Introduction

The purpose of this assignment was to execute the SVM method for condition assessment of shafts using vibration data. Data was taken from a test involving a rotor-bearing testbed which was built to analyze the strength of shaft vibrations under varying conditions. Provided data samples were divided into test data and trial data, with the latter being used to train the algorithm.

Training data was either collected as:

- “healthy” with standard operating procedures and a balanced shaft
- “faulty_1” with a level 1 unbalanced load of one screw being added to the disk
- “faulty_2” with a level 2 unbalanced load of two screws being added to the disk

After data processing through SVM, test data was analyzed to determine if the vibrations experienced could be correctly categorized as identified in the assignment.

Given a rotation speed of 20 Hz and a sample collection rate of 2560 Hz, over 100 samples per rotation were recorded, allowing for distinct measurable differences between “healthy”, “faulty_1” and “faulty_2” data sets. The raw data was recorded by an accelerometer mounted on the bearing housing, with vibration amplitude recorded over 15 seconds for a total of 38400 samples per trial. Once all data had been compiled, key data points were extracted and extrapolated to demonstrate trends within each dataset. With four choices of kernel tricks to apply, the SVM model was to determine the level of imbalance for each test sample data recording, ranging from zero to two screws.

Implemented Solution

1. **Set file path:**

Specific to the user's machine, the libsvm/matlab folder location is declared.

2. **Sample Preparation:**

The data is sourced from text files located in three different directories: "Healthy," "Faulty," and "Testing."

The dir function is used to list all .txt files in each of the three directories separately.

- healthy_fileList contains the list of healthy training data files.
- faulty1_fileList contains the list of faulty (unbalanced by 1 screw) training files.
- faulty2_fileList contains the list of faulty (unbalanced by 2 screws) training files.
- test_fileList contains the list of testing data files.

Four matrices are initialized to hold the data: normal_data, faulty1_data, faulty2_data, and test_data. Each matrix is preallocated with dimensions 38400 (rows) by 20 (columns) for the training data, 38400 (rows) and 30 (columns) for the test data.

Two loops are used to read and transcribe data from the text files.

- For training data (healthy, faulty_1, and faulty_2):
  Each file is opened and its data is imported using the import_file_data function. The data from the files is transcribed into the respective matrices (normal_data and faulty_data) with 38400 rows (samples) and 20 columns (samples). Each column of these matrices represents one data sample.
- For test data:
  Another loop reads and transcribes data from the test files into the test_data matrix. The test data matrix has 38400 rows and 30 columns.

Through the script, data is sampled sequentially from the text files, with each file representing one data sample. The assumption is that each text file contains time series data with 38400 data points (samples) in each file. For training data, the code collects 20 samples from the "Healthy" directory, 20 samples from the "Faulty_1" directory, and 20 samples from the "Faulty_2" directory, placing each sample in a separate column of the corresponding matrix. For test data, the code collects 30 samples from the "Testing" directory and places each sample in a separate column of the test_data matrix.

3. **Feature extraction and selection:**

The Fast Fourier Transform (FFT) is applied three times on each sample in a loop. Prior to each FFT, a bandpass filter is applied on the signal at a specified upper and lower frequency that surrounds the known harmonic frequencies of approximately 20, 40, and 60 Hz with an offset of ±10 Hz.

Therefore, every iteration of the loop effectively conditions the signal for the FFT to yield the amplitudes of the first, second, and third harmonics as three frequency-domain features for use in the model.

To extract time-domain features, the raw data had too much noise to use as-is. This was particularly true for the healthy samples, where the buried noise signals' amplitudes could greatly exceed those of the true vibration waves.

Given that the third harmonic frequency is at approximately 60 Hz (visual observation showed it was actually closer to 63 Hz in most cases, since the first harmonic was often at 21 Hz), all higher frequencies could be attenuated with a lowpass filter set at 70 Hz to isolate the desired signal.

After conditioning the data with the lowpass filter, an envelope was generated at the upper and lower ends of each signal. For every sample, the mean value of the upper and lower envelope yielded two time-domain features for the model.

4. Plot features

To visualize the features being fed into the model, the following figures are generated by the script:

- Amplitudes of Healthy and Faulty Training Samples at 1X, 2X and 3X Harmonics
- Amplitudes of Test Samples at 1X Harmonic
    - Each colored region contains the samples that should be classified into their corresponding title. However, note that the *cutoff of each region is approximated*, and could be made more credible or adjusted with additional testing data sent through the model.
- Scatter Plots of 1X, 2X and 3X Harmonic Amplitudes vs. Filtered Envelope Mean Values
- Lowpass Filtered Time-Domain Signal Envelope Mean Values

5. SVM

The first step is to create feature matrices for the training and testing data (to be named "FeatMat_train" and "FeatMat_test" as pre-established by the provided example code).

With the feature matrices created, the "svmtrain" command trains the SVM with the entire training feature matrix and the user's choice of kernel trick. Each of the four kernel trick options is simply labeled with an integer, so the user can easily edit the value with subsequent runs of this section.

Finally, "svmpredict" executes the SVM process on the test feature matrix. The output in MATLAB shows the resultant accuracy value.

6. Confusion Matrix

With each classification labeled the same way as it was for the training data (1: healthy, 2: unbalanced with one screw, 3: unbalanced with two screws), the confusion matrix is ready to generate. It appears in the output with the title indicating the iteration number (see the Discussion section below for an explanation of the Iteration numbers), as well as the selected kernel trick.

Flow Diagram:

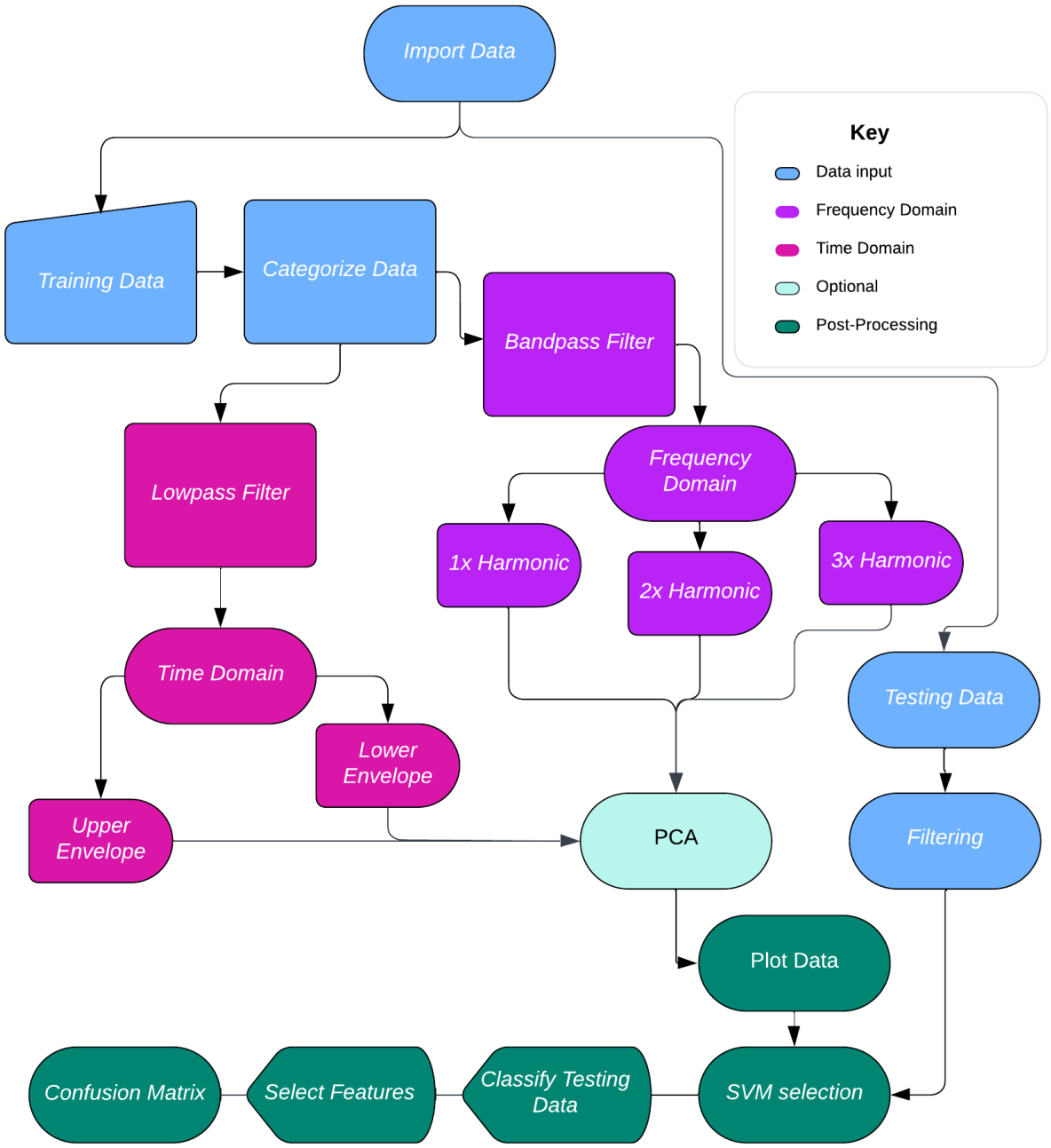


Figure 1: Flow Chart for Solution Process

Results

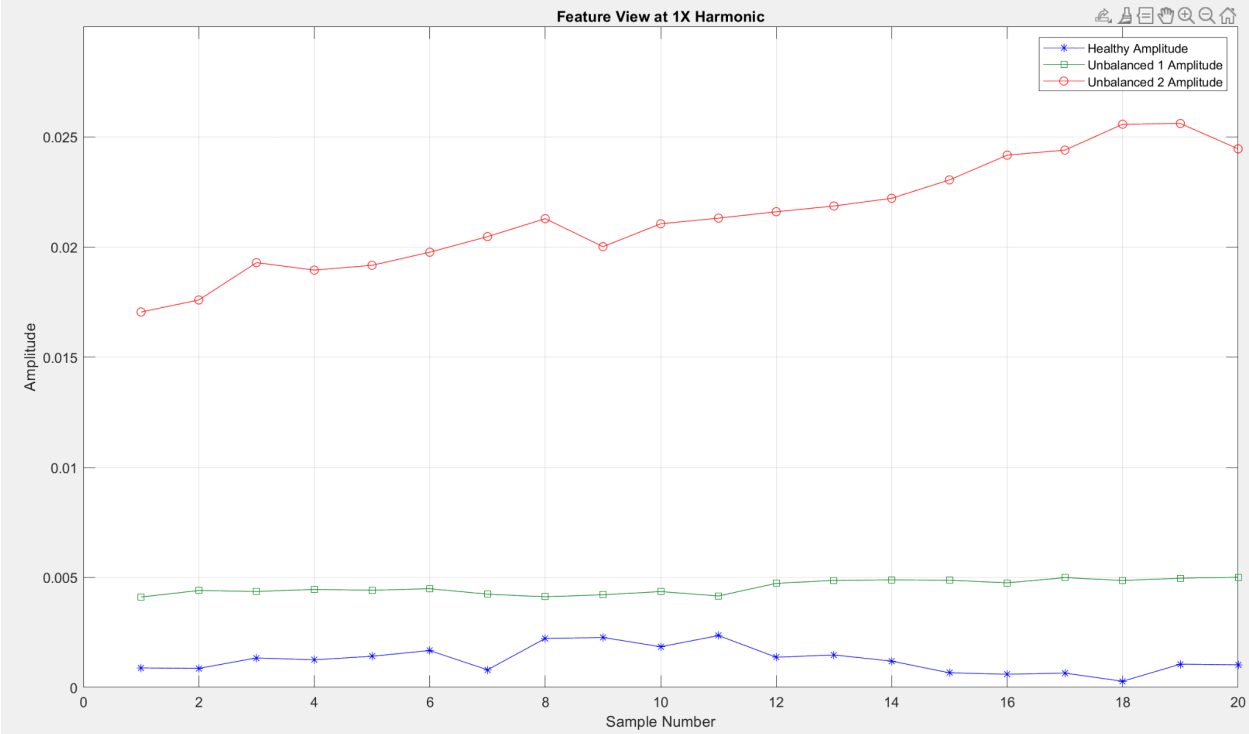


Figure 2: Feature View of Healthy and Faulty Training Samples at 1X Harmonic

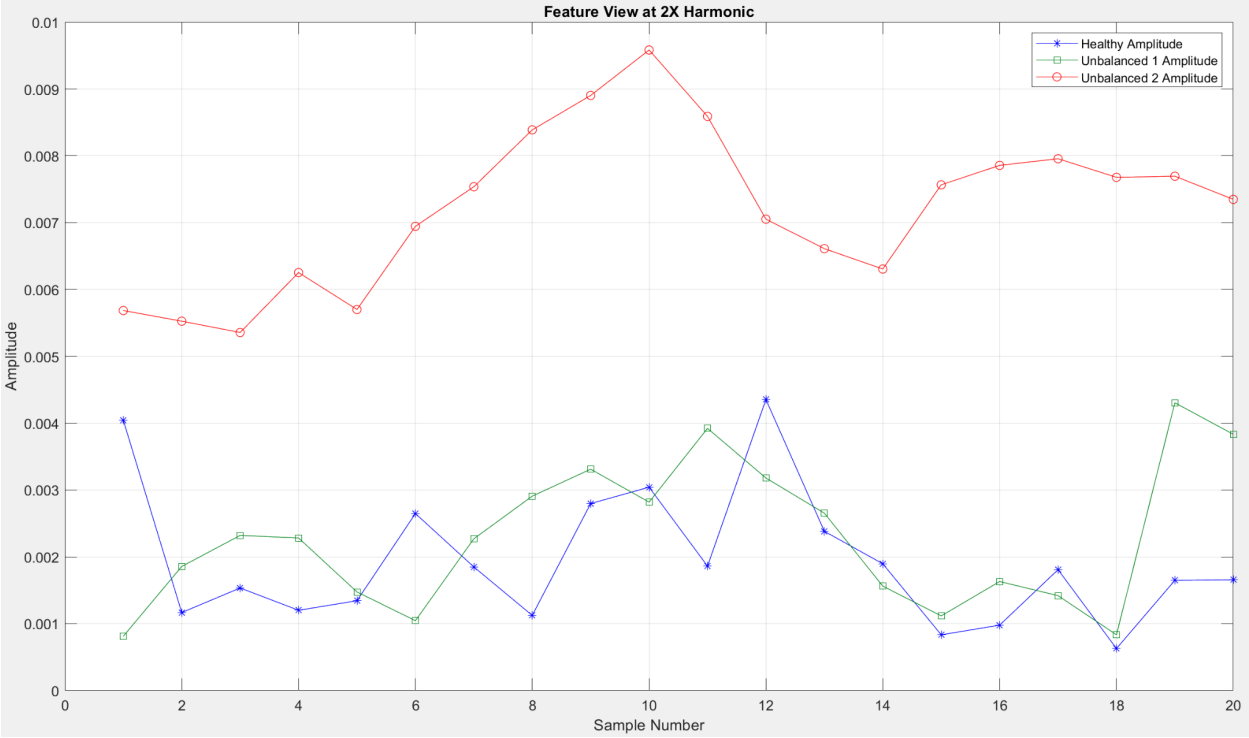


Figure 3: Feature View of Healthy and Faulty Training Samples at 2X Harmonic

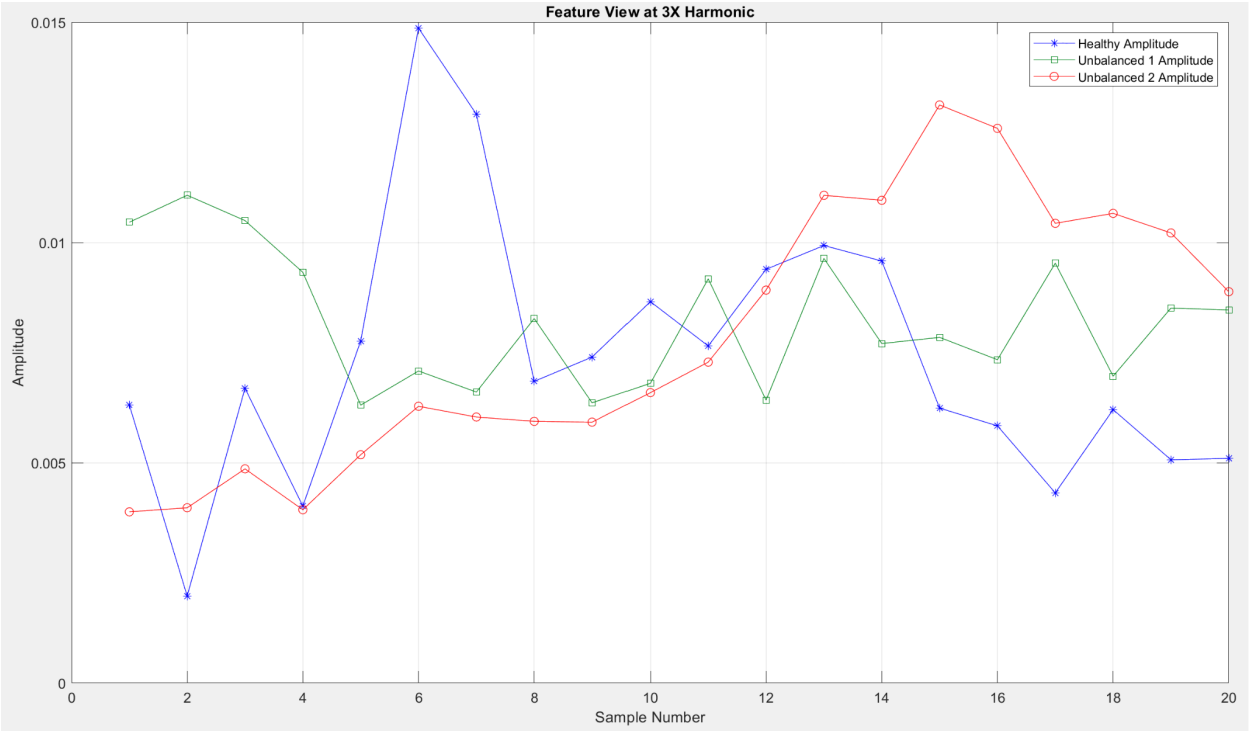


Figure 4: Feature View of Healthy and Faulty Training Samples at 3X Harmonic

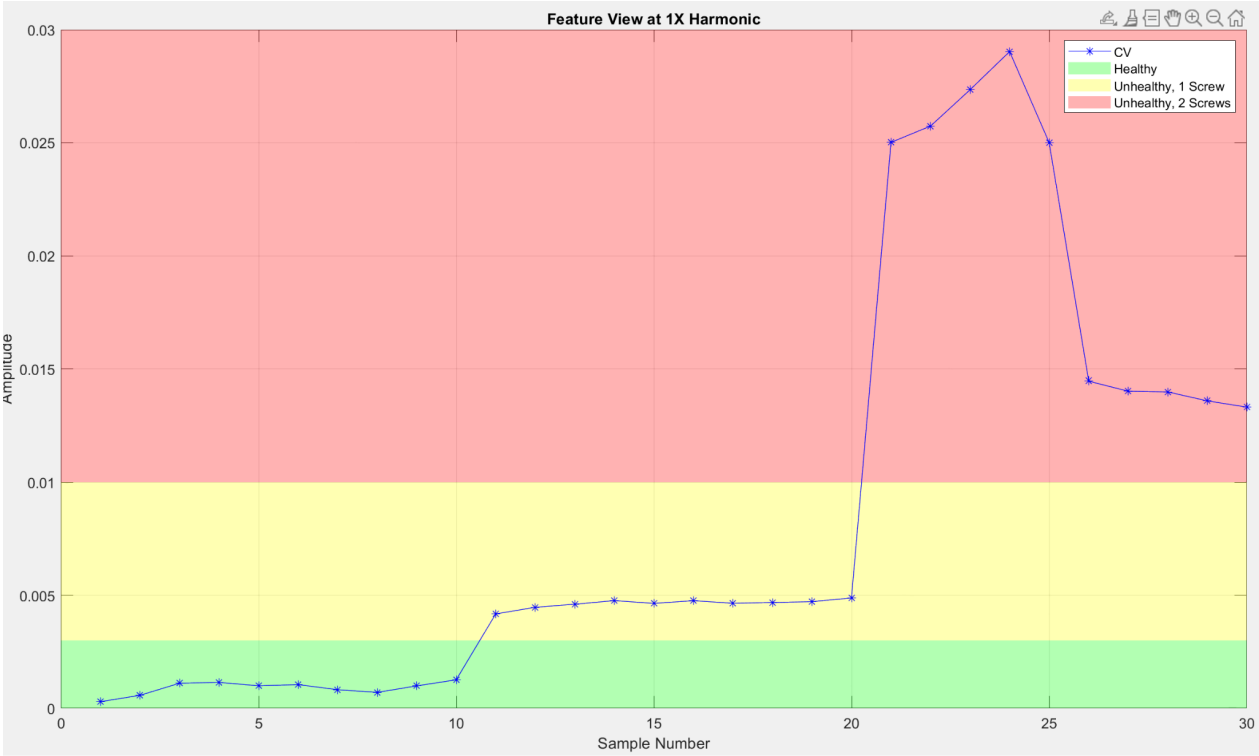


Figure 5: Feature View of Testing Samples at 1X Harmonic*

*Colored regions indicate where the model should classify each sample, for 100% accuracy. Cutoffs for each region are approximated.

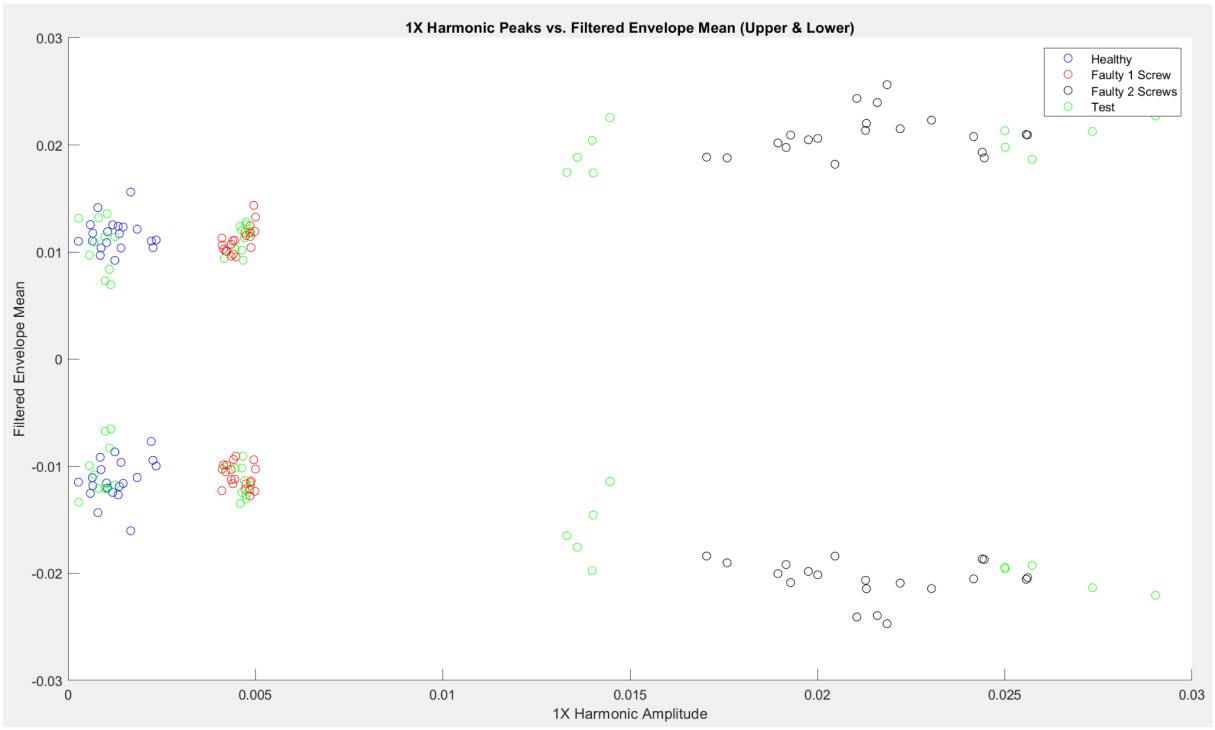


Figure 6: Scatter Plot of 1X Harmonic Amplitude vs. Filtered Envelope Mean

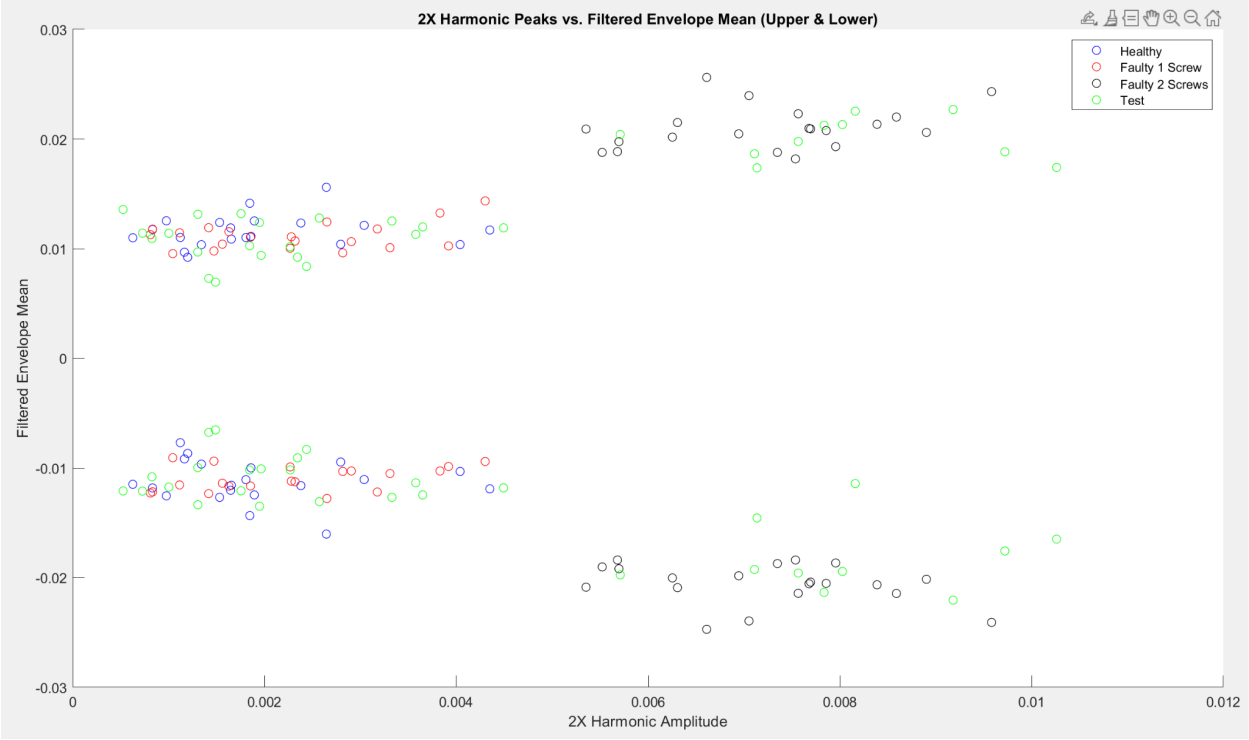


Figure 7: Scatter Plot of 2X Harmonic Amplitude vs. Filtered Envelope Mean

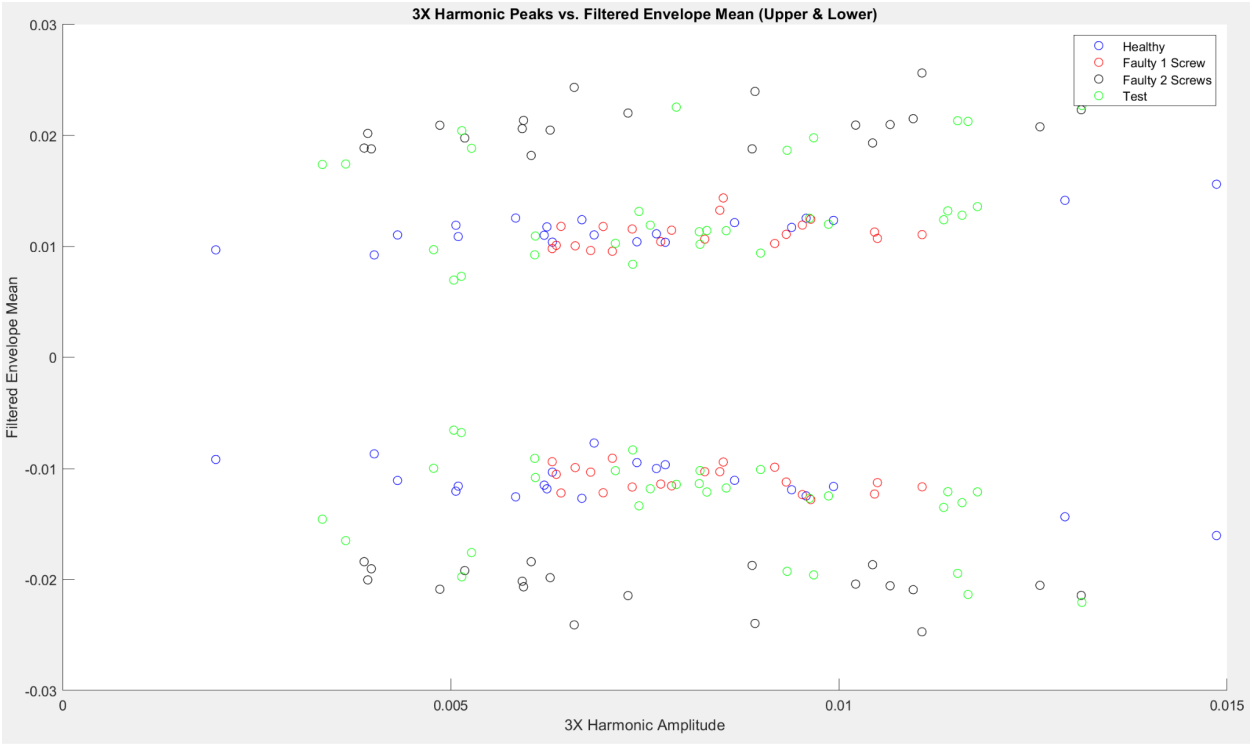


Figure 8: Scatter Plot of 3X Harmonic Amplitude vs. Filtered Envelope Mean

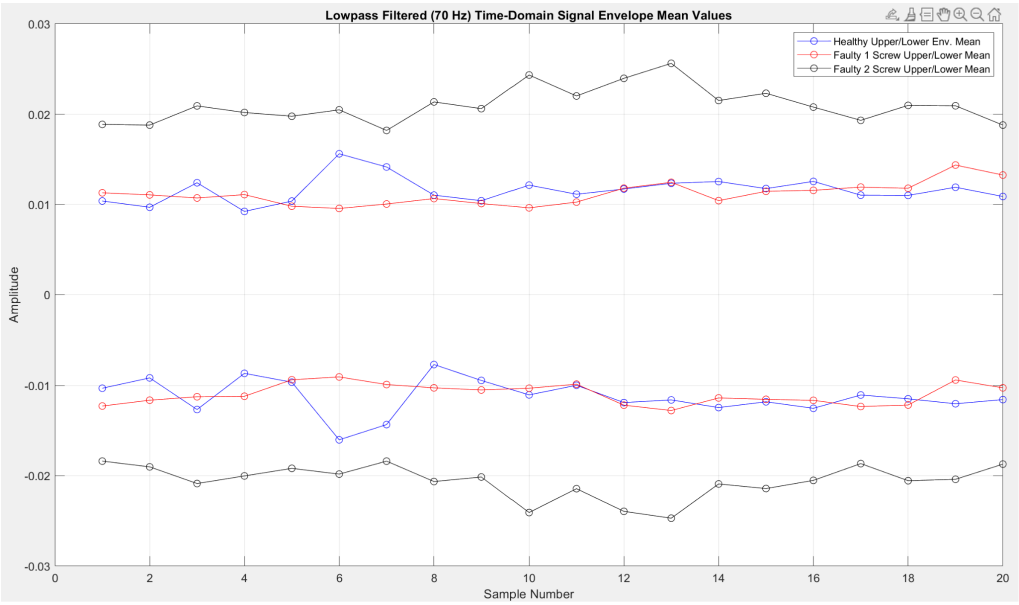
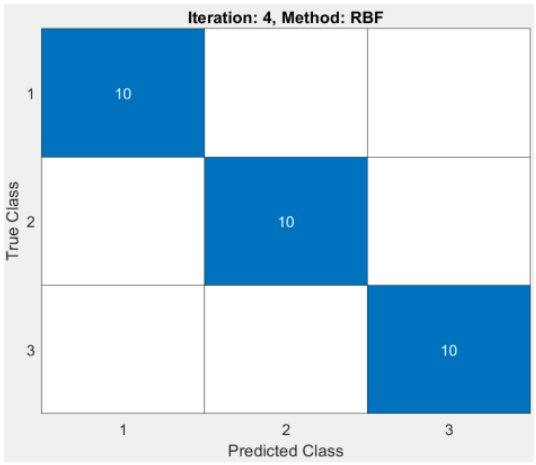
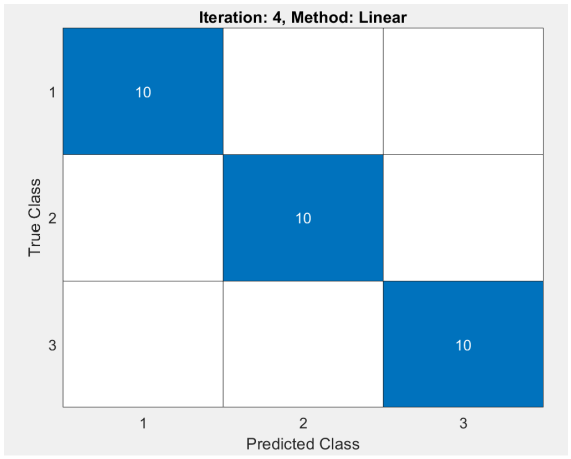
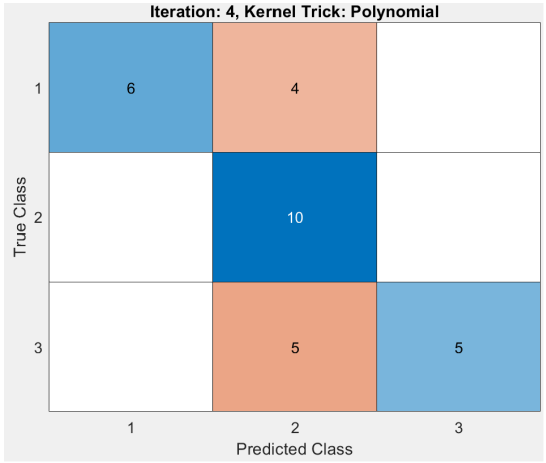


Figure 9: Lowpass Filtered Time-Domain Signal Envelope Mean Values







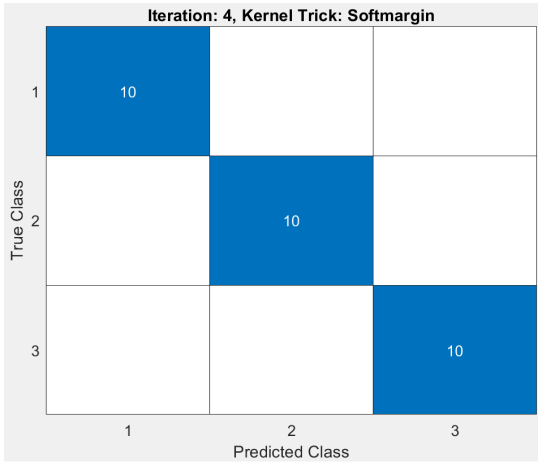


Figure 10: Confusion Matrix of all four Kernel Trick options, using all five feature values.

Discussion

The challenging part of this assignment was experimenting with different features to achieve 100% accuracy. During all these trials mentioned below, we iterated over all four kernel tricks provided in the script.

Features from Frequency domain alone:

- Iteration 1: First Harmonic from Frequency Domain
    - We first tried only in the frequency domain, considering the first harmonics as our feature. Using this approach, we were able to achieve an accuracy value of 83.33% for all four kernel tricks.

- Iteration 2: First, second and third harmonics from Frequency Domain
    - We introduced two more features from the Frequency domain i.e second harmonic and third harmonic. Now with these three features we trained our SVM and achieved the best accuracy of 96.67%.

Features from both frequency and time domain:

Now that we had already used the first three harmonic features from the frequency domain, we turned to time domain analysis for more features. As the raw time domain data was too noisy, we introduced a low pass filter at 70 Hz and applied an envelope to get upper and lower envelopes. Then the mean values of these upper and lower envelopes were calculated and fed into the model as our two time-domain features.

Iteration 3: First harmonic and Upper Envelope Mean:

As we have these five features, for this iteration we took one feature from each domain, i.e First Harmonic from Frequency Domain and Upper Envelope Mean from Time Domain. We lost accuracy drastically with this combination of features. The best accuracy during this iteration was 80%.

Iteration 4: All five features

After losing a considerable amount of accuracy in the third iteration, we tried training the model using all five extracted features. With this approach, we were able to achieve the best accuracy of 100% for three kernel tricks (RBF, Linear, and Sigmoid). However, the Polynomial kernel trick yielded only 70% accuracy.

Iteration #	Frequency Domain Features			Time Domain Features		Accuracy with Specific Kernel			
	1X	2X	3X	Upper Mean	Lower Mean	rbf	linear	Polynomial	Sigmoid
1	X					83.33	83.33	83.33	83.33
2	X	X	X			96.67	96.67	73.33	96.67
3	X			X		80	80	36.67	80
4	X	X	X	X	X	100	100	70	100

Figure 11: Table of Iterations and their Utilized Features & Accuracy Output

Conclusion

With the addition of one class in our training data compared to HW2, we found the need to extract features from the time domain - along with the frequency domain - to effectively train our SVM model. We were able to achieve the accuracy of 100% when we incorporated all five extracted features. While performing our iterations with different combinations of features, it became more clear that PCA could be of help in future assignments to find the optimal combination of features that would yield 100% accuracy, while reducing the total computational burden.

Appendix – Codes

HW3_SVM_Group6_20231001.m

```
%% 0: Clean up
clear; clc;
close all

%% 1: Set file path
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Specify the location of the libsvm/matlab folder %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dir_lib = ['C:\Users\jaemi\OneDrive\Documents\MATLAB\HW3\For
Students\libsvm\matlab\'];

%% 2: Import data
%%%%%%%%%%%%%%%%%%%%
% Write your code %
%%%%%%%%%%%%%%%%%%%%
% Make sure you change file path
healthy_fileList =
dir(fullfile("C:\Users\jaemi\OneDrive\Documents\MATLAB\HW3\For
Students\Training\Healthy\", "*.txt"));
faulty1_fileList =
dir(fullfile("C:\Users\jaemi\OneDrive\Documents\MATLAB\HW3\For
Students\Training\Faulty\Unbalance 1\", "*.txt"));
faulty2_fileList =
dir(fullfile("C:\Users\jaemi\OneDrive\Documents\MATLAB\HW3\For
Students\Training\Faulty\Unbalance 2\", "*.txt"));
test_fileList = dir(fullfile("C:\Users\jaemi\OneDrive\Documents\MATLAB\HW3\For
Students\Testing\", "*.txt"));
% Predefine a 38400 by 20 matrix for normal and faulty data
% 38400 per samples at 20 samples for each dataset
normal_data = zeros(38400, 20);
faulty1_data = zeros(38400, 20);
faulty2_data = zeros(38400, 20);
test_data = zeros(38400, 30);
% Declare an empty array to store the first peak amplitude value after FFT
normal_amplitude = zeros(20,3);
faulty1_amplitude = zeros(20,3);
faulty2_amplitude = zeros(20,3);
test_amplitude = zeros(30,3);
% read from line 6 onwards to until the end of the txt file
datalines = [6, Inf];
% Loop through and read each file for healthy and faulty data
for i = 1:20
healthy_fileName = "C:\Users\jaemi\OneDrive\Documents\MATLAB\HW3\For
Students\Training\Healthy\" + healthy_fileList(i,1).name;
```

```
bad1_fileName = "C:\Users\jaemi\OneDrive\Documents\MATLAB\HW3\For
Students\Training\Faulty\Unbalance 1\" + faulty1_fileList(i,1).name;
bad2_fileName = "C:\Users\jaemi\OneDrive\Documents\MATLAB\HW3\For
Students\Training\Faulty\Unbalance 2\" + faulty2_fileList(i,1).name;
% Import the file data
healthy_data = import_file_data(healthy_fileName, datalines);
bad1_data = import_file_data(bad1_fileName, datalines);
bad2_data = import_file_data(bad2_fileName, datalines);
% Transcribe it to matrix
for j = 1:38400
normal_data(j,i) = healthy_data(j,1);
faulty1_data(j,i) = bad1_data(j,1);
faulty2_data(j,i) = bad2_data(j,1);
end
end
% Loop through and read each file for test data
for i = 1:30
test_fileName = "C:\Users\jaemi\OneDrive\Documents\MATLAB\HW3\For
Students\Testing\" + test_fileList(i,1).name;
% Import the file data
temp_test_data = import_file_data(test_fileName, datalines);
% Transcribe it to matrix
for j = 1:38400
test_data(j,i) = temp_test_data(j,1);
end
end

%% 3: Feature extraction / FFT
%%%%%%%%%%%%%%%%%%%
% Write your code %
%%%%%%%%%%%%%%%%%%%
Fs = 2560; % Sampling frequency
T = 1/Fs; % Sampling period
L = 38400; % Length of signal
f = Fs/L*(0:(L/2-1));
frequency_bins = [10 30; 30 50; 50 70];
% Loop through each sample-set and perform FFT and record first peak
for i = 1:20
% Load healthy dataset values
X = normal_data(:,i);
for j = 1:3
X2 = bandpass(X,frequency_bins(j,:),Fs);
% Perform FFT for Single-Sided Amplitude Spectrum
Y = fft(X2);
P2 = abs(Y/L);
P1 = P2(1:L/2);
P1(2:end-1) = 2*P1(2:end-1);
% Record the first harmonic peak
normal_amplitude(i,j) = max(P1);
```

```
end
end
% Repeat but for faulty Unbalance 1 dataset
for i = 1:20
Fs = 2560; % Sampling frequency
T = 1/Fs; % Sampling period
L = 38400; % Length of signal
X = faulty1_data(:,i);
for j = 1:3
X2 = bandpass(X,frequency_bins(j,:),Fs);
Y = fft(X2);
P2 = abs(Y/L);
P1 = P2(1:L/2);
P1(2:end-1) = 2*P1(2:end-1);
faulty1_amplitude(i,j) = max(P1);
end
end
% Repeat but for faulty Unbalance 2 dataset
for i = 1:20
X = faulty2_data(:,i);
for j = 1:3
X2 = bandpass(X,frequency_bins(j,:),Fs);
Y = fft(X2);
P2 = abs(Y/L);
P1 = P2(1:L/2);
P1(2:end-1) = 2*P1(2:end-1);
faulty2_amplitude(i,j) = max(P1);
end
end
% Repeat but for testing dataset
for i = 1:30
X = test_data(:,i);
for j = 1:3
X2 = bandpass(X,frequency_bins(j,:),Fs);
Y = fft(X2);
P2 = abs(Y/L);
P1 = P2(1:L/2);
P1(2:end-1) = 2*P1(2:end-1);
test_amplitude(i,j) = max(P1);
end
end
% Time-Domain Features: Mean Values from Upper & Lower Envelopes
% Raw signal is conditioned with a lowpass filter at 70 Hz.
mean_normal = zeros(20,2);
mean_faulty2 = zeros(20,2);
mean_faulty1 = zeros(20,2);
mean_test = zeros(30,2);
fpass = 70;
interval = 10;
```

```
for i = 1:20
X = lowpass(normal_data(:,i), fpass, Fs);
[upper, lower] = envelope(X, interval, 'peak');
mean_normal(i,1) = mean(upper);
mean_normal(i,2) = mean(lower);
end
for i = 1:20
X = lowpass(faulty1_data(:,i), fpass, Fs);
[upper, lower] = envelope(X, interval, 'peak');
mean_faulty1(i,1) = mean(upper);
mean_faulty1(i,2) = mean(lower);
end
for i = 1:20
X = lowpass(faulty2_data(:,i), fpass, Fs);
[upper, lower] = envelope(X, interval, 'peak');
mean_faulty2(i,1) = mean(upper);
mean_faulty2(i,2) = mean(lower);
end
for i = 1:30
X = lowpass(test_data(:,i), fpass, Fs);
[upper, lower] = envelope(X, interval, 'peak');
mean_test(i,1) = mean(upper);
mean_test(i,2) = mean(lower);
End

%% 4: Plot signals, features
%%%%%%%%%%%%%%%%%%%%
% Write your code %
%%%%%%%%%%%%%%%%%%%%
% % Plotting Frequency-Based Features
% figure
% plot(normal_amplitude(:,1), '-b*')
% grid on
% title('Feature View at 1X Harmonic');
% xlabel('Sample Number');
% ylabel('Amplitude');
% hold on
% plot(faulty1_amplitude(:,1), '-square', 'Color', "#00841a")
% plot(faulty2_amplitude(:,1), '-ro')
% hold off
% legend('Healthy Amplitude', 'Unbalanced 1 Amplitude', 'Unbalanced 2
Amplitude');
% figure
% plot(normal_amplitude(:,2), '-b*')
% grid on
% title('Feature View at 2X Harmonic');
% xlabel('Sample Number');
% ylabel('Amplitude');
% hold on
```

```
% plot(faulty1_amplitude(:,2), '-square', 'Color', "#00841a")
% plot(faulty2_amplitude(:,2), '-ro')
% hold off
% legend('Healthy Amplitude', 'Unbalanced 1 Amplitude', 'Unbalanced 2
Amplitude');
%
% figure
% plot(normal_amplitude(:,3), '-b*')
% grid on
% title('Feature View at 3X Harmonic');
% xlabel('Sample Number');
% ylabel('Amplitude');
% hold on
% plot(faulty1_amplitude(:,3), '-square', 'Color', "#00841a")
% plot(faulty2_amplitude(:,3), '-ro')
% hold off
% legend('Healthy Amplitude', 'Unbalanced 1 Amplitude', 'Unbalanced 2
Amplitude');
% % Plotting Test Data 1X Harmonic Amplitudes
% figure
% plot(test_amplitude(:,1), '-b*')
% grid on
% title('Feature View at 1X Harmonic');
% xlabel('Sample Number');
% ylabel('Amplitude');
% hold on
% legend('Test Amplitude');
% y1 = yregion(0, 0.003, FaceColor="g", DisplayName="Healthy")
% y2 = yregion(0.003, 0.01, FaceColor="y", DisplayName="Unhealthy, 1 Screw")
% y3 = yregion(0.01, 0.03, FaceColor="r", DisplayName="Unhealthy, 2 Screws")
% legend
% hold off
% % Scatter Plot of 1X Harmonic Amplitude vs. Filtered Envelope Mean
% figure
% scatter(normal_amplitude(:,1), mean_normal, 'bo')
% hold on
% scatter(faulty1_amplitude(:,1), mean_faulty1, 'ro')
% scatter(faulty2_amplitude(:,1), mean_faulty2, 'ko')
% scatter(test_amplitude(:,1), mean_test, 'go')
% title('1X Harmonic Peaks vs. Filtered Envelope Mean (Upper & Lower)');
% xlabel('1X Harmonic Amplitude');
% ylabel('Filtered Envelope Mean');
% legend('Healthy', '', 'Faulty 1 Screw', '', 'Faulty 2 Screws', '', 'Test');
% hold off
%
% % Scatter Plot of 2X Harmonic Amplitude vs. Filtered Envelope Mean
% figure
% scatter(normal_amplitude(:,2), mean_normal, 'bo')
% hold on
```

```
% scatter(faulty1_amplitude(:,2), mean_faulty1, 'ro')
% scatter(faulty2_amplitude(:,2), mean_faulty2, 'ko')
% scatter(test_amplitude(:,2), mean_test, 'go')
% title('2X Harmonic Peaks vs. Filtered Envelope Mean (Upper & Lower)');
% xlabel('2X Harmonic Amplitude');
% ylabel('Filtered Envelope Mean');
% legend('Healthy', '', 'Faulty 1 Screw', '', 'Faulty 2 Screws', '', 'Test');
% hold off
% % Scatter Plot of 3X Harmonic Amplitude vs. Filtered Envelope Mean
% figure
% scatter(normal_amplitude(:,3), mean_normal, 'bo')
% hold on
% scatter(faulty1_amplitude(:,3), mean_faulty1, 'ro')
% scatter(faulty2_amplitude(:,3), mean_faulty2, 'ko')
% scatter(test_amplitude(:,3), mean_test, 'go')
% title('3X Harmonic Peaks vs. Filtered Envelope Mean (Upper & Lower)');
% xlabel('3X Harmonic Amplitude');
% ylabel('Filtered Envelope Mean');
% legend('Healthy', '', 'Faulty 1 Screw', '', 'Faulty 2 Screws', '', 'Test');
% hold off
% % Lowpass Filtered (70 Hz) Time-Domain Signal Envelope Mean Values
% figure
% plot(mean_normal, '-bo')
% grid on
% hold on
% plot(mean_faulty1, '-ro')
% plot(mean_faulty2, '-ko')
% title('Lowpass Filtered (70 Hz) Time-Domain Signal Envelope Mean Values');
% xlabel('Sample Number');
% ylabel('Amplitude');
% legend('Healthy Upper/Lower Env. Mean', '', ...
% 'Faulty 1 Screw Upper/Lower Mean', '', ...
% 'Faulty 2 Screw Upper/Lower Mean', '');
% hold off

%% 5. SVM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Before this section, you need prepare
% - FeatMat_train: Feature matrix for training data
% - FeatMat_test : Feature matrix for testing data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
normal_features = [normal_amplitude(:,(1:2)) mean_normal];
faulty1_features = [faulty1_amplitude(:,(1:2)) mean_faulty1];
faulty2_features = [faulty2_amplitude(:,(1:2)) mean_faulty2];
FeatMat_train = [normal_features; faulty1_features; faulty2_features];
FeatMat_test = [test_amplitude(:,(1:2)) mean_test];
cd(dir_lib)
N_train = 60;
Nh = 20; % number of healthy
```

```
Nf1 = 20; % number of faulty1
% Training data
Train_X = FeatMat_train;
Train_Y = zeros(N_train,1);
Train_Y(1:Nh,1) = 1; % 1 means healthy
Train_Y(Nh+1:Nh+Nf1,1) = 2; % 2 means unbalanced 1 screw
Train_Y(Nh+Nf1+1:N_train,1) = 3; % 3 means unbalanced 2 screw
% Test Data
Test_X = FeatMat_test;
Test_Y = zeros(30,1);
Test_Y( 1:10,1) = 1;
Test_Y(11:20,1) = 2;
Test_Y(21:30,1) = 3;
% train SVM with different kernel
Method_list = {'rbf','linear','polynomial','Sigmoid'}; % kernel function
selection
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Here you can select kernel function
% Try different kernel and check the results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Method = Method_list{4}; % 1: rbf, 2: linear, 3: polynomial, 4: softmargin
switch Method
case 'rbf'
svmStruct = svmtrain(Train_Y,Train_X,'-s 0 -t 2 -g 0.333 -c 1');
% refer to README file in libsvm for more infomation
case 'linear'
svmStruct = svmtrain(Train_Y,Train_X,'-s 0 -t 0 -g 0.333 ');
case 'polynomial'
svmStruct = svmtrain(Train_Y,Train_X,'-s 0 -t 1 -g 0.333 ');
case 'Sigmoid'
svmStruct = svmtrain(Train_Y,Train_X,'-s 0 -t 3 -g 0.333 ');
end
% Test and predict label
% use trained SVM model for classification
[predicted_result, accuracy,~] = svmpredict(Test_Y,Test_X,svmStruct);

%% 6. Confusion Matrix
%%%%%%%%%%%%%%%%%%%%
% Write your code %
%%%%%%%%%%%%%%%%%%%%
class1 = 1 * ones(10,1);
class2 = 2 * ones(10,1);
class3 = 3 * ones(10,1);
all_classes = [class1; class2; class3];
g1 = all_classes'; % Known groups
g2 = predicted_result';  % Predicted groups
C = confusionmat(g1,g2); % Generate Confusion Matrix with Title
confusionchart(C)
if strcmp(Method, Method_list{1})
```

```
title_str = ('Iteration: 4, Kernel Trick: RBF')
elseif strcmp(Method, Method_list{2})
title ('Iteration: 4, Kernel Trick: Linear')
elseif strcmp(Method, Method_list{3})
title ('Iteration: 4, Kernel Trick: Polynomial')
elseif strcmp(Method, Method_list{4})
title ('Iteration: 4, Kernel Trick: Softmargin')
end
% plot(predicted_result)
```

Import_file_data.m

```
function filedata = import_file_data(filename, dataLines)
%IMPORTFILE Import data from a text file
% NORMALDATAMAR2614TIME15221 = IMPORTFILE(FILENAME) reads data from
% text file FILENAME for the default selection. Returns the data as a
% table.
%
% NORMALDATAMAR2614TIME15221 = IMPORTFILE(FILE, DATALINES) reads data
% for the specified row interval(s) of text file FILENAME. Specify
% DATALINES as a positive scalar integer or a N-by-2 array of positive
% scalar integers for dis-contiguous row intervals.
%
% Example:
% NormalDataMar2614Time15221 =
importfile("/Users/user/Documents/UMD2023/ENME485/Assignments/HW2/Reading
Materials/Training/Healthy/Normal Data Mar-26-14 Time 1522-1.txt", [6, Inf]);
%
% See also READTABLE.
%
% Auto-generated by MATLAB on 20-Sep-2023 14:03:45
%% Input handling
% If dataLines is not specified, define defaults
if nargin < 2
dataLines = [6, Inf];
end
%% Set up the Import Options and import the data
opts = delimitedTextImportOptions("NumVariables", 1);
% Specify range and delimiter
opts.DataLines = dataLines;
opts.Delimiter = ",";
% Specify column names and types
% opts.VariableNames = "Date3262014";
opts.VariableTypes = "double";
% Specify file level properties
opts.ExtraColumnsRule = "ignore";
```

```
opts.EmptyLineRule = "read";
% Import the data
filedata = readtable(filename, opts);
filedata = table2array(filedata);
end
```