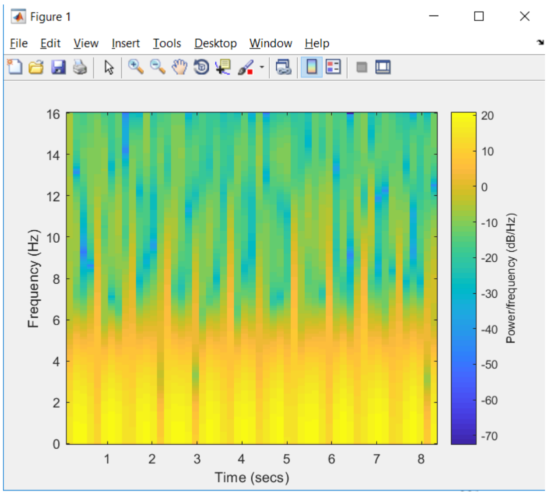
This code is used for different activities and different measurements to further understand the spectrogram.

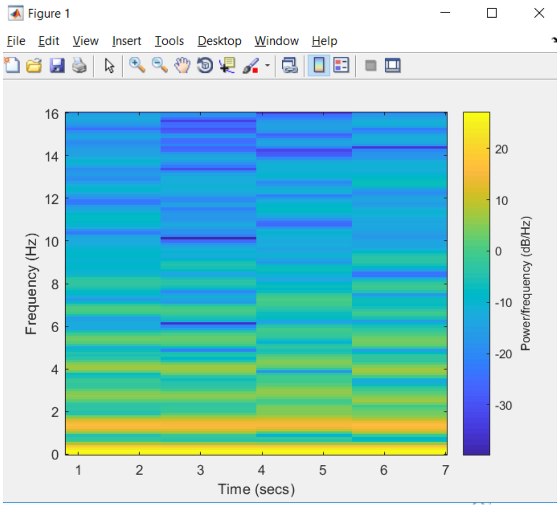
By changing the window size and the sampling frequencies, we learn more about the spectral properties of the data

For Climbing Stairs –



This is the spectrogram of the x component of the acceleration for a default window size

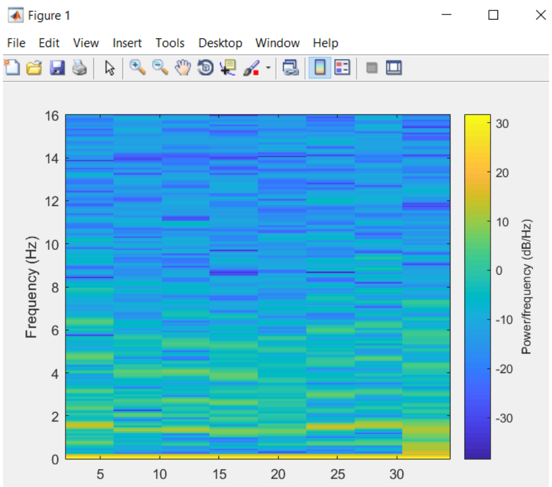




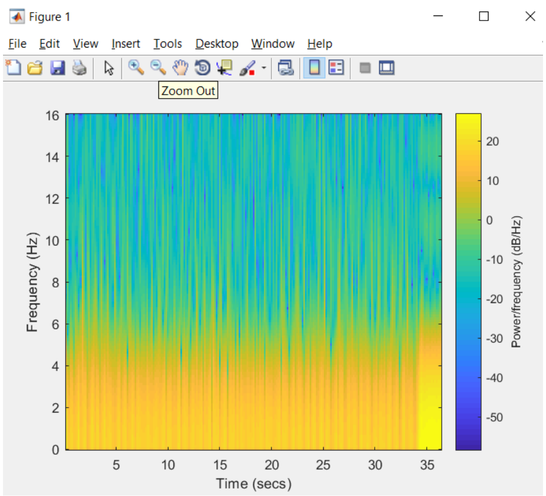
Spectrogram (X,10,[ ],[ ],32,’yaxis’)

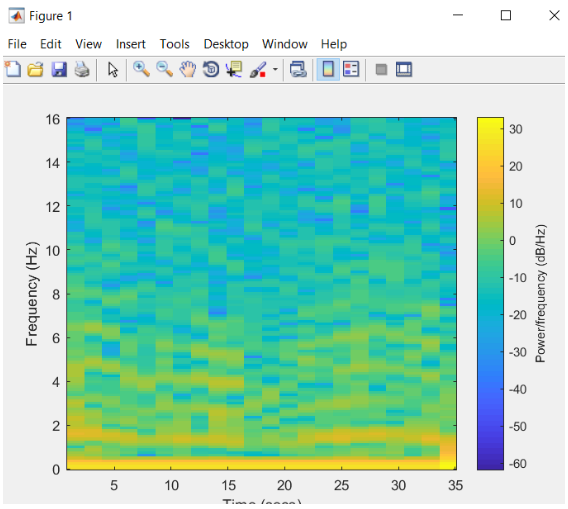
Spectrogram (X,100,[ ],[ ],32,’yaxis’)

for **Walking –**



This is the spectrogram of the x component of the acceleration for a default window size

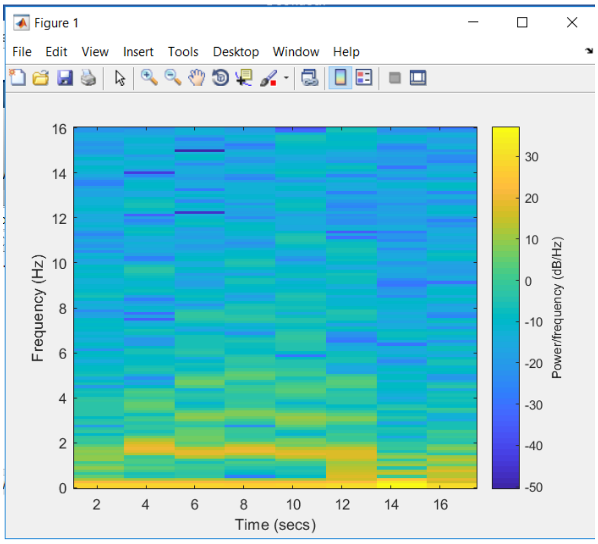




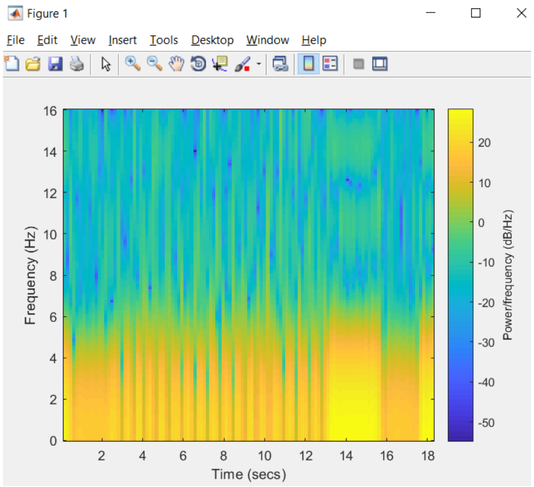
Spectrogram (X,10,[ ],[ ],32,’yaxis’)

Spectrogram (X,100,[ ],[ ],32,’yaxis’)

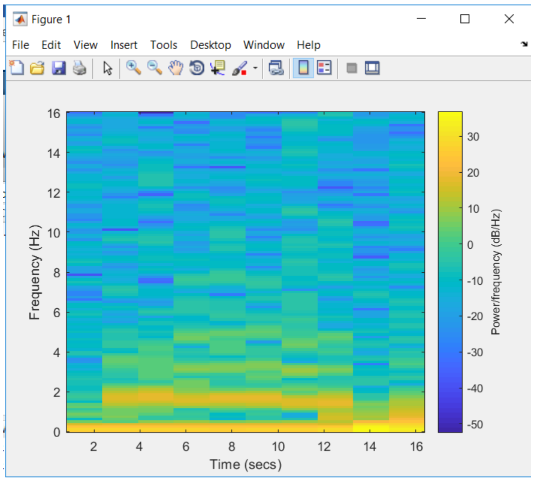
For **Descending Stairs**



This is the spectrogram of the x component of the acceleration for a default window size



Spectrogram (X,10,[ ],[ ],32,'yaxis')



Spectrogram (X,100,[ ],[ ],32,'yaxis')

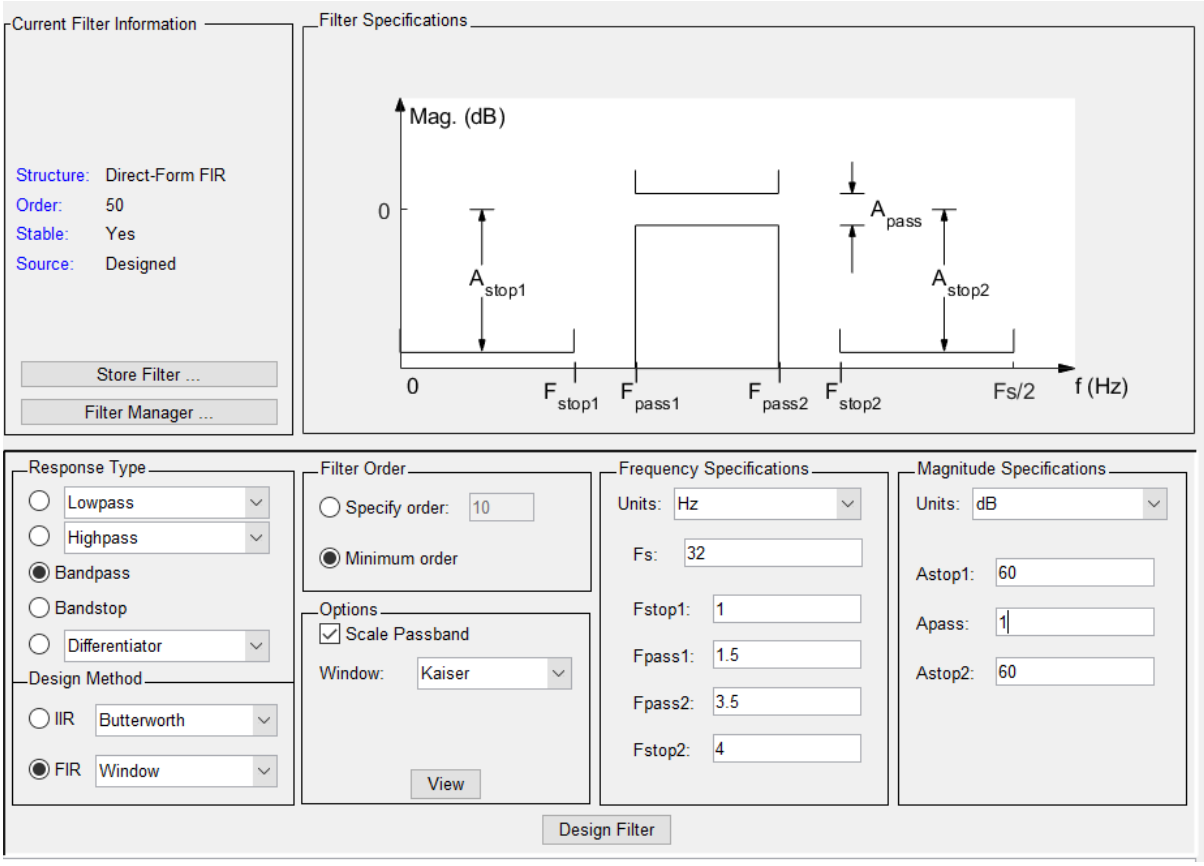
If we look at the default spectrograms for all 3 activities we notice that the highest frequency activity is in climbing stairs followed by descending stairs and finally there is very less activity in walking.

The above inference is made by only comparing the x component of the acceleration data. By comparing different a different accelerometer axis reading we can arrive at a similar result.

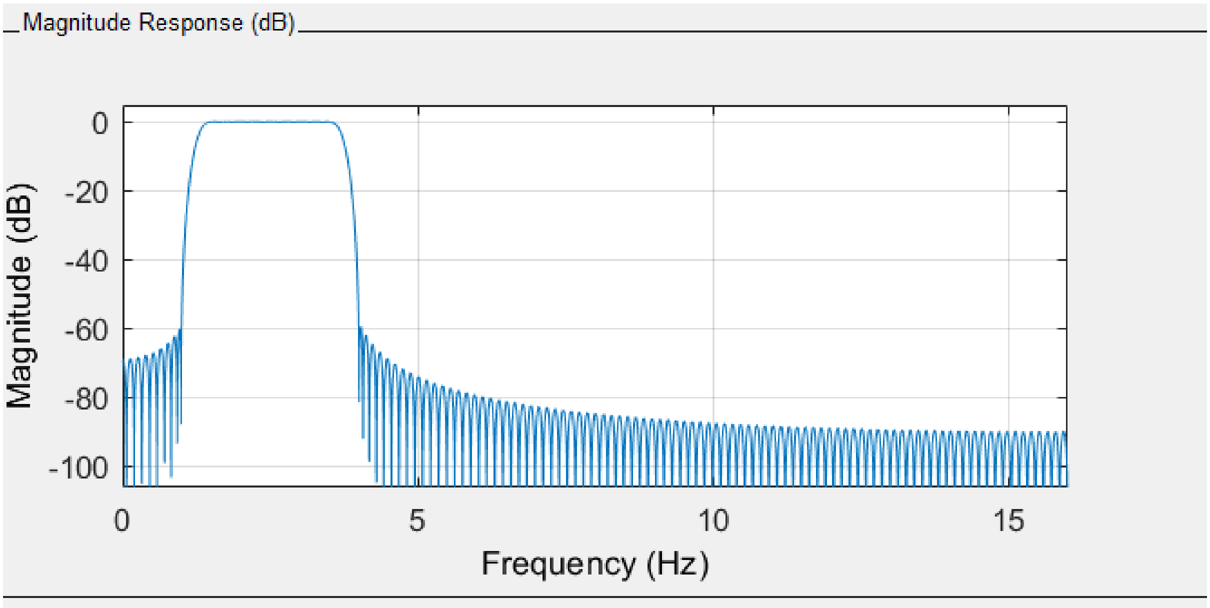
Windowing essentially enhances the ability of the FFT to extract spectral data from signals. Windowing functions act on raw data to reduce the effects of the leakage that occurs during an FFT of the data. There are different types of windows like Hamming window, rectangular, Kaiser etc, each of which help smoothen the curve.

FILTER

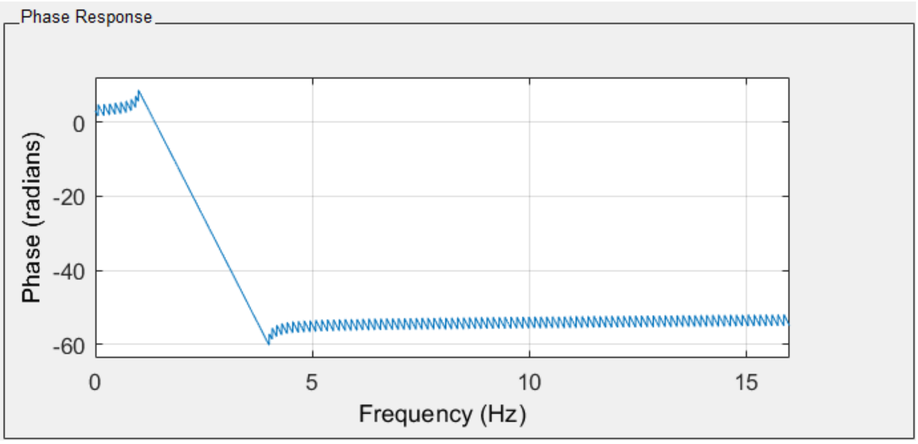
A Kaiser Window FIR filter is designed with bandwidth between 1Hz and 4Hz and a sampling frequency of 32Hz as shown



The magnitude response is –

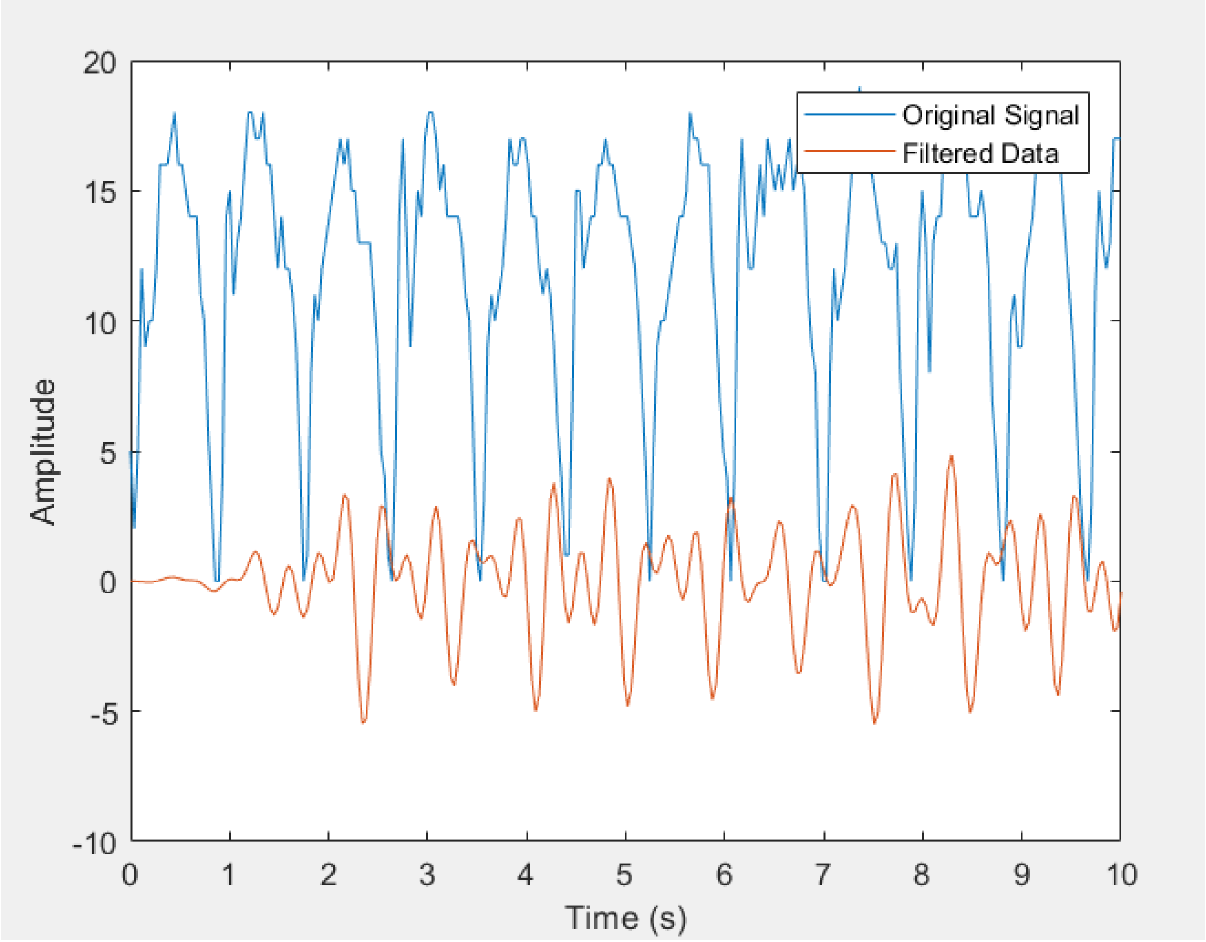


The phase response is –



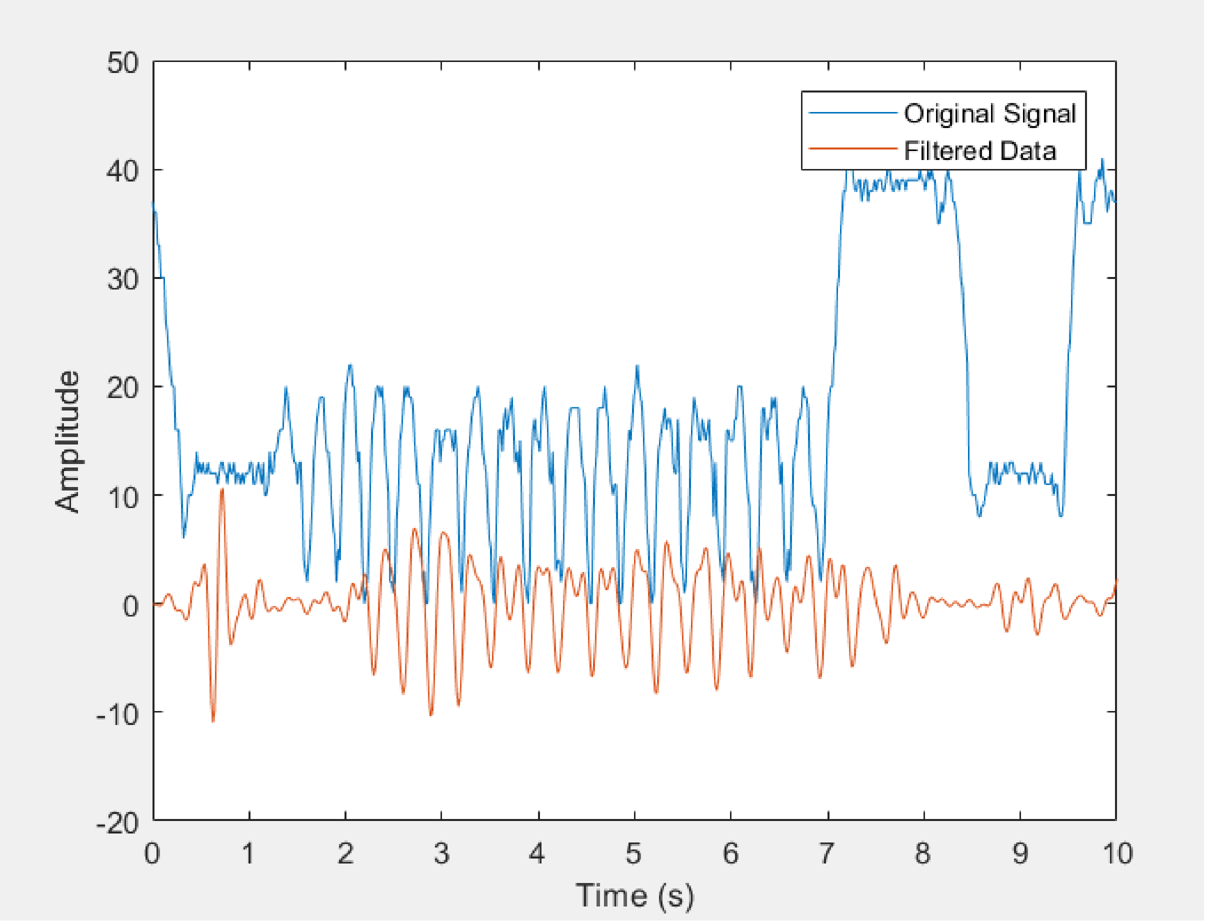
MATLAB CODE

```
Fs = 32;
t = linspace(0,10,270);
y2 = filter(Hd,ax);
plot(t,ax,t,y2)
xlabel('Time (s)')
ylabel('Amplitude')
legend('Original Signal','Filtered Data')
```

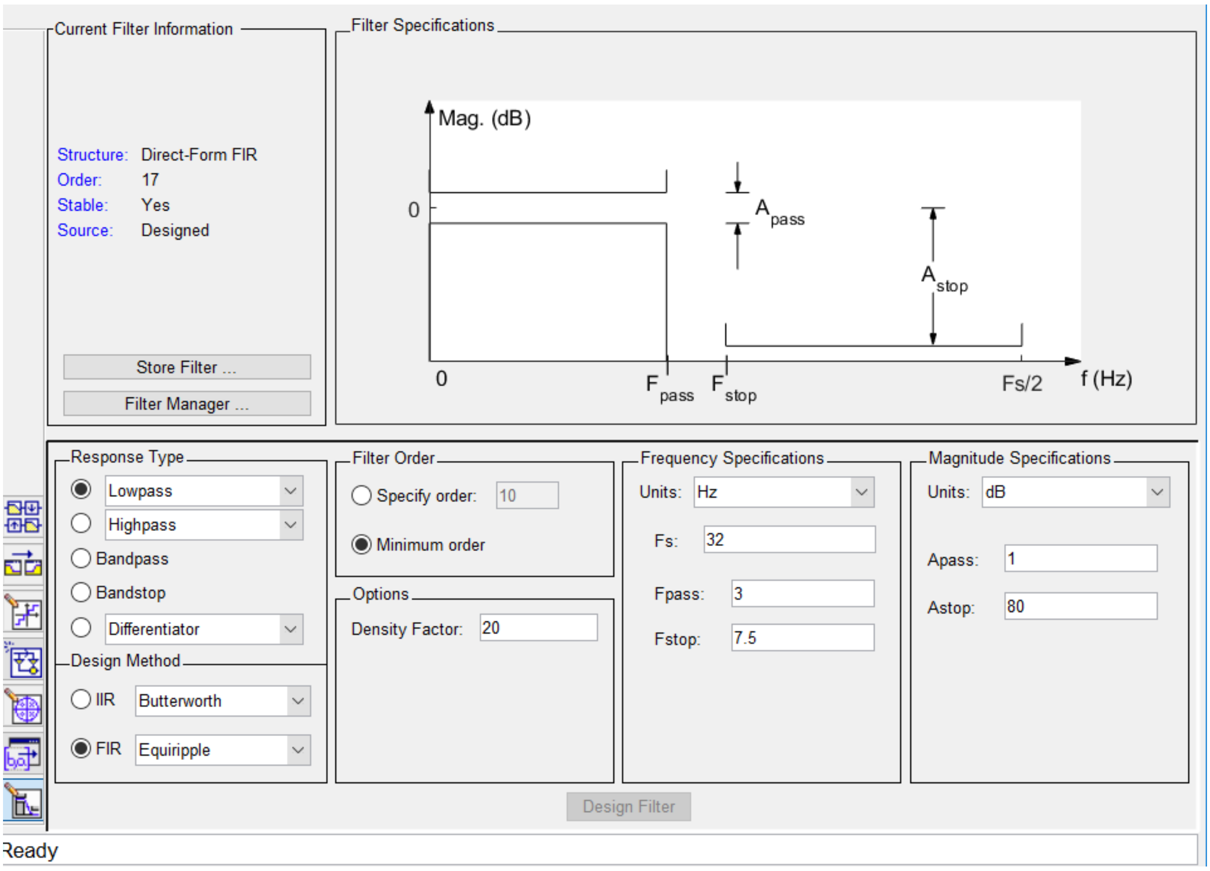


MATLAB CODE

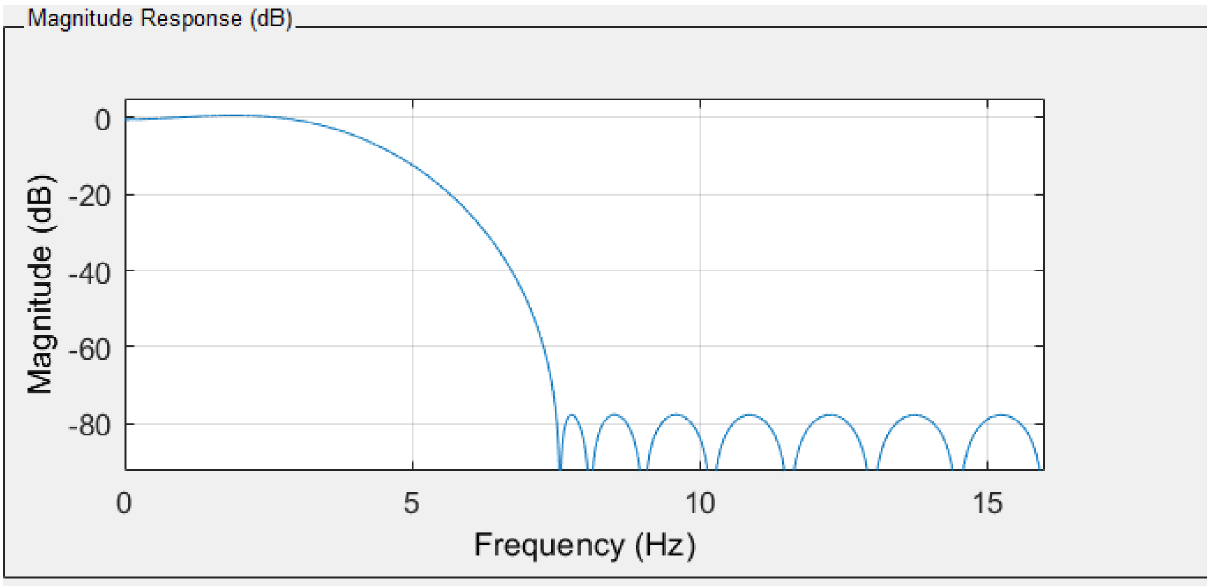
```
Fs = 32;
t = linspace(0,10,594);
y2 = filter(Hd,VarName1);
plot(t,VarName1,t,y2)
xlabel('Time (s)')
ylabel('Amplitude')
legend('Original Signal','Filtered Data')
```



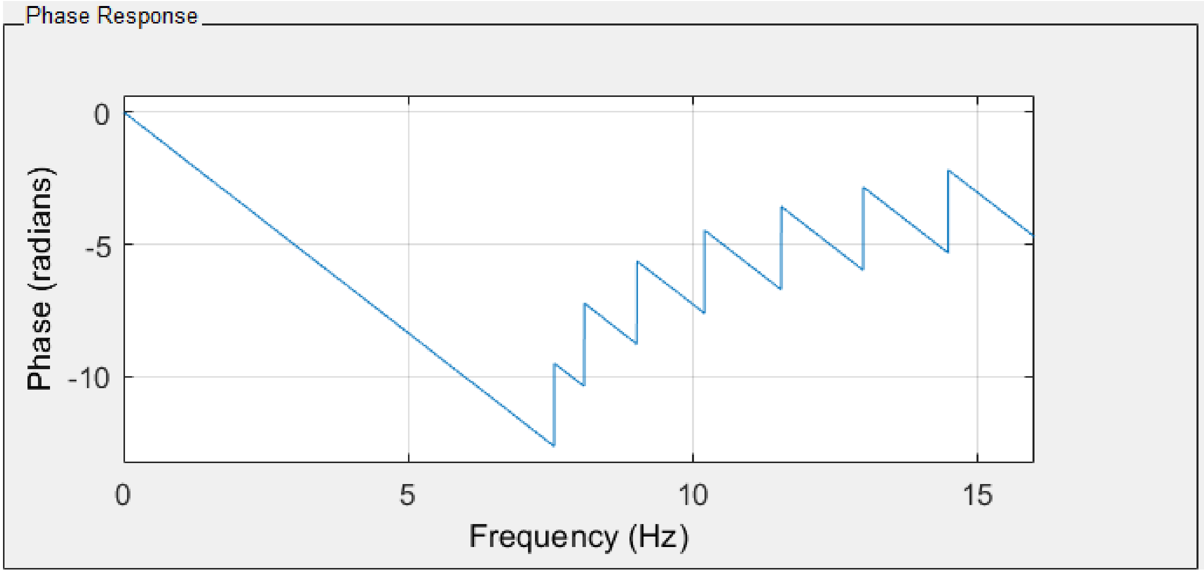
Although after analyzing the data I feel it is better to use a low-pass filter as it better fits the data



The magnitude response is –

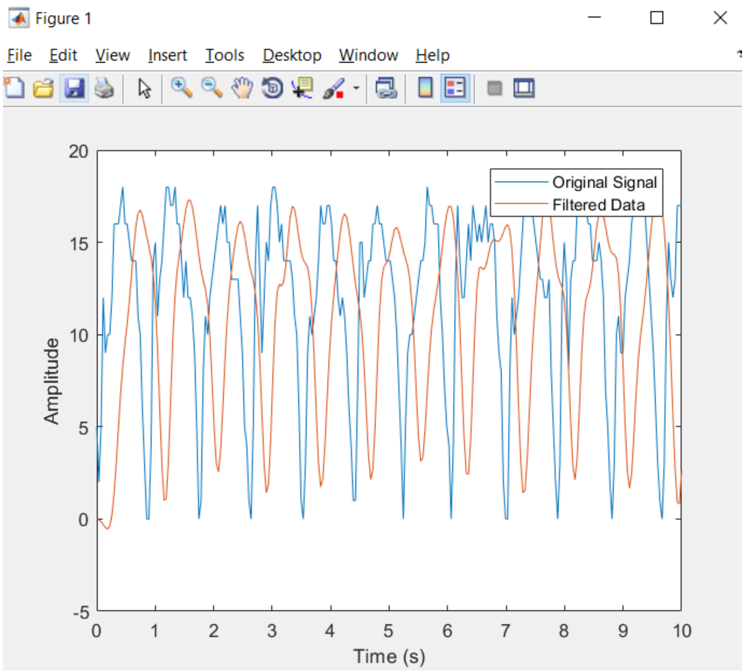


The phase response is –



MATLAB CODE

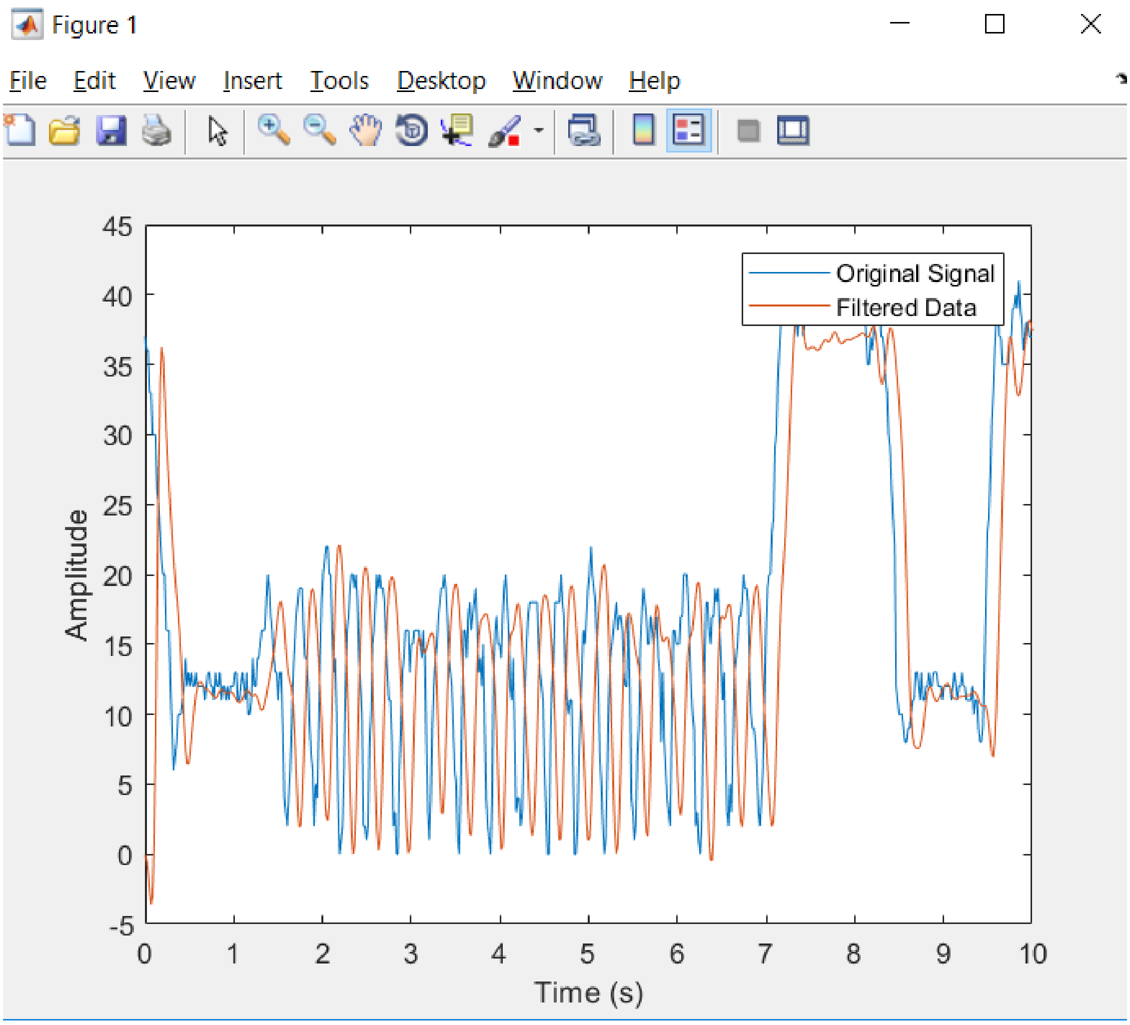
```
Fs = 32;
t = linspace(0,10,270);
y2 = filter(Hd,ax);
plot(t,ax,t,y2)
xlabel('Time (s)')
ylabel('Amplitude')
legend('Original Signal','Filtered Data')
```



This filter nearly fits the original signal

Another entry is tested

```
Fs = 32;
t = linspace(0,10,594);
y2 = filter(Hd,VarName1);
plot(t,VarName1,t,y2)
xlabel('Time (s)')
ylabel('Amplitude')
legend('Original Signal','Filtered Data')
```



The filter returns less noisy data

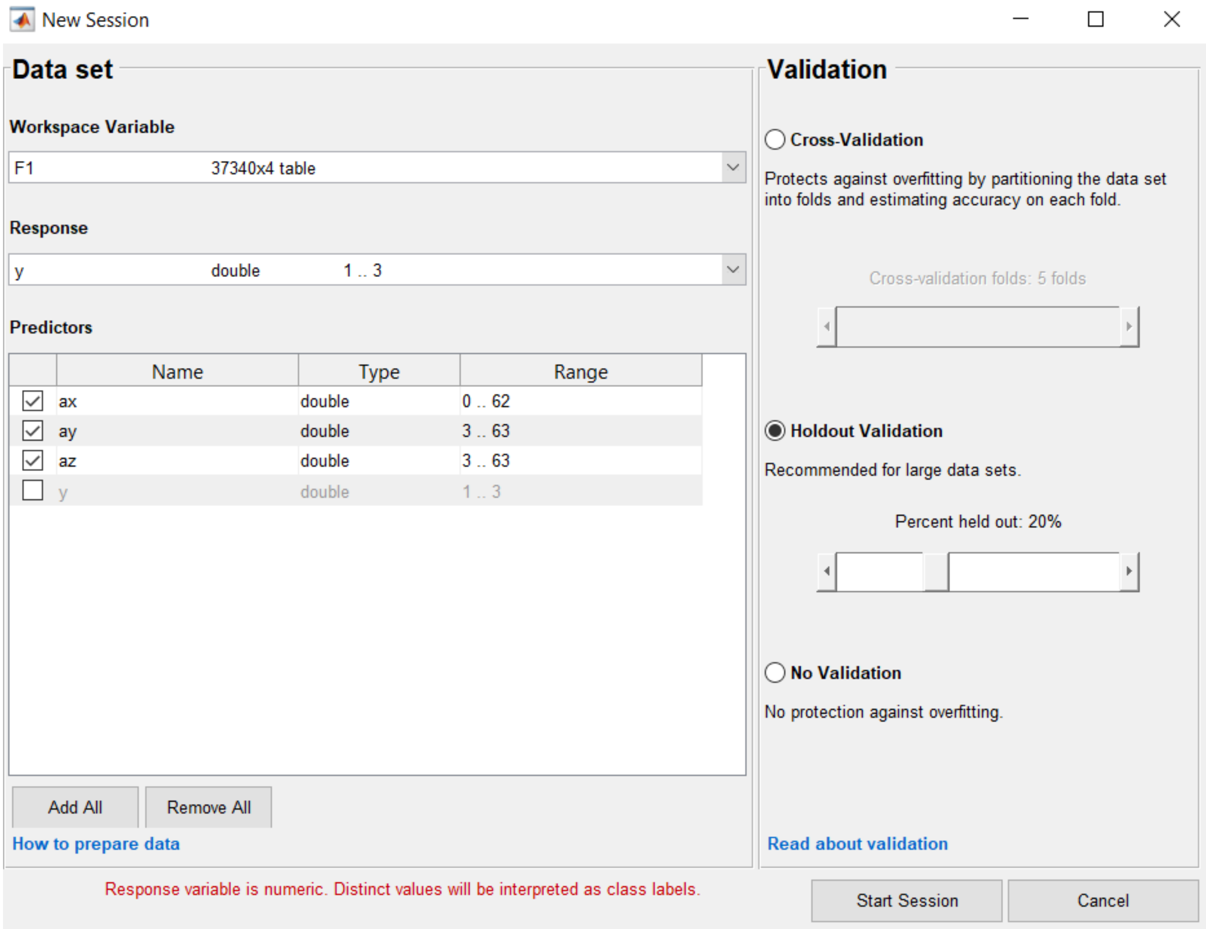
CLASSIFY and ANALYSIS

We first need to format our data such that it can be trained properly. Since we are using ML to help us differentiate between the 3 activities and we do not care about the timestamp of the data collected or the person we collected the data from. We combine all the recorded accelerometer readings with another output column “y” where the action performed is marked.

So, essentially, we have 3 columns of accelerometer readings and the corresponding action (climbing stairs, walking or descending stairs) of each reading is marked as either 1, 2 or 3 in the “y” column.

Now that our data is ready to be processed, we can perform the required ML.

We use Holdout validation to separate our train and test data. So 20% of our data is used as test data.



We now try training it using Decision Tree, SVM and KNN

Decision Tree

A fine tree is used which gives us an accuracy of 67.3%

Model 3: Trained

Results

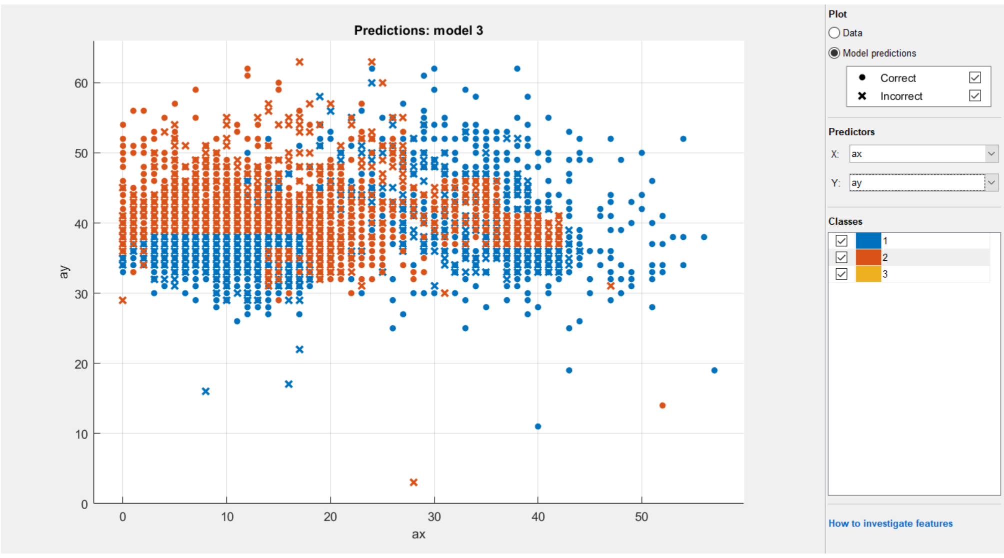
Accuracy	67.3%
Prediction speed	~1500000 obs/sec
Training time	0.71268 sec

Model Type
Preset: Fine Tree
Maximum number of splits: 100
Split criterion: Gini's diversity index
Surrogate decision splits: Off

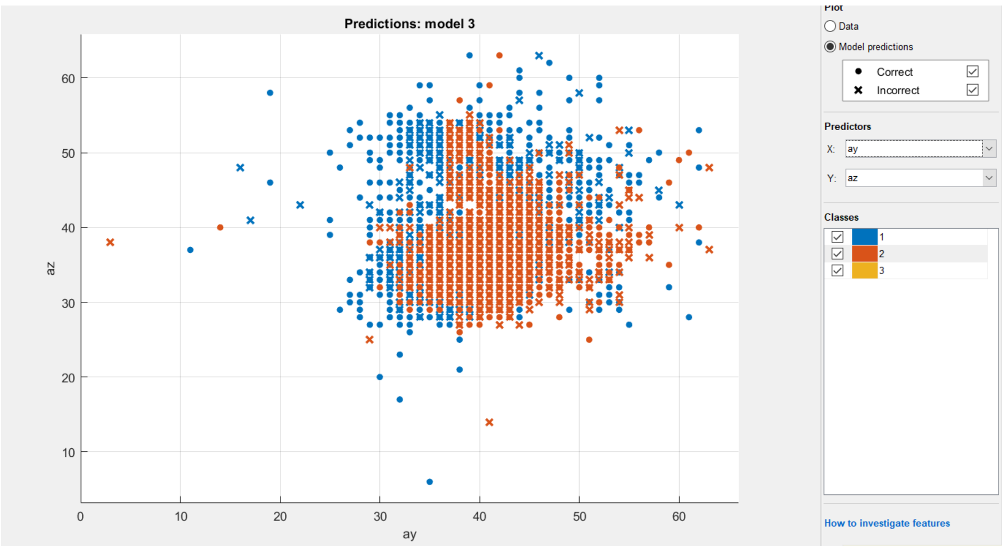
Feature Selection
All features used in the model, before PCA

PCA
PCA disabled

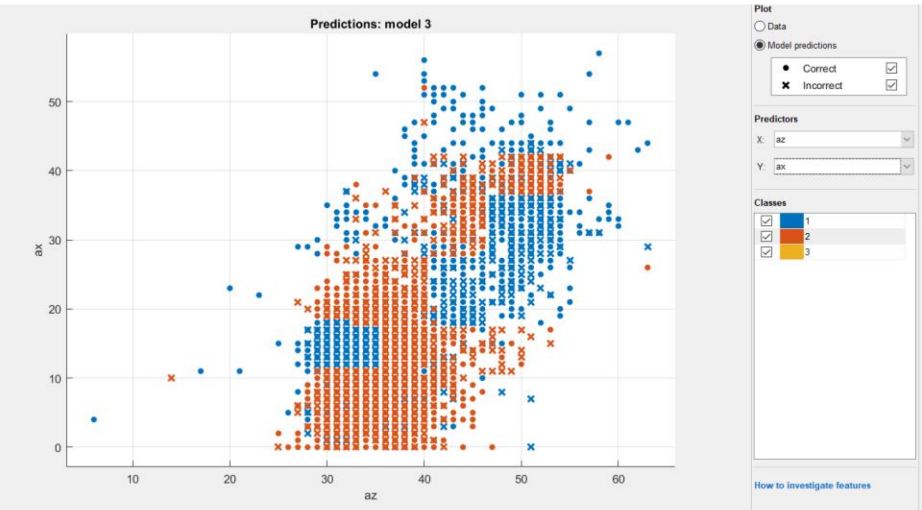
For ax and ay



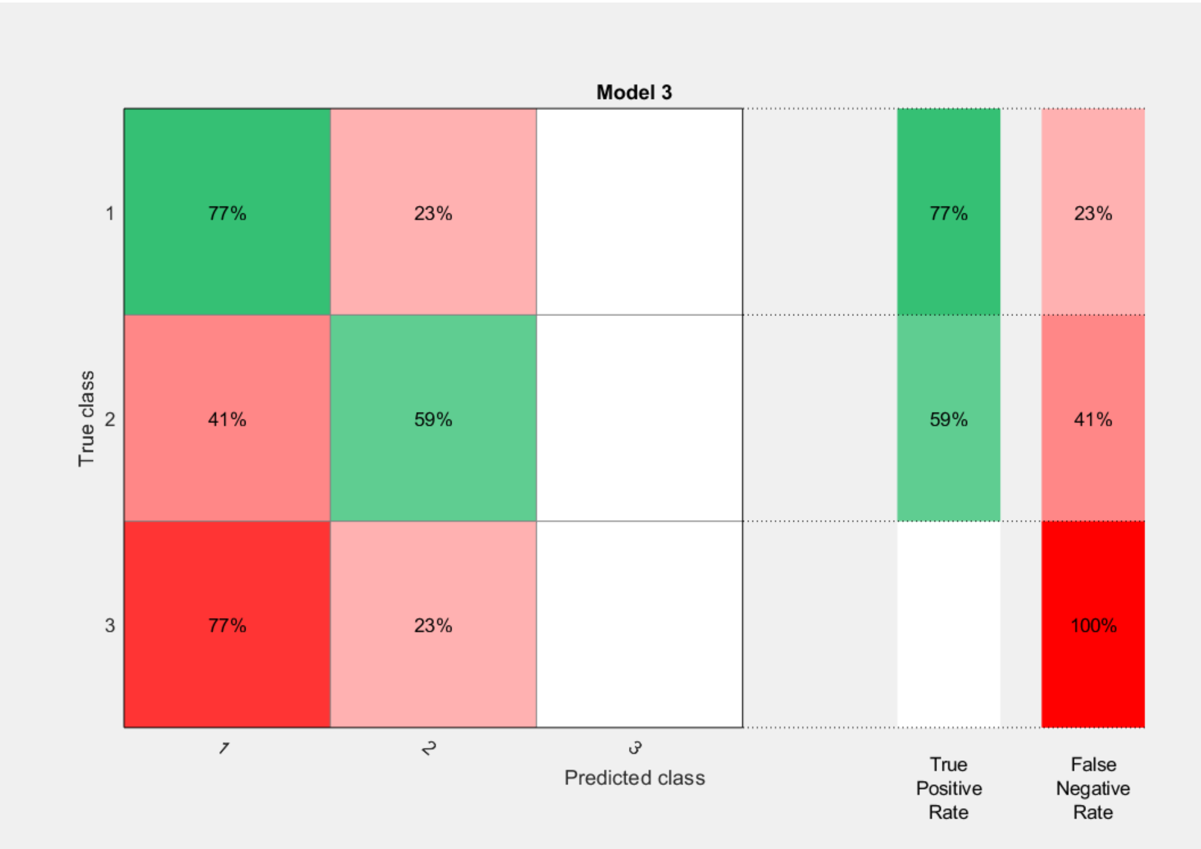
For ay and az



For az and ax



The confusion matrix is generated to test the accuracy of our model and obtain our true positive and false negative rates



SVM

A fine Gaussian SVM is used which gives us a 68.5% accuracy

Model 2: Trained

Results

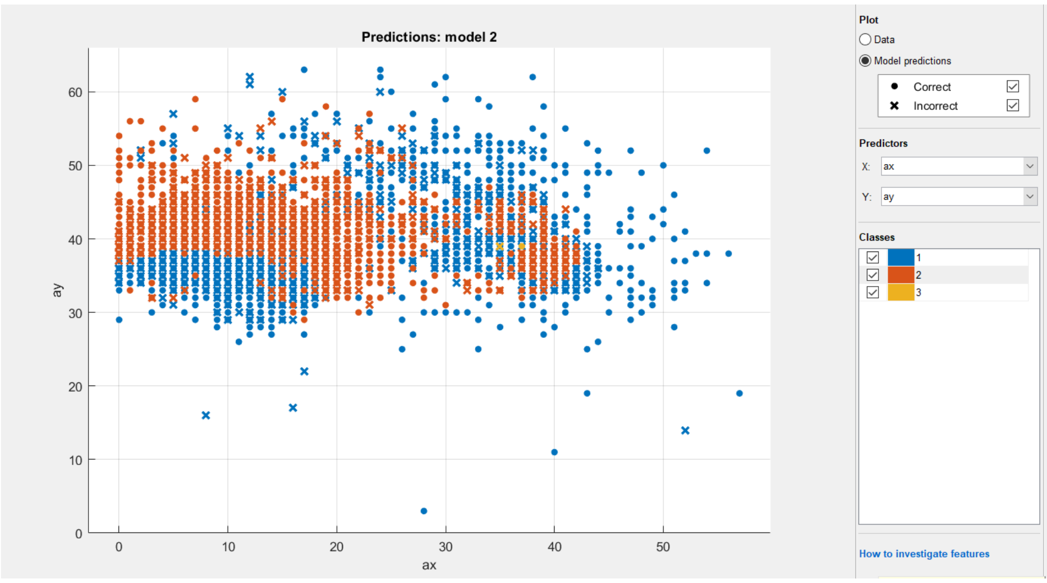
Accuracy	68.5%
Prediction speed	~3800 obs/sec
Training time	155.67 sec

Model Type
Preset: Fine Gaussian SVM
Kernel function: Gaussian
Kernel scale: 0.43
Box constraint level: 1
Multiclass method: One-vs-One
Standardize data: true

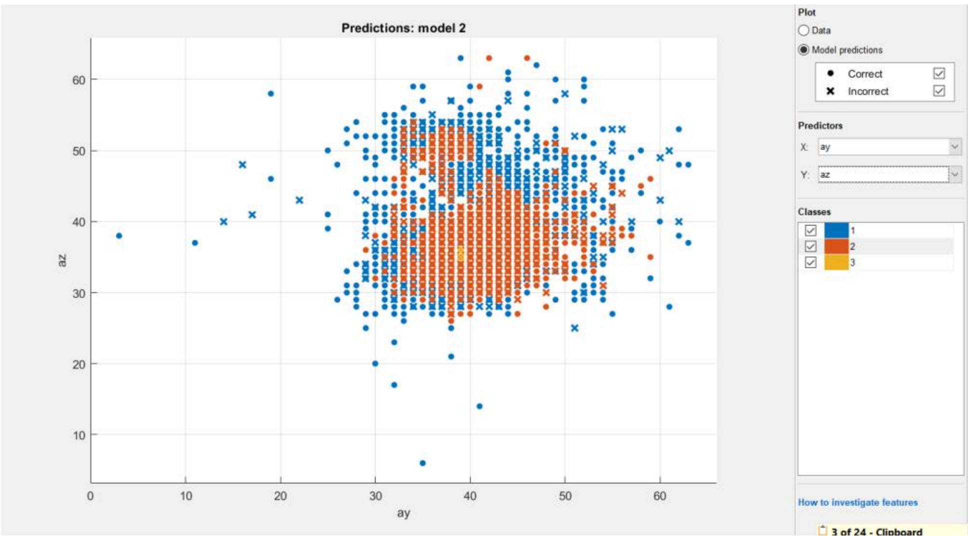
Feature Selection
All features used in the model, before PCA

PCA
PCA disabled

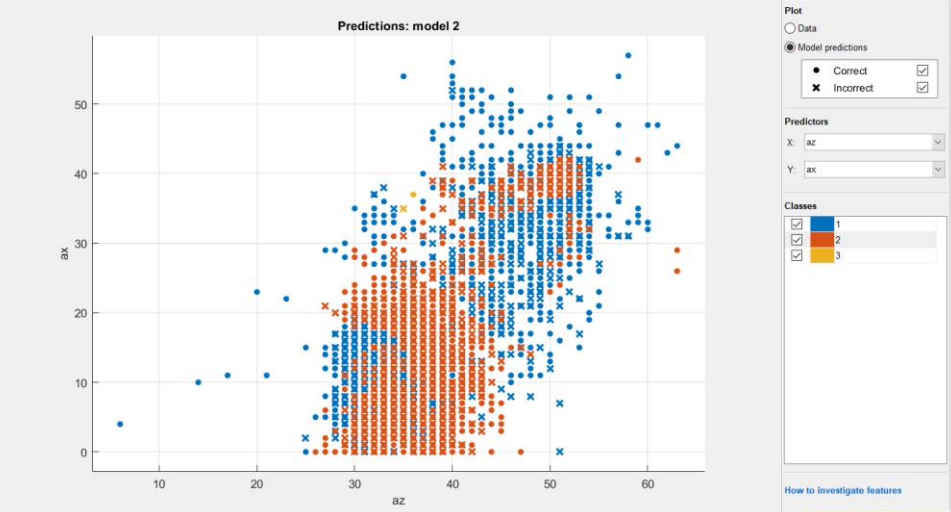
For ax and ay



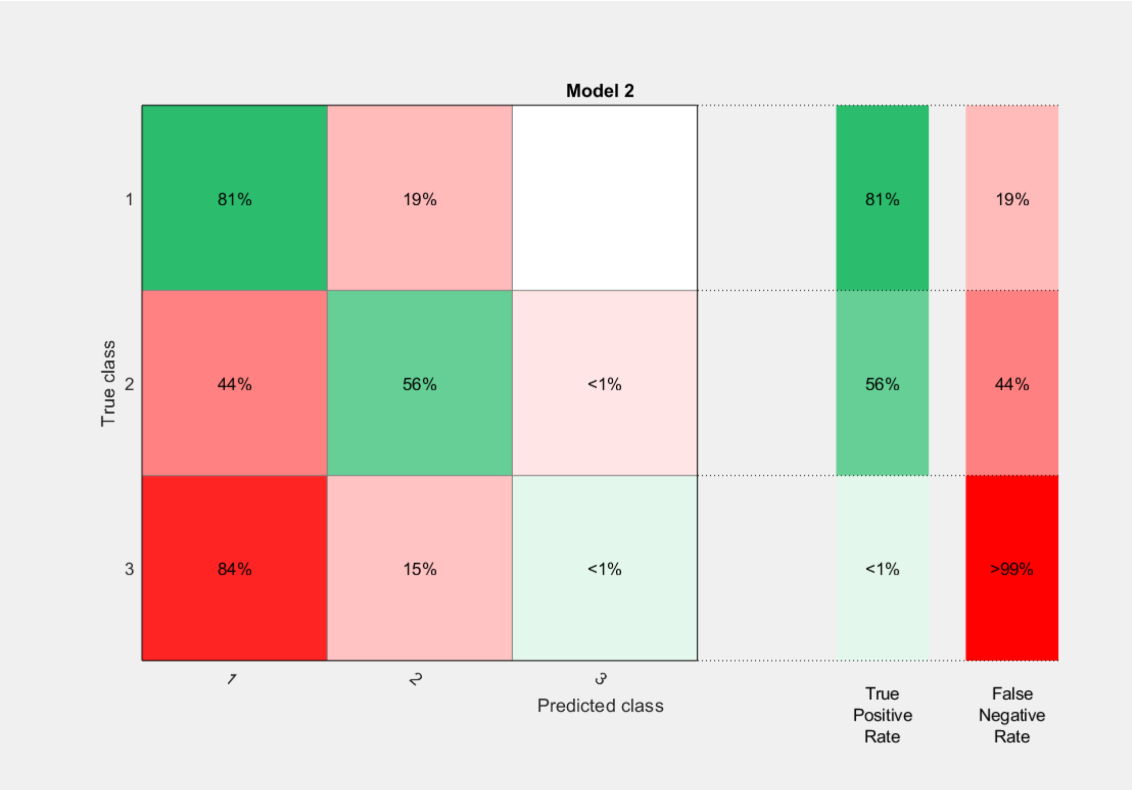
For ay and az



For az and ax



The confusion matrix is generated to test the accuracy of our model and obtain our true positive and false negative rates



KNN – K nearest neighbour is used which gives us an accuracy of 61.8%

Model 5: Trained

Results

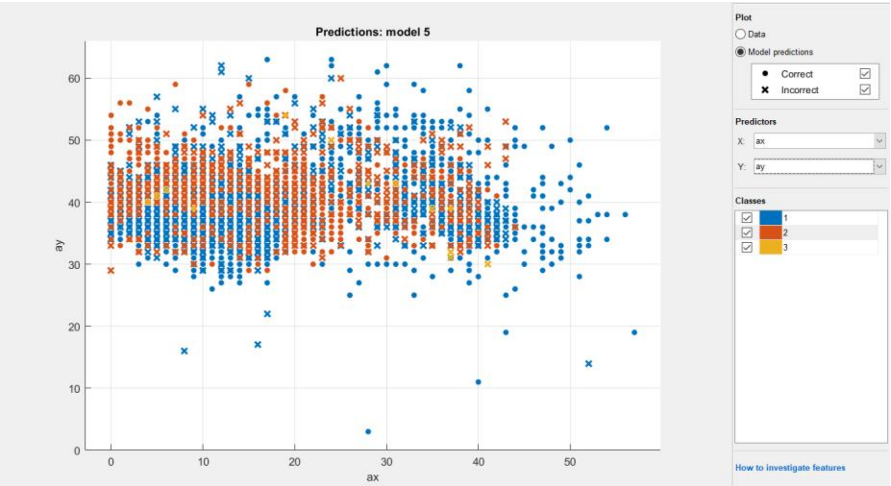
Accuracy	61.8%
Prediction speed	~110000 obs/sec
Training time	0.87905 sec

Model Type
Preset: Fine KNN
Number of neighbors: 1
Distance metric: Euclidean
Distance weight: Equal
Standardize data: true

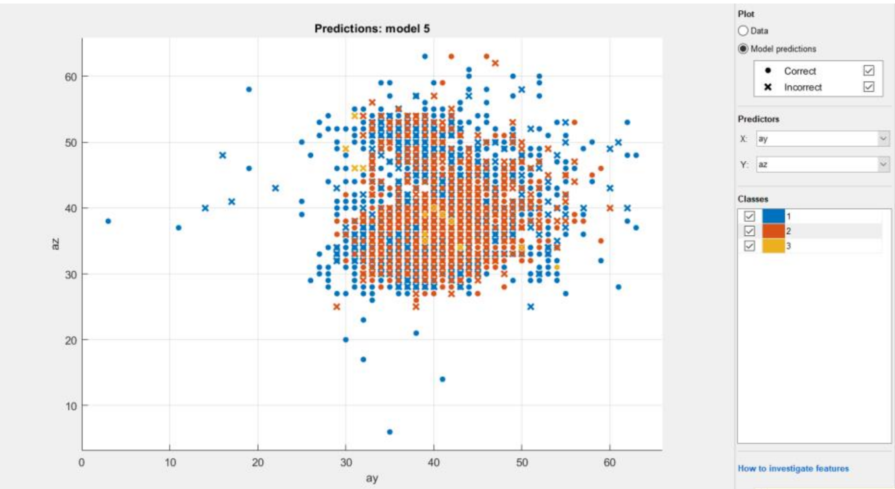
Feature Selection
All features used in the model, before PCA

PCA
PCA disabled

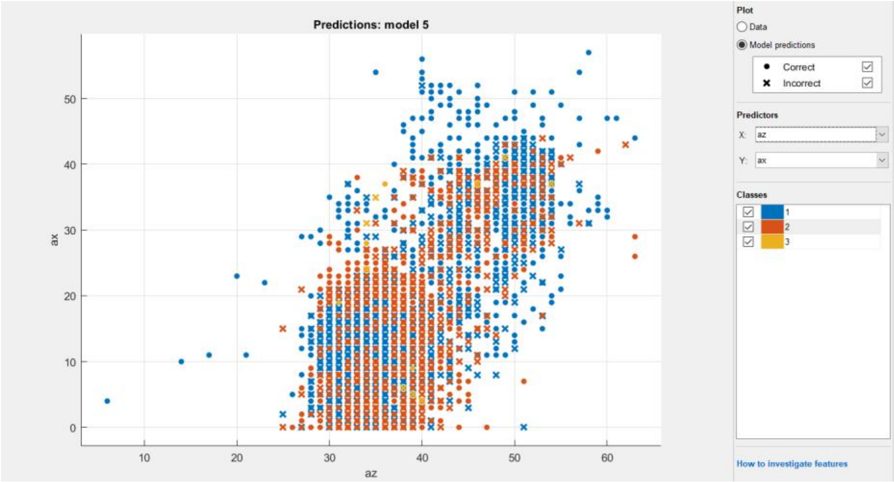
For ax and ay



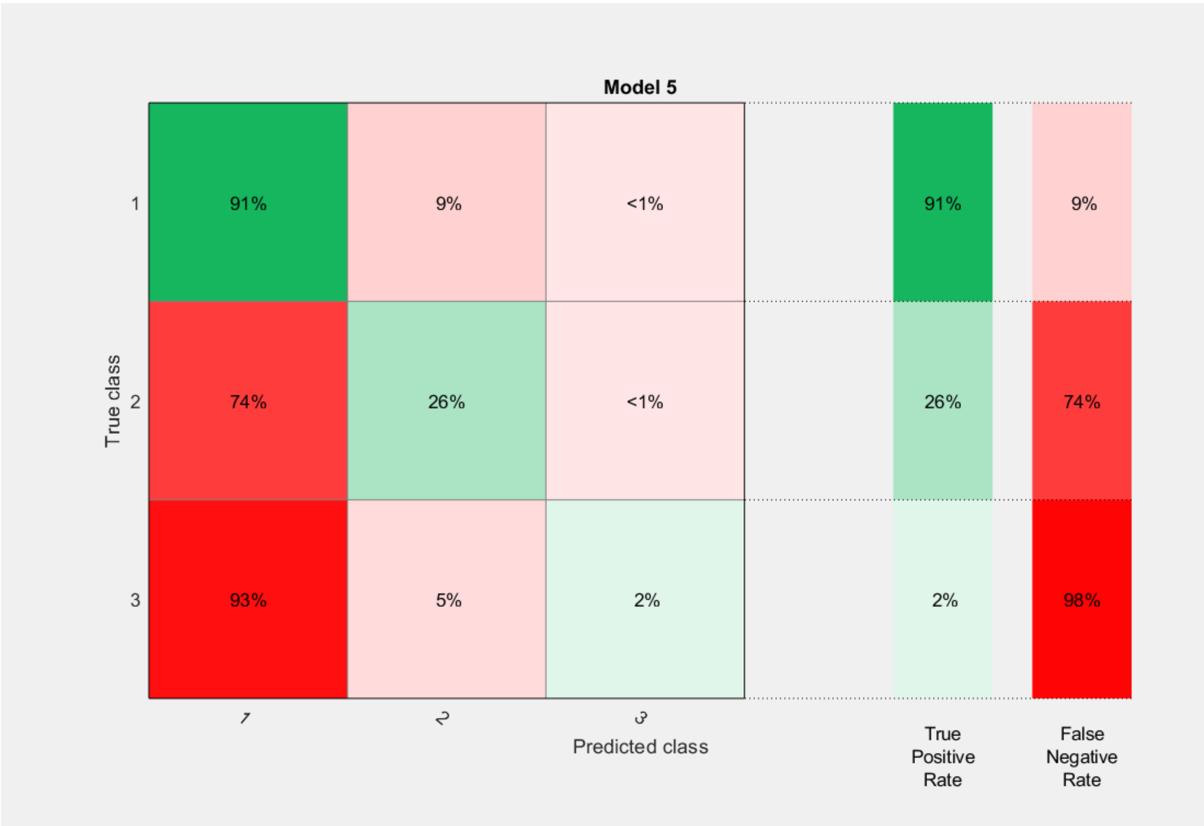
For ay and az



For az and ax



The confusion matrix is generated to test the accuracy of our model and obtain our true positive and false negative rates



From all 3 methods we observe that the model is able to predict the ax value very well but as other parameters of ay and az are added to it, the prediction resuly becomes worse and we eventually lose all our accuracy.

We know that there will be some error associated with every model that we use for predicting the true class of the target variable. This will result in False Positives and False Negatives. There's no hard rule that says what should be minimised and it depends solely on the task at hand. In some cases, having a false positive or a false negative can make a big difference. For e.g. Misdiagnosing someone with cancer or marking an important email as spam.

Here, having a false positive means that the correct accelerometer reading was wrongly detected to be the action. In essence, let's say the accelerometer reading corresponding to 2 was actually accepted as 1 by the model. A false positive error is a type I error where the test is checking a single condition, and wrongly gives an affirmative (positive) decision

A false negative means that the wrong accelerometer reading was detected to be the action. In essence, let's say the accelerometer reading corresponding to 1 was not marked as 1 by the model. A false negative error is a type II error occurring in a test where a single condition is checked for and the result of the test is erroneously that the condition is absent.

We cannot change the data and the classifier in order to shift the relative balance between false positive and false negative detections. But if we get more data, then we can develop a better model. And developing a better model means we get better accuracy and hence lesser false positives and false negatives.

When developing detection algorithms or tests, a balance must be chosen between risks of false negatives and false positives. Usually there is a threshold of how close a match to a given sample must be achieved before the algorithm reports a match. The higher this threshold, the more false negatives and the fewer false positives.

The classification is further analyzed on Python, the code for which is attached in a PDF titled 'code'