Introduction

The purpose of this assignment was to develop a regression 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. Trial data was either collected as “healthy” with standard operating procedures, or “faulty” with a level 2 unbalance of two screws being added to the disk in order to introduce an unbalance to the system. After data processing through logistic regression, test data was analyzed to determine if the vibrations experienced were that of a faulty or healthy shaft.

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” and “faulty” 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. Data was analyzed to determine what level of unbalance it had, from 0 screws introduced to 2.

Implemented Solution

1. **Sample Preparation:**

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

File Listing:
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.
- faulty_fileList contains the list of faulty training data files.
- test_fileList contains the list of testing data files.

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

Data Reading and Transcription:
Two loops are used to read and transcribe data from the text files.

- For training data (healthy and faulty):

  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 also has 38400 rows and 30 columns.

Data Sampling Strategy:
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 and 20 samples from the "Faulty" 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.

2. **Feature extraction and selection:**

Using Fast Fourier Transform (FFT) on each sample and recording the amplitude of the first peak in the resulting frequency spectrum as a feature. The approach for the feature extraction used in the program is explained below in details:

Loop Iteration:
The code begins by entering a loop that iterates through each of the 20 data samples for training and 30 data samples for testing.

Sampling Parameters:
Sampling parameters are defined:

1. Fs: Sampling frequency, set to 2560 Hz.
2. T: Sampling period, computed as the inverse of Fs, representing the time interval between consecutive data points.
3. L: Length of the signal, set to 38400. This is the number of data points in each sample.

Data Loading:
The data from the current sample is loaded into the variable X. X contains the time-domain signal for the current sample.

Frequency Vector Generation:
A frequency vector f is generated. This vector represents the frequencies corresponding to the FFT output. It is calculated as follows:

$$f = Fs/L * (0:(L/2-1))$$

Fs/L calculates the frequency resolution, and (0:(L/2-1)) creates an array of frequencies up to half the sampling frequency. This represents the positive frequencies because the FFT result is symmetric.

FFT Computation:
The FFT is computed for the time-domain signal X using the fft function. This results in a complex-valued frequency spectrum, stored in the variable Y.

Amplitude Spectrum Calculation:
The code calculates the amplitude spectrum of the FFT result (Y) in the following steps:

1. P2 = abs(Y/L): Computes the absolute values of the FFT coefficients and scales them by 1/L.
2. P1 = P2(1:L/2): Extracts the single-sided amplitude spectrum by considering only the positive frequency components (up to half of the spectrum).
3. P1(2:end-1) = 2*P1(2:end-1): Doubles the amplitude of all frequency components except the DC (0 Hz) and Nyquist frequency components to account for the single-sided spectrum.

Peak Identification:
The code identifies peaks in the amplitude spectrum using the findpeaks function.
The MinPeakDistance parameter is set to 300, which means that two peaks must be at least 300 frequency bins apart to be considered distinct. The result, pks, contains the peak amplitudes and their corresponding frequencies.

Recording the First Peak:
The code records the amplitude of the first peak in the normal_amplitude array for the current sample. Since pks contains peaks sorted by amplitude, pks(2) corresponds to the amplitude of the first peak in the spectrum.

Iteration:
The loop continues to the next sample (i) and repeats the feature extraction process for all 20 samples.

The result of this code is a feature vector (normal_amplitude) containing the amplitude of the first peak in the frequency spectrum for each of the 20 data samples. Using a similar approach, we extracted the features from the test samples.

3. Algorithm:

FeatureMatrix Construction:
The FeatureMatrix is constructed by concatenating three sets of feature vectors vertically:

1. normal_amplitude: Feature vectors (amplitudes of the first peak in the frequency spectrum) for normal samples.
2. faulty_amplitude: Feature vectors for faulty samples.
3. test_amplitude: Feature vectors for test samples.

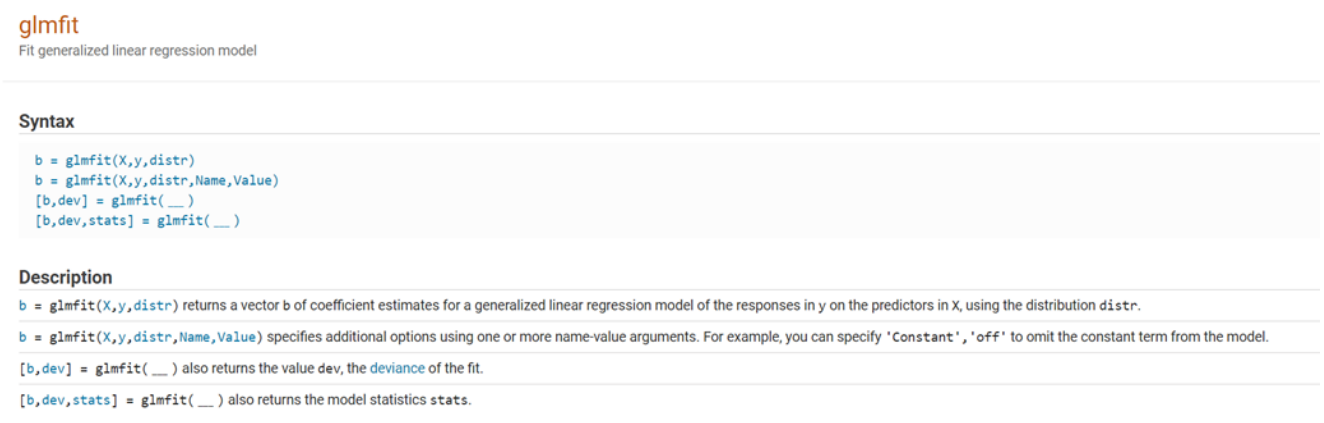
Each row of the FeatureMatrix represents a different sample, and each column corresponds to a feature (amplitude of the first peak).

Data Splitting:
The feature matrix is split into two subsets for training the LR model:

- BaselineData: This matrix includes the feature vectors for the first 20 samples (assumed to be normal or baseline data).
- DegradedData: This matrix includes the feature vectors for the next 20 samples (assumed to be degraded or faulty data).

Label Vector Creation:
A label vector Label is created to associate class labels with the training data. For BaselineData, labels are set to 0.95, indicating good or normal samples. For DegradedData, labels are set to 0.05, indicating degraded or faulty samples. The labels are created using the ones function to generate vectors of the appropriate size and then scaling them.

LR Model Training:
The logistic regression model is trained using the glmfit function. The input to the model is the concatenated feature matrix [BaselineData; DegradedData], and the output is the label vector Label. The 'binomial' option specifies that a binomial logistic regression model should be used, suitable for binary classification problems.



glmfit

Fit generalized linear regression model

Syntax

```
b = glmfit(X,y,distr)
b = glmfit(X,y,distr,Name,Value)
[b,dev] = glmfit( __ )
[b,dev,stats] = glmfit( __ )
```

Description

`b = glmfit(X,y,distr)` returns a vector b of coefficient estimates for a generalized linear regression model of the responses in y on the predictors in X, using the distribution `distr`.

`b = glmfit(X,y,distr,Name,Value)` specifies additional options using one or more name-value arguments. For example, you can specify `'Constant','off'` to omit the constant term from the model.

`[b,dev] = glmfit( __ )` also returns the value `dev`, the deviance of the fit.

`[b,dev,stats] = glmfit( __ )` also returns the model statistics `stats`.

Test Data Selection:
A set of test sample indices, TestSampleIndex, is defined. This set includes indices from 41 to 70, corresponding to 30 test samples.

Test Feature Matrix:
The feature matrix for the test samples, TestFeatureMatrix, is constructed by selecting rows from the FeatureMatrix based on the TestSampleIndex.

Confidence Value Calculation:
The trained LR model (beta) is used to calculate a "CV" (Confidence Value) or "Health Value" for the test data. The glmval function is applied to the test feature matrix (TestFeatureMatrix) using the LR model. The result, CV_Test, contains the calculated confidence values for each test sample. The 'logit' option specifies that the logistic regression model should be used for this calculation.

glmval

Generalized linear model values

Syntax

```
yhat = glmval(b,X,link)
[yhat,dylo,dyhi] = glmval(b,X,link,stats)
[...] = glmval(...,param1,val1,param2,val2,...)
```

Flow Diagram:

Data Preparation:

- Data is sourced from text files in three directories: "Healthy," "Faulty," and "Testing."
- File listings are generated for each directory (healthy_fileList, faulty_fileList, test_fileList).
- Matrices (normal_data, faulty_data, test_data) are initialized for data storage.
- Data is read from text files and transcribed into matrices for training and testing.

Feature Extraction:

For each sample in the training and testing data:

- Sampling parameters (Fs, T, L) are defined.
- Time-domain data is loaded into the variable X.
- A frequency vector f is generated based on Fs and L.
- FFT is computed to obtain the frequency spectrum (Y).
- Amplitude spectrum (P1) is calculated from Y.
- Peaks in the amplitude spectrum are identified using findpeaks.
- The amplitude of the first peak is recorded in normal_amplitude or test_amplitude.

FeatureMatrix Construction:

- Feature vectors (amplitudes of the first peak) are organized into FeatureMatrix.
- FeatureMatrix includes normal_amplitude, faulty_amplitude, and test_amplitude.
- Each row represents a sample, and columns correspond to features.

Data Splitting for Training:

- Training data is split into BaselineData and DegradedData.
- BaselineData contains feature vectors for normal samples.
- DegradedData contains feature vectors for faulty samples.

Label Vector Creation:

- A label vector (Label) is created for training data.
- Labels are set to 0.95 for normal samples and 0.05 for faulty samples.

LR Model Training:

- A logistic regression (LR) model is trained using glmfit.
- BaselineData and DegradedData serve as input, and Label is the output.
- The 'binomial' option specifies binary classification.

Test Feature Matrix:

- The feature matrix for test samples (TestFeatureMatrix) is constructed.
- It includes feature vectors for the selected test samples.

Confidence Value Calculation:

- The trained LR model (beta) is used to calculate health values (CV_Test) for test samples.
- glmval is applied to TestFeatureMatrix using the LR model.

Results

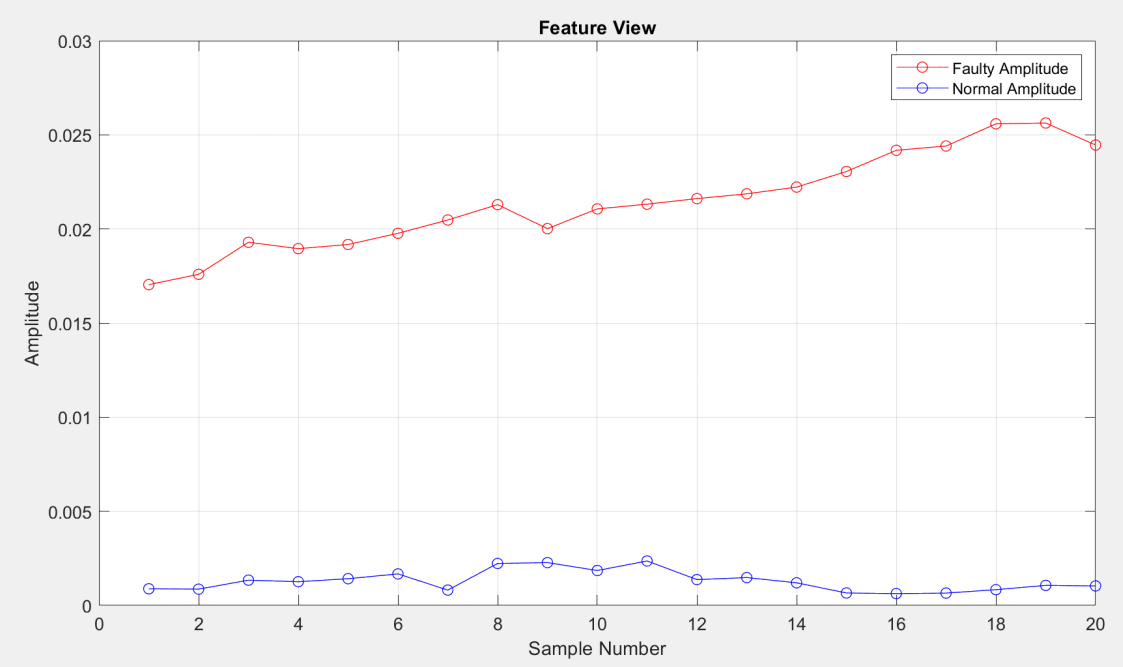


Figure 1: Feature View of Healthy and Faulty Training Samples at 1X Harmonic

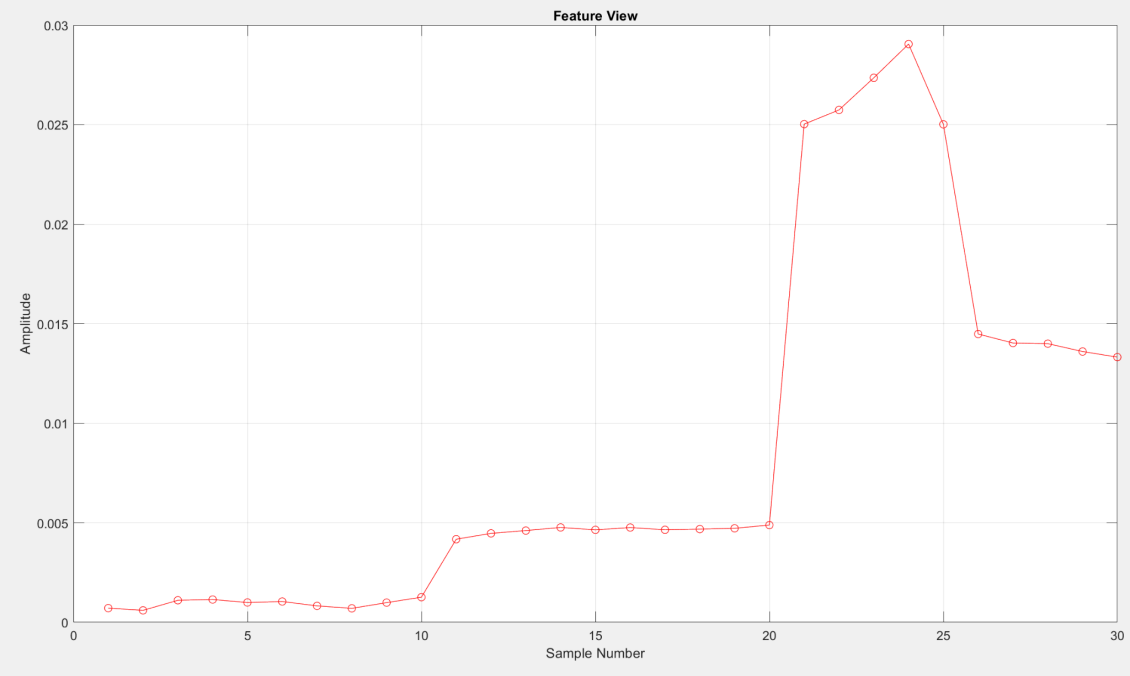


Figure 2: Feature View of Testing Samples at 1X Harmonic

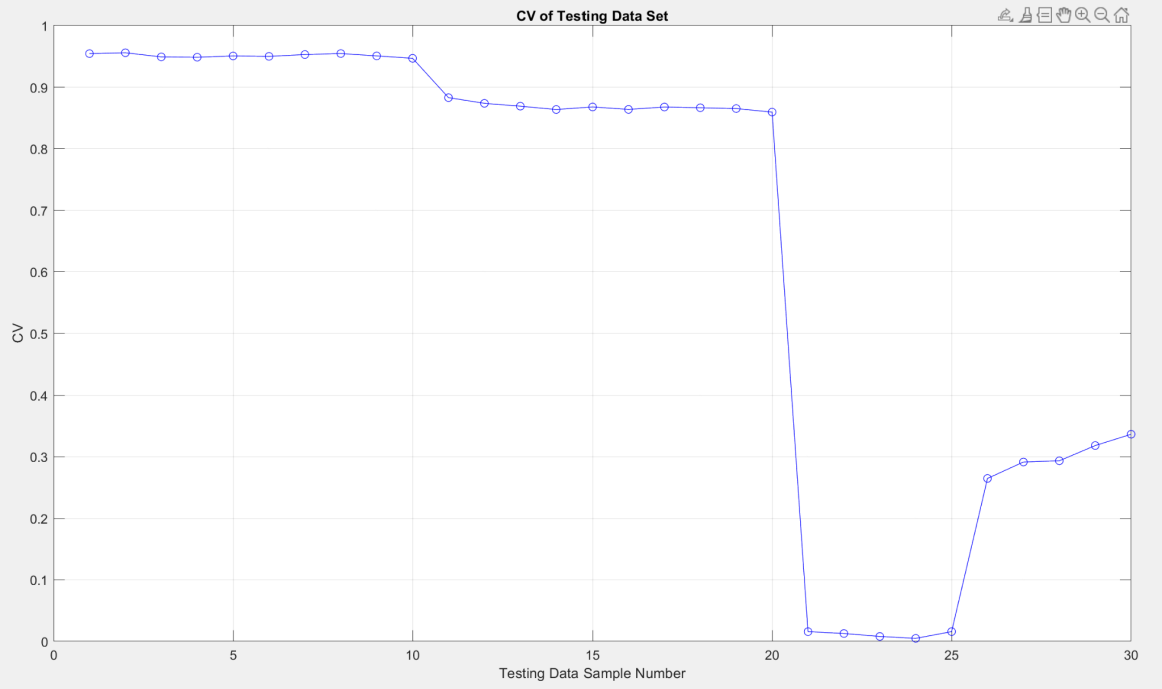


Figure 3: Confidence Value of Testing Samples with respect to trained LR Model

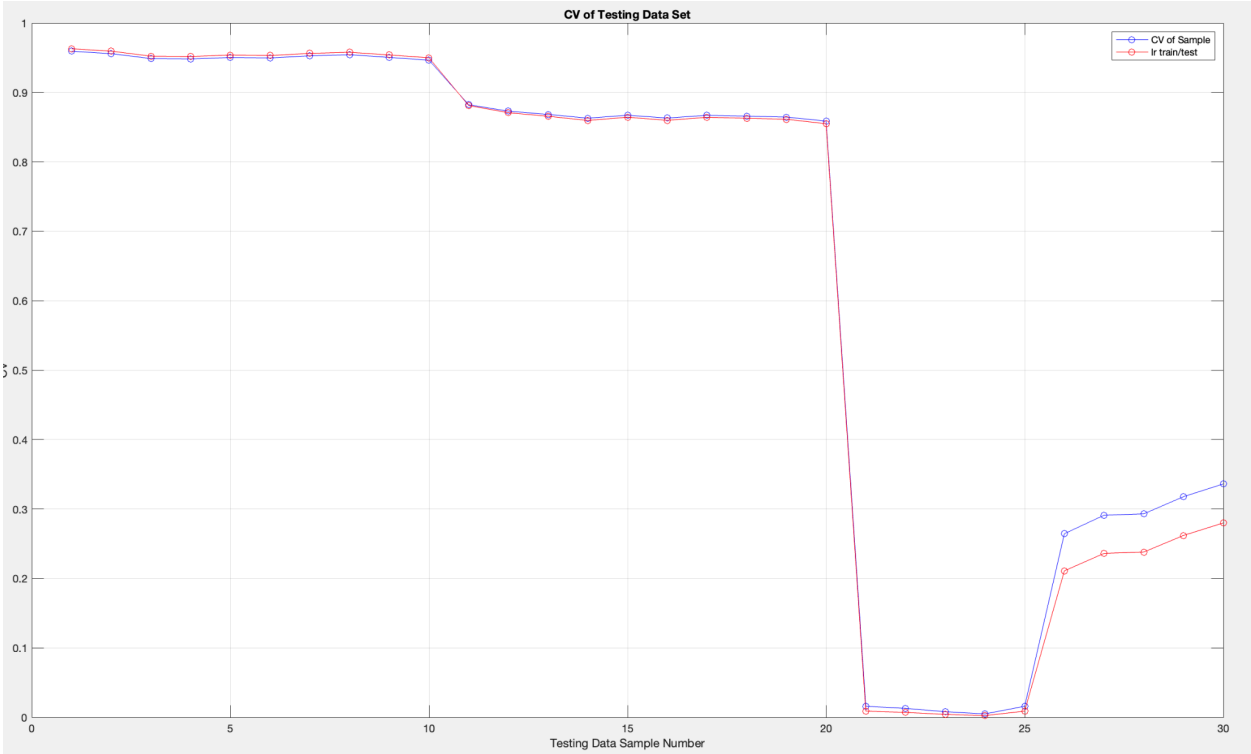


Figure 4 : LR_CV Model versus LR_Train and LR_Test Model

Discussion

The modeling method uses logistic regression to categorize the outputs into two categories established by the training data: healthy and faulty.

Since it is known that the test stand speed was set to 20 Hz, the initial step was to extract the amplitude of the first harmonic peaks from each sample's frequency plot near 20 Hz. Manual observation showed this was consistently captured by recording the first peak within the first 300 points (after the peak at 0) of each FFT plot, which is modified to be a single-sided amplitude plot. This way, only the real frequency values are under consideration for containing meaningful information that can be used for feature extraction.

Figure 1 shows that there is a clear difference between the healthy and faulty training sample sets; the faulty samples show acceleration values nearly ten times higher than those of healthy samples.

This difference is expected, due to the centrifugal force caused by the eccentrically loaded screws as the shaft assembly rotates about its axis. The test stand's accelerometer, which records vibrational data by way of transverse accelerations during testing, would capture a sinusoidal pattern with significantly higher amplitudes in the faulty data compared to the healthy sets.

Next, the same process of plotting the amplitude peaks near 20 Hz is repeated for the testing data sample set. Results are shown in Figure 2. The output graph here already lends some visibility to the distinction between potentially healthy and faulty samples, based on comparisons to the amplitudes presented in the testing and training samples.

At this point in the script, the LR model has been trained with 20 samples of healthy and faulty data. The final step is to generate plots representing the confidence values of the testing samples, based on the trained logistic regression model. Figure 4 shows the CVs between the two different approaches, based on the suggested algorithms. The blue data points show the outputs solely based on the provided LR_CV.m script. The red points show the CV after using the LR_train.m and LR_test.m scripts.

The overlap between the two models is significantly close for samples 1 to 25, but there is a difference of roughly 0.05 between the models for samples 26 to 30. This offset is potentially due to the extra algorithmic steps that LR_train and LR_test utilize to manipulate the actual matrices before using the glmfit in training and glmval functions to calculate and fit the test values to the logistic regression curve. Whereas in comparison, LR_CV.m script uses glmfit

and glmval without any extra matrix manipulation. It is hard to determine which trained model is better at evaluating the health value of the samples without additional training and testing data.

Additionally, note that there is an approximate 0.1 CV difference between samples 1 to 10 and samples 11 to 20 despite both subsets considered as healthy. This could be due to a change in fixturing, sensors, calibration of the DAQ (data acquisition system), natural degradation, or any fluctuation caused by the variation in power supply within the machinery and its connected circuit. Given the opportunity, further investigation should be conducted to diagnose or explain the reason for the drop in CV.

Figure 5 provides an overlay of colored regions on the CV plots. These colors distinguish the CV ranges that likely indicate the health condition of a sample. If the CV is between 1 and 0.7, then there is high confidence according to the logistic regression model that the sample is healthy. However, if the sample is below 0.2, then there is high confidence that the sample is unhealthy with 2 screws. The issue however, is trying to find the upper limit for the unbalanced sample with 1 screw embedded. Hence, there is an orange zone between 0.2 and 0.5 to indicate that the sample is unhealthy with 1 screw. Between 0.5 and 0.7 is the yellow zone, where it is unsure whether the machine the sample is collected from is experiencing unbalance due to external factors such as smaller screws or from natural degradation of sensors, shaft, fixtures or any other machinery part.

Note that the limits of these regions are based on the limited sample sets; additional testing samples would reaffirm - or show a need to reassess - these cutoff values.

In addition to the first harmonics, consideration was taken to use the second harmonic amplitude as an additional analysis feature. However, as shown in Figure 6 and 7, the extracted features showed far less clear distinction between the healthy and faulty sets, indicating that this data would not further train the model in a beneficial way. The trained model based on the second harmonic peaks as features shown in Figure 8, in compassion to the model trained on the first harmonic peaks, is visibly less accurate and reliable in determining the healthiness of the test samples. This indicates further harmonic amplitudes do not present themselves as usable or meaningful data, compared to the first harmonic amplitudes.

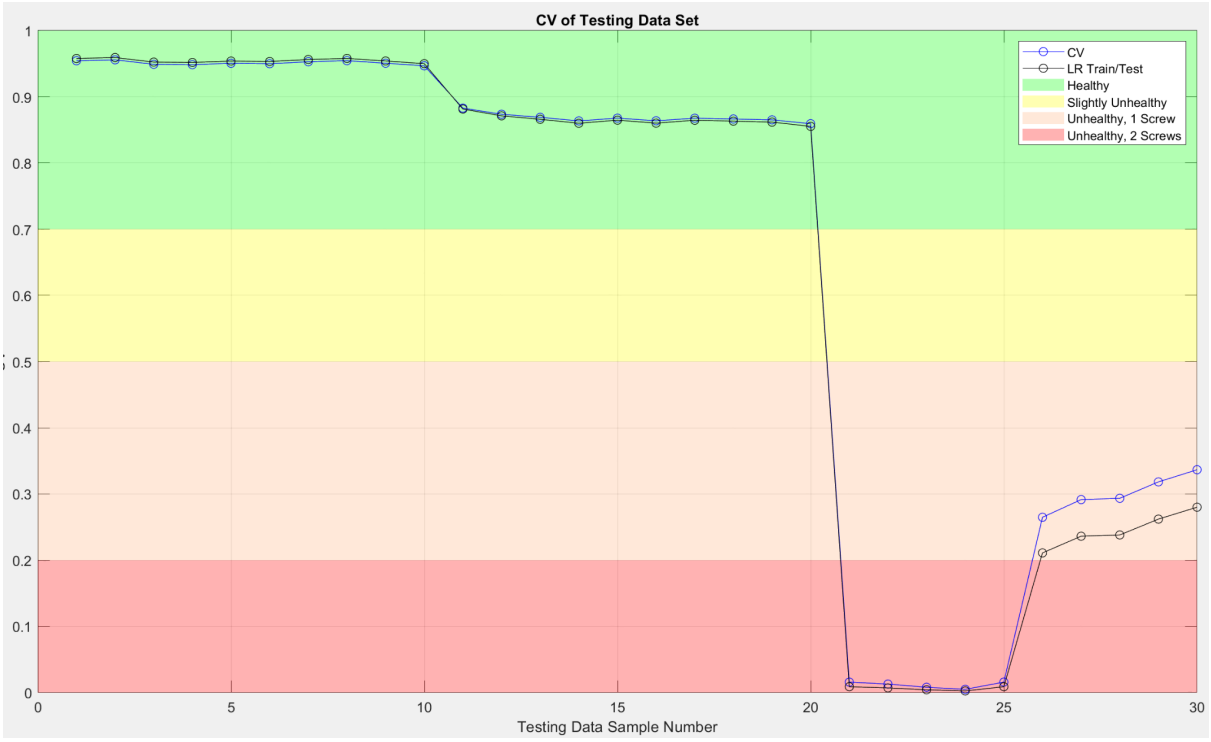


Figure 5: LR_CV Model versus LR_Train & LR_Test Model

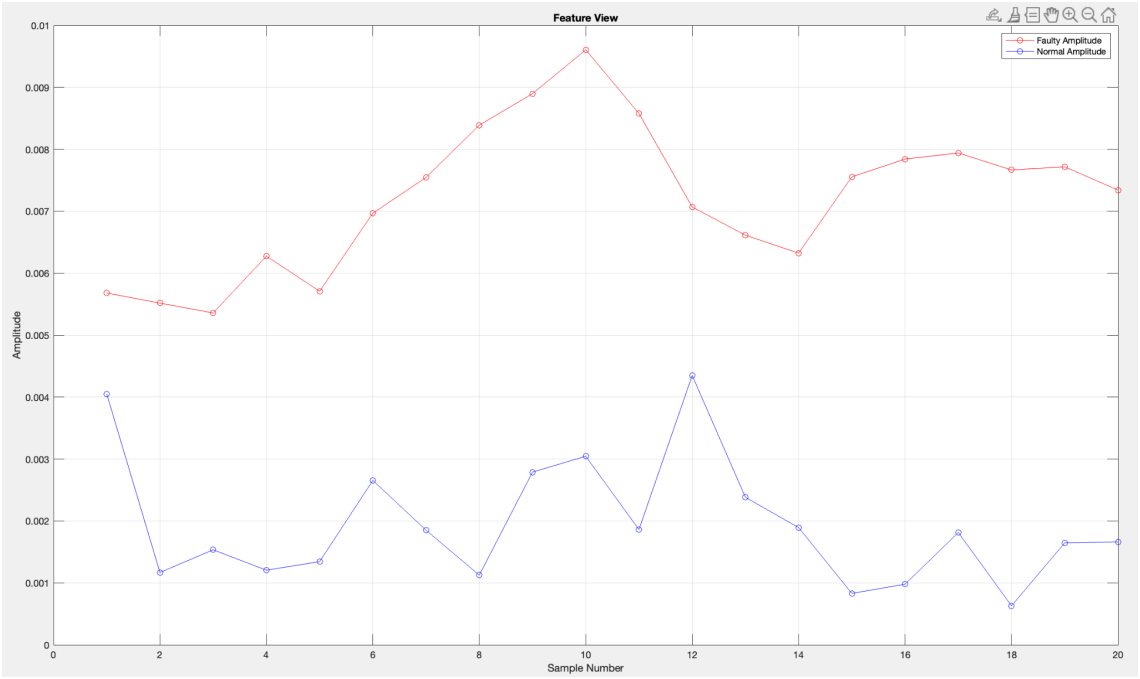


Figure 6: Feature View of Healthy and Faulty Training Samples at 2X Harmonic

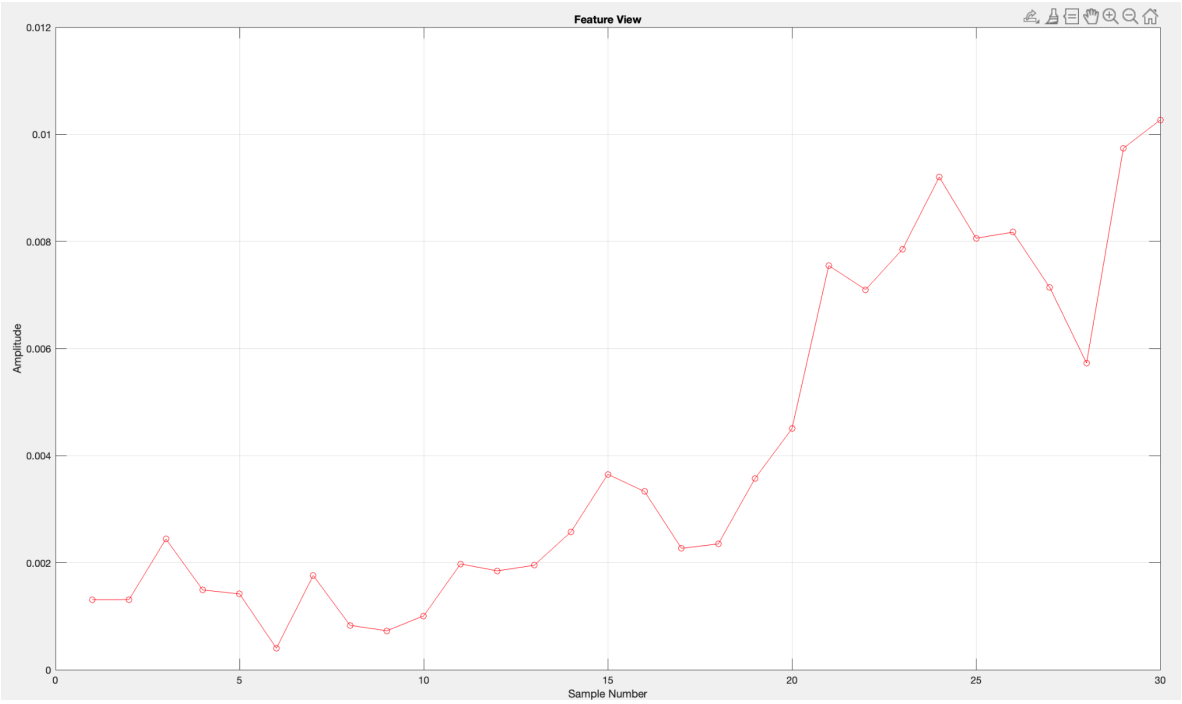


Figure 7: Feature View of Testing Samples at 2X Harmonic

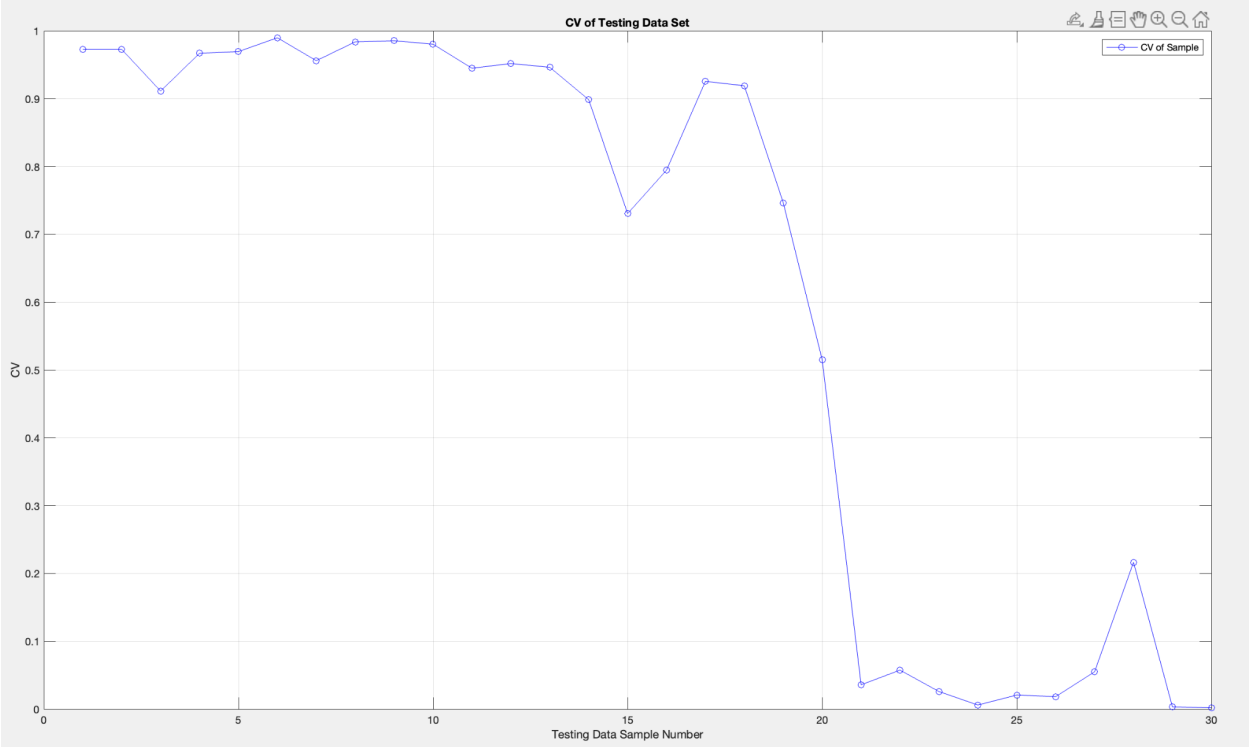


Figure 8: Confidence Value of Testing Samples trained with 2X Harmonic LR Data

Conclusion

This logistic regression training method has proven its ability to assess the provided test samples based on the first harmonic amplitudes from the provided data. The CVs generated by the model show visible differences between subgroups as outlined in Figure 5 to the point that they show the most likely number of unbalancing screws installed for a particular test.

Appendix – Codes

ENME485_HW2.m

```
clear; clc;
% Make sure you change file path
healthy_fileList =
dir(fullfile("/Users/user/Documents/UMD2023/ENME485/Assignments/HW2/Reading
Materials/Training/Healthy", "*.txt"));
faulty_fileList =
dir(fullfile("/Users/user/Documents/UMD2023/ENME485/Assignments/HW2/Reading
Materials/Training/Faulty", "*.txt"));
test_fileList =
dir(fullfile("/Users/user/Documents/UMD2023/ENME485/Assignments/HW2/Reading
Materials/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);
faulty_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(20,1);
faulty_amplitude = zeros(20,1);
test_amplitude = zeros(30,1);
% 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 =
"/Users/user/Documents/UMD2023/ENME485/Assignments/HW2/Reading
Materials/Training/Healthy/" + healthy_fileList(i,1).name;
bad_fileName = "/Users/user/Documents/UMD2023/ENME485/Assignments/HW2/Reading
Materials/Training/Faulty/" + faulty_fileList(i,1).name;
% Import the file data
healthy_data = import_file_data(healthy_fileName, datalines);
bad_data = import_file_data(bad_fileName, datalines);
% Transcribe it to matrix
for j = 1:38400
normal_data(j,i) = healthy_data(j,1);
faulty_data(j,i) = bad_data(j,1);
end
end
% Loop through and read each file for test data
for i = 1:30
test_fileName = "/Users/user/Documents/UMD2023/ENME485/Assignments/HW2/Reading
Materials/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
%%
% Loop through each sample-set and perform FFT and record first peak
for i = 1:20
Fs = 2560; % Sampling frequency
T = 1/Fs; % Sampling period
L = 38400; % Length of signal
% Load healthy dataset values
X = normal_data(:,i);
f = Fs/L*(0:(L/2-1));
% Perform FFT for Single-Sided Amplitude Spectrum
Y = fft(X);
P2 = abs(Y/L);
P1 = P2(1:L/2);
P1(2:end-1) = 2*P1(2:end-1);
% Identify the peaks within 400 values
pks = findpeaks(P1, 'MinPeakDistance',300);
% Record the first peak
normal_amplitude(i,1) = pks(2);
end
% Repeat but for faulty dataset
for i = 1:20
Fs = 2560; % Sampling frequency
T = 1/Fs; % Sampling period
L = 38400; % Length of signal
X = faulty_data(:,i);
f = Fs/L*(0:(L/2-1));
Y = fft(X);
P2 = abs(Y/L);
P1 = P2(1:L/2);
P1(2:end-1) = 2*P1(2:end-1);
pks = findpeaks(P1, 'MinPeakDistance',300);
faulty_amplitude(i,1) = pks(2);
end
% Plot Feature View graph
plot(faulty_amplitude, '-ro')
grid on
title('Feature View');
xlabel('Sample Number');
ylabel('Amplitude');
hold on
plot(normal_amplitude, '-bo')
hold off
legend('Faulty Amplitude', 'Normal Amplitude');
%% Plot Testing Features
```

```
% Repeat but for test dataset
for i = 1:30
Fs = 2560; % Sampling frequency
T = 1/Fs; % Sampling period
L = 38400; % Length of signal
X = test_data(:,i);
f = Fs/L*(0:(L/2-1));
Y = fft(X);
P2 = abs(Y/L);
P1 = P2(1:L/2);
P1(2:end-1) = 2*P1(2:end-1);
pks = findpeaks(P1, 'MinPeakDistance',300);
test_amplitude(i,1) = pks(2);
end
% Plot Feature View graph
plot(test_amplitude, '-ro')
grid on
title('Feature View');
xlabel('Sample Number');
ylabel('Amplitude');
%% Select Training Portion and PCA (front/back)
%GoodSampleIndex (in this example I assume the baseline data is first 20
%samples
GoodSampleIndex=1:20;
%Lets assume the next 20 samples are from a degraded system
DegradedSampleIndex=21:40;
FeatureMatrix = [normal_amplitude; faulty_amplitude; test_amplitude];
%Baseline Data
BaselineData=FeatureMatrix(GoodSampleIndex,:);
DegradedData=FeatureMatrix(DegradedSampleIndex,:);
%% Train LR Model
%Label Vector (0.95 for good samples, 0.05 for bad samples
Label=[ones(size(BaselineData,1),1)*0.95; ones(size(DegradedData,1),1)*0.05];
%fit LR Model (glm-fit)
beta = glmfit([BaselineData; DegradedData],Label,'binomial');
%% Calculating Health Value (using LR Model)
%assume I have some test-data (assume it is samples 41-70)
TestSampleIndex = 41:70;
TestFeatureMatrix=FeatureMatrix(TestSampleIndex,:);
%calculate CV (Health Value)
CV_Test = glmval(beta,TestFeatureMatrix,'logit') ; %Use LR Model
plot(CV_Test, '-bo')
legend('CV of Sample')
grid on
title("CV of Testing Data Set")
xlabel("Testing Data Sample Number")
ylabel("CV")
%% lr_train
result = lr_train(normal_amplitude, faulty_amplitude, 1);
```

```
result_test = lr_test(test_amplitude, result);
plot(CV_Test, '-bo')
grid on
title("CV of Testing Data Set")
xlabel("Testing Data Sample Number")
ylabel("CV")
hold on
plot(result_test, '-ro')
legend('CV of Sample', 'LR Train/Test')
%% DEBUGGING
% Plot in time-domain
Fs = 2560; % Sampling frequency
T = 1/Fs; % Sampling period
L = 38400; % Length of signal
t = (0:L-1)*T; % Time vector
X = normal_data(:,1);
plot(t,X)
title("Data Acceleration")
xlabel("t (sec)")
ylabel("Acceleration")
%% DEBUGGING
% Plot Fast Fourier Transform
% Referred to https://www.mathworks.com/help/matlab/ref/fft.html
f = Fs/L*(0:(L/2-1));
Y = fft(X);
P2 = abs(Y/L);
P1 = P2(1:L/2);
P1(2:end-1) = 2*P1(2:end-1);
plot(f,P1)
title("Single-Sided Amplitude Spectrum of fft(X)")
xlabel("f (Hz)")
ylabel("|P1(f)|")
% Find Peaks
[pks, locs] = findpeaks(P1, 'MinPeakDistance',300);
first_harmonic_peak = pks(3);
P1(locs(3))
```

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
```

lr_train.m

```
function result = lr_train(normal_data, faulty_data, SVD_CuttOff)
Data = [normal_data;faulty_data];
% do PCA first
% remove means
mu = mean(Data);
tempData = Data - repmat(mu,size(Data,1),1);
%normalize variance
STD = std(tempData, 1);
% tempData = tempData./repmat(STD, size(Data,1),1);
C = tempData' * tempData/(size(Data,1)-1);
%the above two lines do the same thing as the following line.
%C = cov(Data);
[V,S]=eig(C);
TotSum=trace(S);
VHlp=[];
DHlp=[];
Sum=0;
for i=size(Data,2):-1:1
if Sum<SVD_CuttOff*TotSum
DHlp = [DHlp S(i,i)];
VHlp = [VHlp V(:,i)];
Sum=Sum+S(i,i);
else
break;
end
end;
% calculate transformation matrix which lower dimension of Data and
% unifiy the data. The term diag(DHlp.^(-1/2)) is optional
% TransMatrix=diag(DHlp.^(-1/2))*transpose(VHlp);
TransMatrix=transpose(VHlp);
result.mu = mu;
result.std = STD;
result.TransMatrix = TransMatrix;
temp = normal_data - repmat(mu,size(normal_data,1),1);
%tranform data to principal compnent space.
%be careful normData are row vectors, so do not use TransMatrix * normData
Normal = temp * TransMatrix';
num_Normal = size(Normal,1);
temp = faulty_data - repmat(mu,size(faulty_data,1),1);
%tranform data to principal compnent space.
%be careful normData are row vectors, so do not use TransMatrix * normData
Faulty = temp * TransMatrix';
num_Faulty = size(Faulty,1);
%using Matlab built-in function in the statistics toolbox
Label = [ones(num_Normal,1)*0.95; ones(num_Faulty, 1)*0.05];
result.beta = glmfit([Normal;Faulty], Label, 'normal','link','logit');
```

lr_test.m

```
function result = lr_test(test_data, param)
mu = param.mu;
TransMatrix = param.TransMatrix;
STD = param.std;
temp = test_data - repmat(mu,size(test_data,1),1);
% temp = temp./repmat(STD, size(test_data,1),1);
%tranform data to principal compnent space.
%be careful normData are row vectors, so do not use TransMatrix * normData
Test = temp * TransMatrix';
%using Matlab built-in function in the statistics toolbox
result = glmval(param.beta, Test,'logit');
```

LR_CV.m

```
%E-Manufacturing 2013
%Logistic Regression
%How to Use This Method
%% clear stuff
clear all
clc
close all
%% Select Training Portion and PCA (front/back)
%GoodSampleIndex (in this example I assume the baseline data is first 100
%samples
GoodSampleIndex=1:20;
%Lets assume the last 100 samples are from a degraded system
DegradedSampleIndex=21:40;
FeatureMatrix = [normal_amplitude; faulty_amplitude; test_amplitude];
%Baseline Data
BaselineData=FeatureMatrix(GoodSampleIndex,:);
DegradedData=FeatureMatrix(DegradedSampleIndex,:);
%% Train LR Model
%Label Vector (0.95 for good samples, 0.05 for bad samples
Label=[ones(size(BaselineData,1),1)*0.95; ones(size(DegradedData,1),1)*0.05];
%fit LR Model (glm-fit)
beta = glmfit([BaselineData; DegradedData],Label,'binomial');
%% Calculating Health Value (using LR Model)
%assume I have some test-data (assume it is samples 301-400)
TestSampleIndex = 41:70;
TestFeatureMatrix=FeatureMatrix(TestSampleIndex,:);
%calculate CV (Health Value)
CV_Test = glmval(beta,TestFeatureMatrix,'logit') ; %Use LR Model
plot(CV_Test, '-ro')
grid on
```