Faculty of Natural and Mathematical Sciences
Department of Informatics

Bush House, King's College London
Strand Campus,30 Aldwych
London WC2B 4BG
Telephone 20 7848 2145
Fax 020 7848 2851

7CCSMDPJ

Individual Project Submission 2018/19

Name:	Jiachang Xu
Student Number:	1819235
Degree Programme:	MSc Data Science
Project Title:	Fault Diagnosis with Machine Learning Methods
Supervisor:	Dr. Hak-Keung Lam
Word Count:	4,954

RELEASE OF PROJECT

Following the submission of your project, the Department would like to make it publicly available via the library electronic resources. You will retain copyright of the project.

☑ I **agree** to the release of my project

☐ I **do not** agree to the release of my project

Signature: **Date:** August 22, 2019

Department of Informatics
King's College London
United Kingdom

7CCSMDPJ Individual Project

Fault Diagnosis with Machine Learning Methods

Name: **Jiachang Xu**
Student Number: 1819235
Course: MSc Data Science

Supervisor: Dr. Hak-Keung Lam

This dissertation is submitted for the degree of MSc Data Science.

Acknowledgements

In this section, I would like to pay special appreciation to my supervisor, Dr. Hak-Keung Lam, for guiding me through the methodology and scheduling of this project. I would also like to thank Guangyu Jia for helping me understand fast Fourier transform and MATLAB setup .

Abstract

Because the roller bearings are critical moving components in all kinds of machinery, the demand for an automatic diagnosis system that accurately identifies fault types of defect bearings is urgent. This paper discovers a novel feature extraction method that stacks discriminative features from different domains, such as traditional statistical analysis, ensemble empirical mode decomposition (EEMD), etc. The experiment proceeds under a controlled environment, which is a 3-layer artificial neural network (ANN) with 10 hidden units. The experiment further proves hat the selected feature combination generalises to different data sources and different load factors. Based on the discovery of the effective feature combination, this paper proposes an automated fault diagnosis system that contain multiple models for different load factor intervals to predict the fault type of a roller bearing.

Nomenclature

$\mathbf{a}^{[l]}$	The activation vector at the $l-1^{\text{th}}$ layer in the ANN architecture
$\mathbf{b}^{[l]}$	The bias vector from the $l-1^{\text{th}}$ layer to the l^{th} layer in the ANN architecture
CWRU	Case Western Reserve University
E	The number of ensembles in the EEMD method
EEMD	Ensemble empirical mode decomposition
`EEMD_F1`	EEMD feature #1 (visually discriminative)
`EEMD_F3`	EEMD feature #5 (visually discriminative)
`EEMD_F6`	EEMD feature #6 (visually discriminative)
`stats_F14`	Statistical feature #14 (visually discriminative)
`stats_F19`	Statistical feature #19 (visually discriminative)
EMD	Empirical mode decomposition
$\mathbf{f}$	A vector of frequency values derived from fast Fourier transform
I	The number of intrinsic mode functions in the EEMD method
IMF	Intrinsic mode functions
K	The number of spectrum lines from fast Fourier transform
M	The number of samples in a data set
MFPT	Society For Machinery Failure Prevention Technology
N	The number of data points in a sample
$\mathbf{s}$	A vector of spectrum amplitudes derived from fast Fourier transform
sf	The sampling frequency
$\mathbf{T}_{\mathbf{normal}}$	A row vector of numerical target outputs, where each entry is a sample
$\mathbf{T}_{\mathbf{onehot}}$	A matrix of one-hot target outputs, where each column is a sample
`top3_combo`	The stacking of the original values of the top 3 candidate features (i.e. EEMD feature #5, EEMD feature #6, statistical feature #14) along with their respective averages on the dimension of z-score
$\mathbf{W}^{[l]}$	The weight matrix from the $l-1^{\text{th}}$ layer to the l^{th} layer in the ANN architecture
$\mathbf{X}$	A matrix where each column is a signal series
$\mathbf{x}$	A column vector that represents a signal series
$\mathbf{z}^{[l]}$	The weighted sum vector from the $l-1^{\text{th}}$ layer to the l^{th} layer in the ANN architecture
z	The z-score threshold

Contents

List of Figures

List of Tables

1 Introduction

Roller bearings are critical components in any rotary machinery. Undetected faulty bearings can cause lower operational efficiency and higher maintenance cost [2]. Therefore, it is an imperative to develop an automated diagnosis system to accurately categorise bearing fault types [3]. Previous researches mostly focus on finding effective feature extraction method, which can produce accurate categorisation with shallow classification models. However, most previous papers uses different feature extraction methods, and different classification models. For example, one paper may simply extract statistical features from the vibration signals, and choose support vector machine for classification purpose [4]; another publication may use ensemble empirical mode decomposition (EEMD) to extract more complex, and well-hidden features, and use ANN with wavelet activation for classification [5].Although the researchers' liberty of choosing their own combination of feature extraction method and classification model promotes academic breakthrough, it is hard to rank the effectiveness of feature extraction methods under a controlled environment. Additional challenges include the possibility that a specific combination of feature extraction and classification model may work on 1 data set, but fails to generalise to completely different data. Therefore, it begs the questions which feature extraction method is the most discriminative under a controlled environment, and how it would generalise. More specifically, this paper aims to find the most effective composite selection of features under the realm of traditional statistical analysis and EEMD algorithm, coupled with a 3-layer ANN model to make the final categorisation of bearing fault type; this paper also examines the degree of generalisation of the chosen method (i.e. composed features, and model architecture) to different data sets. The chosen architecture of the classification model serves as the controlled environment in the experimentation. The experimental results show that the combination of 2 EEMD-derived features and 1 traditional frequency-domain feature can achieve accurate prediction with this controlled environment of a shallow ANN architecture. Furthermore, this feature combination is proven to generalise to both different data sources and different load factors. Based on these experimental findings, this paper proposes a multi-model fault diagnosis system to categorise the fault type of a vibration signal series. The structure of this paper is organised as follows. Section 2 presents all the technical background needed to understand this project. Section 3 reviews the literature that is related to this field. Section 4 lists the formal process to conduct experimentation in this project. Section 5 discusses the results of experimentation, and proposes an automated diagnosis system based on the findings from the results.

2 Background

This section provides necessary background information to understand this project, including fast Fourier transform (Subsection 2.1), traditional time-domain and frequency-domain analysis (Subsection 2.2), empirical mode decomposition (Subsection 2.3), ensemble empirical mode decomposition (Subsection 2.4), and artificial neural network (Subsection 2.5).

2.1 Fast Fourier Transform

The fast Fourier transform (FFT) algorithm is an improvement of Fourier transform, which breaks down a series of signal into it underlying frequencies and the corresponding amplitudes of those frequencies [6, 7]. The FFT algorithm has an advantageous time complexity of $O(N\log N)$ [8]. In the practise of programming, the MATLAB function `fft()` [1] can be utilised to perform the FFT algorithm on a signal series. In Figure 1 [1], $\mathbf{X}(t)$ is a signal series corrupted with zero-mean random noise. After we apply the `fft()` method on $\mathbf{X}(t)$, its underlying frequencies are identified as $\mathbf{f}$, and corresponding amplitude $|\ \mathbf{P1}(f)\ |$. The true underlying frequencies can be identified by "spikes" in the spectrum (Figure 2), which are approximately 50 Hz and 125 Hz. The minor fluctuation in the spectrum is introduced by the presence of zero-mean random noise.

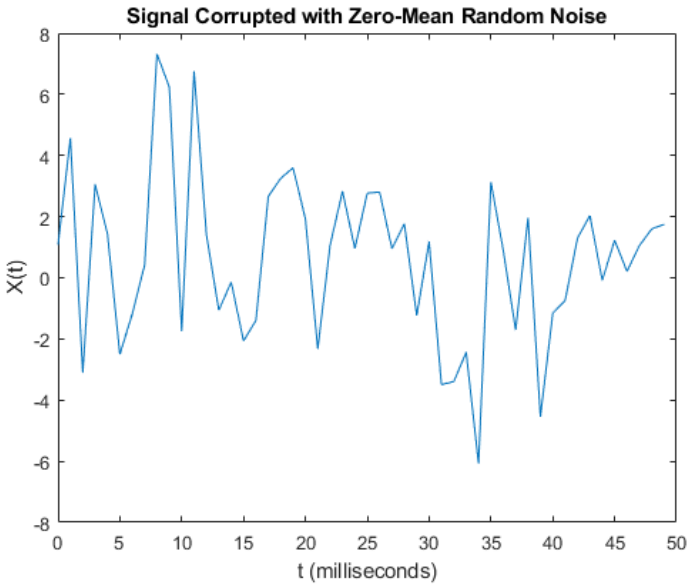

Figure 1: Signal corrupted with zero-mean random noise [1]

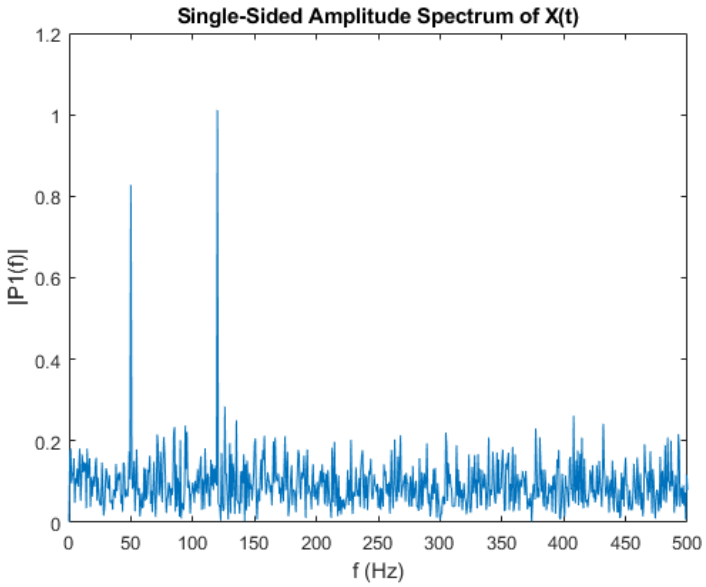

Figure 2: Single-sided amplitude spectrum of $\mathbf{X}(t)$ [1]

2.2 Traditional Statistical Analysis

The traditional statistical analysis [4, 9] calculates statistical features in the time domain and frequency domain. There are 6 time-domain features and 13 frequency-domain features used in the experimentation of this project.

2.2.1 Time-Domain Analysis

Time-domain analysis extracts features directly from the raw vibration signal. The raw vibration signal series is represented by a vector $\mathbf{x}$ of length N. $\mathbf{x}(n)$ represents the n^{th} element in the raw signal. There are 6 time-domain features [4, 9] derived for this project. They are the shape factor F_1 (Equation 2.1), the crest factor F_2 (Equation 2.2), the impulse factor F_3 (Equation 2.3), the margin factor F_4 (Equation 2.4), the kurtosis factor F_5 (Equation 2.5), and the skewness factor F_6 (Equation 2.6).

$$F_1 = \frac{\sqrt{\frac{1}{N}\sum_{n=1}^{N}\mathbf{x}(n)^2}}{\frac{1}{N}\sum_{n=1}^{N}|\mathbf{x}(n)|} \tag{2.1}$$

$$F_2 = \frac{\max|\mathbf{x}(n)|}{\sqrt{\frac{1}{N}\sum_{n=1}^{N}\mathbf{x}(n)^2}} \tag{2.2}$$

$$F_3 = \frac{\max|\mathbf{x}(n)|}{\frac{1}{N}\sum_{n=1}^{N}|\mathbf{x}(n)|} \tag{2.3}$$

$$F_4 = \frac{\max|\mathbf{x}(n)|}{\left(\frac{1}{N}\sum_{n=1}^{N}\sqrt{|\mathbf{x}(n)|}\right)^2} \tag{2.4}$$

$$F_5 = \frac{\frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}(n)-\mathbf{x})^4}{\left(\sqrt{\frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}(n)-\overline{\mathbf{x}})^2}\right)^4} \tag{2.5}$$

$$F_6 = \frac{\frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}(n)-\mathbf{x})^3}{\left(\sqrt{\frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}(n)-\overline{\mathbf{x}})^2}\right)^3} \tag{2.6}$$

2.2.2 Frequency-Domain Analysis

Frequency-domain analysis extracts features based on the underlying frequencies of the raw vibration signal. In this project, fast Fourier transform (Subsection 2.1) is used to compute the frequency-amplitude spectrum from the raw signal. The frequency spectrum and the amplitude spectrum are represented by vectors $\mathbf{f}$ and $\mathbf{s}$ of the same length K, respectively. $\mathbf{f}(k)$ represents the k^{th} element in the frequency vector $\mathbf{f}$; $\mathbf{s}(k)$ represents the k^{th} element in the amplitude vector $\mathbf{s}$; correspondingly, $\mathbf{s}(k)$ is the amplitude of the underlying frequency $\mathbf{f}(k)$ of the same index k. There are 13 frequency-domain features [4, 9] derived for this project, including the frequency-domain vibration energy F_7 (Equation 2.7), the metrics that measure the convergence of frequency spectrum power F_8, F_9, F_{10}, F_{12}, F_{16}, F_{17}, F_{18}, F_{19} (Equation 2.8, 2.9, 2.10, 2.12, 2.16, 2.17, 2.18, 2.19, respectively), and the metrics that describes the position change of main frequencies F_{11}, F_{13}, F_{14}, F_{15} (Equation 2.11, 2.13, 2.14, 2.15, respectively).

$$F_7 = \frac{\sum_{k=1}^{K}\mathbf{s}(k)}{K} \tag{2.7}$$

$$F_8 = \frac{\sum_{k=1}^{K}(\mathbf{s}(k)-F_7)^2}{K-1} \tag{2.8}$$

$$F_9 = \frac{\sum_{k=1}^{K}(\mathbf{s}(k)-F_7)^3}{K(\sqrt{F_8})^3} \tag{2.9}$$

$$F_{10} = \frac{\sum_{k=1}^{K}(\mathbf{s}(k)-F_7)^4}{KF_8^2} \tag{2.10}$$

$$F_{11} = \frac{\sum_{k=1}^{K}\mathbf{f}(k)\mathbf{s}(k)}{\sum_{k=1}^{K}\mathbf{s}(k)} \tag{2.11}$$

$$F_{12} = \sqrt{\frac{\sum_{k=1}^{K}(\mathbf{f}(k)-F_{11})^2\mathbf{s}(k)}{K}} \tag{2.12}$$

$$F_{13} = \sqrt{\frac{\sum_{k=1}^{K}(\mathbf{f}(k))^2\mathbf{s}(k)}{\sum_{k=1}^{K}\mathbf{s}(k)}} \tag{2.13}$$

$$F_{14} = \sqrt{\frac{\sum_{k=1}^{K}(\mathbf{f}(k))^4\mathbf{s}(k)}{\sum_{k=1}^{K}(\mathbf{f}(k))^2\mathbf{s}(k)}} \tag{2.14}$$

$$F_{15} = \frac{\sum_{k=1}^{K}(\mathbf{f}(k))^2\mathbf{s}(k)}{\sqrt{\sum_{k=1}^{K}\mathbf{s}(k)\sum_{k=1}^{K}(\mathbf{f}(k))^4\mathbf{s}(k)}} \tag{2.15}$$

$$F_{16} = \frac{F_{12}}{F_{11}} \tag{2.16}$$

$$F_{17} = \frac{\sum_{k=1}^{K}(\mathbf{f}(k) - F_{11})^3\mathbf{s}(k)}{K(F_{12})^3} \tag{2.17}$$

$$F_{18} = \frac{\sum_{k=1}^{K}(\mathbf{f}(k) - F_{11})^4\mathbf{s}(k)}{K(F_{12})^4} \tag{2.18}$$

$$F_{19} = \frac{\sum_{k=1}^{K}(\mathbf{f}(k) - F_{11})^2\mathbf{s}(k)}{K\sqrt{F_{12}}} \tag{2.19}$$

2.3 Empirical Mode Decomposition

The empirical mode decomposition (EMD) method decomposes a signal series into a set of intrinsic mode functions (IMFs) [10]. An IMF is a function where the a function has the properties[9]:

1. that the number of extrema and that of zero-crossing differ by either 0 or 1, and
2. that the upper and lower envelopes defined by the local maxima and the local minima, respectively, are zero-meaned.

Algorithm 1 demonstrates how the EMD method decomposes a raw signal $\mathbf{x}(t)$ into I IMFs $\mathbf{c}_i(i = 1, 2, ..., I)$, and the residue $\mathbf{r}_I$, step by step.

2.4 Ensemble Empirical Mode Decomposition

According to Lei et al [5], the EMD method above tends to mix modes, which might lead to the failure to represent the original signal accurately. Therefore, as an improvement to the EMD method, the ensemble empirical mode decomposition (EEMD) method is proposed, which defines the constituting IMFs as the average an ensemble of M trails [12]. More specifically (Algorithm 2), for each of the E ensembles, the EEMD method superposes a white noise $\mathbf{n}_m(t)$ upon the raw signal series $\mathbf{x}$ to create a new noise-added series $\mathbf{x}_e(t)$, and then decomposes $\mathbf{x}_e(t)$ into I IMFs $\mathbf{c}_{i,e}(i = 1, 2, ..., I)$. The final result

Algorithm 1: Empirical mode decomposition [9, 10, 11]

Input: $\mathbf{x}(t)$: the raw signal series
Initialise: $i = 0, \mathbf{r}_i = \mathbf{x}(t)$
repeat
  Iterate: $i \leftarrow i + 1$
  Initialise: $k = 0, \mathbf{h}_{i,k} = \mathbf{r}_{i-1}$
  repeat
    Iterate: $k \leftarrow k + 1$
    Locate the local maxima and the local minima of $\mathbf{h}_{i,k-1}$
    Interpolate the local maxima and the local minima by cubic spline lines to form the upper and lower envelops of $\mathbf{h}_{i,k-1}$, respectively
    Compute the average $\mathbf{m}_{i,k-1}$ of the upper and lower envelopes of $\mathbf{h}_{i,k-1}$
    Update: $\mathbf{h}_{i,k} \leftarrow \mathbf{h}_{i,k-1} - \mathbf{m}_{i,k-1}$
  until $\mathbf{h}_{i,k}$ is an IMF;
  Compute the i^{th} IMF: $\mathbf{c}_i \leftarrow \mathbf{h}_{i,k}$
  Compute the i^{th} residue: $\mathbf{r}_i \leftarrow \mathbf{r}_{i-1} - \mathbf{c}_i$
until $\mathbf{r}_i$ has less than 2 extrema;
Compute the number of IMFs: $I \leftarrow i$
return the I IMFs $\mathbf{c}_i(i = 1, 2, ..., I)$, and the residue $\mathbf{r}_I$

that the EEMD method returns is the average $\mathbf{a}_i(i = 1, 2, ..., I)$ of each of the I IMFs $\mathbf{c}_{i,m}(i = 1, 2, ..., I, m = 1, 2, ..., M)$ across the dimension of m.

2.5 Artificial Neural Network

The concept of artificial neural network (ANN) [13] derives from biological neuron network. ANN can be graphically represented by a network of nodes (called neurons in ANN architecture) and directed edges (Figure 3). This project uses ANN for classification. An ANN model needs to be trained to obtain the capability to classify new samples. Each iteration of the training of the ANN has a forward propagation step and a backward propagation step. In forward propagation, the data flows from the input neurons, through the hidden neurons, to the output neurons, to make a prediction. When the data flows through a neuron, this neuron computes a weighted sum of the connected neurons from the previous layer, and then activates the weighted sum with a nonlinear function, such as the sigmoid function. In backward propagation, the gradients flow from the output neurons, through the hidden neurons, to the input neurons. When the gradients propagates back 1 layer, the weights on the edges are updated accordingly. A full cycle of forward propagation and backward propagation complete 1 iteration of training of the ANN model (Algorithm 3).

Algorithm 2: Ensemble empirical mode decomposition [2, 5, 12]

Input: E: the number of ensembles
Input: I: the number of intrinsic mode functions (IMFs)
Input: $\mathbf{x}$: the raw signal series
for $e = 1 : E$ **do**
  Add white noise with given amplitude $\mathbf{n}_e(t)$ on top of the raw vibration signal series $\mathbf{x}(t)$ to obtain the noise-added series of the e^{th} trial $\mathbf{x}_e(t)$

$$\mathbf{x}_e(t) \leftarrow \mathbf{x}(t) + \mathbf{n}_e(t)$$

  Use the EMD method to decompose the noise-added series $\mathbf{x}_e(t)$ into I IMFs $\mathbf{c}_{i,e}(i = 1, 2, ..., I)$
end
Compute the average a_i of each IMF from the E trials

$$\mathbf{a}_i \leftarrow \sum_{m=1}^{M} \mathbf{c}_{i,e}, i = 1, 2, ..., I, e = 1, 2, ..., E$$

return the average $\mathbf{a}_i(i = 1, 2, ..., I)$ of each of the I IMFs

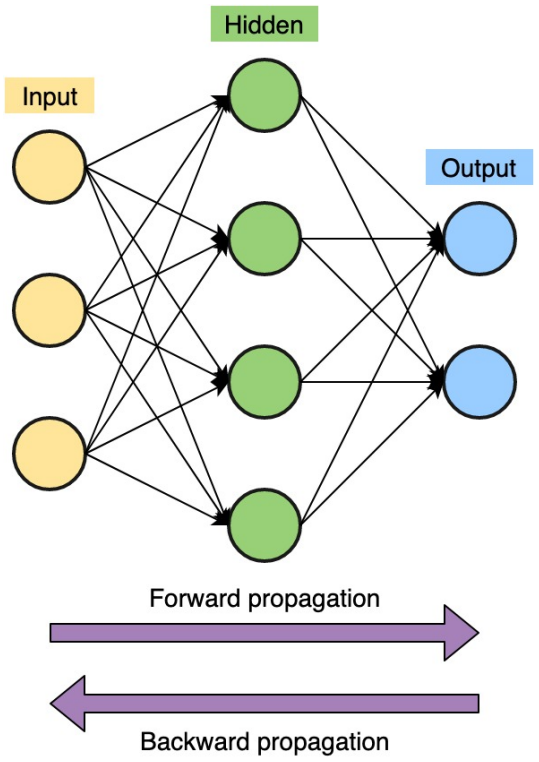

Figure 3: Block diagram of a 3-layer artificial neural network

Algorithm 3: Backpropagation training of ANN model [13]

Input: *examples*: a set of examples, each of which is represented by a input vector $\mathbf{x}$ and a output vector $\mathbf{y}$
Input: *network*: an ANN model with L layers, weights $w_{i,j}$, and an activation function g

```
/* Initialisation                                                    */
```
foreach weight $w_{i,j}$ **do**
  $w_{i,j} \leftarrow$ a small random number
end
```
/* Training                                                          */
```
while stopping criterion is NOT satisfied **do**
  foreach example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
    `/* Forward propagation */`
    foreach node i in the input layer **do**
      $a_i \leftarrow x_i$
    end
    for $l = 2 : L$ **do**
      foreach node j in layer l **do**
        $z_j \leftarrow \sum_i w_{i,j} a_j$
        $a_j \leftarrow g(z_j)$
      end
    end
    `/* Backward propagation */`
    foreach node j in the output layer **do**
      $\Delta[j] \leftarrow g'(z_j)(y_j - a_j)$
    end
    for $l = (L-1) : 1$ **do**
      foreach node i in layer l **do**
        $\Delta[i] \leftarrow g'(z_i) \sum_j w_{i,j} \Delta[j]$
      end
    end
  end
  `/* Update weights */`
  foreach weight $w_{i,j}$ in *network* **do**
    $w_{i,j} \leftarrow w_{i,j} + \alpha * a_i * \Delta[j]$
  end
end
return a train ANN model

3 Related Work

This section reviews the literature that develops new theories and methodologies in the field of bearing fault diagnosis. Lei et al. [5] extracts 10 frequency-domain features using ensemble empirical decomposition method (EEMD) method, and produces accurate classification by feeding those features into a wavelet neural network. Liu et al. [14] combines local mean decomposition and multi-scale entropy as input features into a 3-layer backpropagation ANN to make accurate prediction. Muruganatham et al. [15] performs singular spectrum analysis to extract features and effectively classify bearing faults using ANN. Yu et al. [11] calculates EMD energy entropy as input features and uses ANN to precisely categorise bearing defect types. Zhang et al. [4] compute 19 statistical features, and uses SVM to classify bearing faults accurately. Hussein et al. [16] deploy advanced pretreatment to extract features and precisely classify fault types using a 1-nearest-neightbour classifier. Lei et al. [10] uses the EMD method to preprocesses the vibration signal, extracts traditional statistical features in time and frequency domains, and classifies samples using the combination of several adaptive neuro-fuzzy inference systems (ANFIS) and genetic algorithm (GA). These previous researches in the field of fault diagnosis all use arbitrary combination of unique feature extraction methods, and different classification methods. Therefore, it introduces the question which feature extraction method performs the best if the classification model is controlled. Hence the motivation of this project to find the most discriminative feature combination from a set of feature extraction methods.

4 Approach

This section describes the entire experimental process of this project (Figure 16). Subsection 4.1 formulates the problem this project tries to solve. Subsection 4.2 cites the sources of the data sets this project uses. Subsection 4.3 shows how this project cleans the raw data set ready for feature extraction. Subsection 4.4 lists the features extracted from the cleaned data sets. Subsection 4.5 designs the architecture of the classification model. Subsection 4.6 explains how to select the most discriminative feature(s) and/or combinations of features through trial and error by feeding into the classification model. Subsection 4.7 describes the testing procedure of how the selection of features, and the architecture of the classification model generalises to different data sets.

4.1 Problem Formulation

This project examines the effectiveness of composite selection of features, extracted by traditional statistical analysis, and ensemble empirical mode decomposition (EEMD), in predicting the fault categories of roller bearings, given a shallow architecture of a classification model. Such a shallow architecture serves as a controlled environment. The features that are selected based on their excel performance on the baseline data set will be further tested to see how the combination of selected features and the chosen model generalise to unseen data sets.

4.2 Data Acquisition

This project acquires the data sets from two sources. One is the Seeded Fault Test Data from Case Western Reserve University Bearing Data Center (CWRU) [17], and the other is the Fault Data Set from the Society For Machinery Failure Prevention Technology (MFPT) [18]. The CWRU data set is used for feature selection and generalisation; the MFPT data set is used solely for generalisation on different data sets.

4.3 Data Cleaning

The data cleaning process processes and partitions the 2 raw data sets into 5 clean data subsets for the purpose of feature selection and model generalisation.

4.3.1 Cleaning the CWRU Data

The experimental data description, including class labels, of the CWRU data set is listed in Table 1. The experimental data set is a subset of the raw CWRU data set. This CWRU experimental subset includes the normal condition and all 3 fault locations (i.e. outer raceway, inner raceway, ball), plus 4 fault diameters (i.e. 0.007 in., 0.014 in., 0.021 in., 0.028 in.). This approach aims to test if the feature extraction methods and classification models can identify the underlying characteristics of both different fault locations and variable fault diameters. The CWRU experiment data is further partitioned into 4 subsets based on load factors (Table 2): the subsets A, C, and D have

load factors 1 HP, 2 HP, and 3 HP, respectively; the subset V aggregates the subsets A, C, and D. The subset A is used to select the most discriminative features and/or feature combinations. The subsets C and D are used to examine how the discriminative features and/or feature combinations selected from the subset A generalise to unseen data of different load factors. The subset V is used to see if we need to train different models for different load factors to achieve more accurate performance.

Table 1: CWRU experimental data description

Sampling rate (Hz)	Fault location	Fault diameter (in.)	Class label
12,000	Normal	N/A	1
12,000	Outer raceway	0.014	2
12,000	Inner raceway	0.007	3
12,000	Inner raceway	0.014	4
12,000	Inner raceway	0.021	5
12,000	Inner raceway	0.028	6
12,000	ball	0.014	7

Table 2: CWRU experimental data partition (subset A/C/D/V)

Subset	Load (HP)	No. of samples	Class labels included
A	1	700	1, 2, 3, 4, 5, 6, 7
C	2	700	1, 2, 3, 4, 5, 6, 7
D	3	700	1, 2, 3, 4, 5, 6, 7
V[1]	1, 2, 3	2100	1, 2, 3, 4, 5, 6, 7

4.3.2 Cleaning the MFPT Data

The experimental data description, including class labels, is listed in Table 3. This MFPT experimental subset B includes the normal condition and 2 fault locations (i.e. outer raceway, inner raceway). The subset B is used to examine how the discriminative features and/or feature combinations selected from the CWRU subset A generalise to unseen data of completely different experimental conditions.

Table 3: MFPT experimental data description (subset B)

Sampling rate (Hz)	Fault location	Load (lb)	No. of samples	Class label
97,656	Normal	270	100	1
48,828	Outer raceway	250	100	2
48,828	Inner raceway	250	100	3

[1]CWRU subset V is simply the combination of CWRU subsets A, C, and D.

4.4 Feature Extraction

The feature extraction process computes features from the cleaned data sets. Candidate features include both traditional statistical features in time and frequency domains (Sub-subsection 4.4.1), and features derived from the EEMD method (Sub-subsection 4.4.2).

4.4.1 Traditional Statistical Features

Statistical feature extraction (Subsection 2.2) is a traditional method of time-domain and frequency-domain analysis [4]. Time-domain analysis extracts features directly from the raw sample of vibration signal series. A $1 * M$ row vector is used to vectorise the feature representation of a time-domain feature of multiple samples (Equation 4.1). In this vectorised representation, each element $f_1^{[m]}$ is the value of a specific time-domain feature of each sample.

$$\mathbf{F} = \begin{bmatrix} f_1^{[1]} & f_1^{[2]} & ... & f_1^{[M]} \end{bmatrix} \tag{4.1}$$

Frequency-domain analysis, on the other hand, needs to apply fast Fourier transform (FFT) on the raw vibration signal to obtain its frequency-amplitude spectrum. However, due to the existence of noise in the raw signal (Figure 1), it is difficult to identify the true underlying frequencies from the frequency-amplitude spectrum (Figure 2). Therefore, this project uses 5 z-score thresholds ($z > 0.0, 0.5, 1.0, 1.5, 2.0$) to filter underlying frequencies. Then, different frequency-domain features are extracted from the filtered underlying frequencies of the vibration signal. For each sample, there are 5 feature values derived under the same type of frequency-domain feature. Like that of each EEMD feature, a $6*M$ matrix is used to vectorise the feature representation (Equation 4.2). In this matrix representation, each column $F^{[m]} = [f_{z>0.0}^{[m]}, f_{z>0.5}^{[m]}, f_{z>1.0}^{[m]}, f_{z>1.5}^{[m]}, f_{z>2.0}^{[m]}, f_{avg}^{[m]}]^T$ represents the extracted features of the m^{th} sample. In each column vector, the first 5 elements $[f_{z>0.0}^{[m]}, f_{z>0.5}^{[m]}, f_{z>1.0}^{[m]}, f_{z>1.5}^{[m]}, f_{z>2.0}^{[m]}]^T$ are the values of a specific frequency-domain feature derived from the 5 groups of underlying frequencies for this sample, with z-score thresholds ($z > 0.0, 0.5, 1.0, 1.5, 2.0$), respectively. The last element $f_{avg}^{[m]}$ in each column vector is the average of the first 5 elements.

$$\mathbf{F} = \begin{bmatrix} f_{z>0.0}^{[1]} & f_{z>0.0}^{[2]} & \cdots & f_{z>0.0}^{[M]} \\ f_{z>0.5}^{[1]} & f_{z>0.5}^{[2]} & \cdots & f_{z>0.5}^{[M]} \\ f_{z>1.0}^{[1]} & f_{z>1.0}^{[2]} & \cdots & f_{z>1.0}^{[M]} \\ f_{z>1.5}^{[1]} & f_{z>1.5}^{[2]} & \cdots & f_{z>1.5}^{[M]} \\ f_{z>2.0}^{[1]} & f_{z>2.0}^{[2]} & \cdots & f_{z>2.0}^{[M]} \\ f_{avg}^{[1]} & f_{avg}^{[2]} & \cdots & f_{avg}^{[M]} \end{bmatrix} \tag{4.2}$$

4.4.2 EEMD Features

The EEMD method decomposes the raw vibration signal series into I IMFs which are averaged across M ensemble trials (Subsection 2.3) [2, 5, 12]. Then, the most sensitive

IMF $\mathbf{c}^*$ is selected from all the EEMD-derived IMFs using a method involving kurtosis (Algorithm 4).

Algorithm 4: Select the most sensitive EEMD-derived IMF $\mathbf{c}^*$

Input: $\mathbf{a}_{i,m}(i = 1, 2, ..., I, m = 1, 2, ..., M)$: the I EEMD-derived IMFs from each of the M samples of vibration signal

Compute the kurtosis $k_{i,m}$ of the i^{th} EEMD-derived IMF $\mathbf{c}_{i,m}$ of length N for each of the M samples of vibration signal:

$$k_{i,m} = \frac{N \sum_{n=1}^{N} (\mathbf{a}_{i,m,n} - \overline{\mathbf{a}_{i,m}})^4}{N \sum_{n=1}^{N} (\mathbf{a}_{i,m,n} - \overline{\mathbf{a}_{i,m}})^2}$$

Calculate the mean m_i and the standard deviation std_i of the kurtosis k_i of the i^{th} EEMD-derived IMF $\mathbf{c}_i$ across the M samples

$$avg_i = \frac{1}{M} \sum_{m=1}^{M} k_{i,m}$$

$$std_i = \sqrt{\frac{1}{M-1} \sum_{m=1}^{M} (k_{i,m} - avg_i)^2}$$

Select the most sensitive EEMD-derived IMF $\mathbf{c}^*$ with the highest score of the following criterion:

$$criterion_i = \begin{cases} avg_i * std_i, & \text{normal conditions} \\ \frac{std_i}{avg_i}, & \text{faulty conditions} \end{cases} (i = 1, 2, ..., I)$$

return the most sensitive EEMD-derived IMF $\mathbf{c}^*$

Time-domain and frequency-domain features can be extracted from the most sensitive IMF, due to its time-series nature. There are 4 time-domain factors, and 3 frequency-domain features extracted from the most sensitive IMF $\mathbf{c}^*$ of each vibration signal sample:

1. `EEMD_F1` (time-domain): standard deviation of $\mathbf{c}^*$
2. `EEMD_F2` (time-domain): kurtosis of $\mathbf{c}^*$ (Equation 2.5)
3. `EEMD_F3` (time-domain): shape factor of $\mathbf{c}^*$ (Equation 2.1)
4. `EEMD_F4` (time-domain): impulse factor of $\mathbf{c}^*$ (Equation 2.3)
5. `EEMD_F5` (frequency-domain): mean frequency of $\mathbf{c}^*$
6. `EEMD_F6` (frequency-domain): root mean squared frequency of $\mathbf{c}^*$

7. `EEMD_F7` (frequency-domain): standard deviation frequency of $\mathbf{c}^*$

Since the first 4 features are in the time domain, so we can extract them directly from the most sensitive IMF $\mathbf{c}_*$, and represent them using the format of Equation 4.1, just like the traditional time-domain features. The last 3 features are in the frequency domain. Therefore, we can use the FFT algorithm (Subsection 2.1) and the 5 z-score thresholds to identify the underlying frequency-amplitude spectrum of the most sensitive IMF $\mathbf{c}_*$. Then, the same vectorised representation (Equation 4.2) can be used to format each type of frequency-domain feature, which is computed from the underlying frequency-amplitude spectrum.

4.5 Model Architecture

This approach uses a 3-layer (i.e. 1 input layer, 1 hidden layer, 1 output layer) ANN as a controlled environment to test the discriminativeness of the candidate features. The number of neurons at the input layer depends on the number of feature values of a specific feature or feature combination; the hidden layer has 10 neurons; the output layer has 7 neurons for CWRU data, or 3 neurons for MFPT data. Figure 4 visualises the architecture of the 3-layer ANN model.

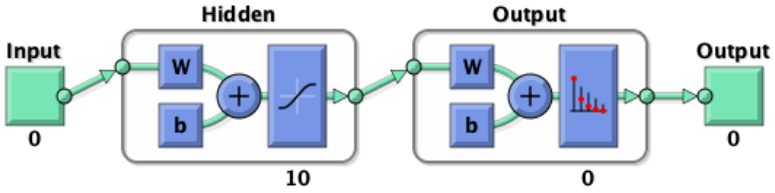

Figure 4: Block diagram of a 3-layer ANN model with 10 hidden neurons, where the number of neurons in the input layer depends on the type of selected features, and where the number of neurons in the output layer depends on which data set it operates on.

Equation 4.3 formulates the input vector $\mathbf{x}$ of 1 sample. Since the input layer acts as the activation of the 0^{th} layer, the input vector $\mathbf{x}$ is mathematically equivalent to $\mathbf{a}^{[0]}$.

$$\mathbf{a}^{[0]} = \mathbf{x} = \begin{bmatrix} x_1 & x_2 & ... & x_{n_x} \end{bmatrix}^T \tag{4.3}$$

Equations 4.4 and 4.5 formulate the weight matrix $\mathbf{W}^{[1]}$ and the bias vector $\mathbf{b}^{[1]}$ from the input layer to the hidden layer, respectively. $n^{[0]}$ is the number of feature values in the input layer. Equation 4.6 formulates the weighted sum $\mathbf{z}^{[1]}$ from the input layer to the hidden layer, using $\mathbf{W}^{[1]}$, $\mathbf{a}^{[0]}$, and $\mathbf{b}^{[1]}$. Equation 4.8 formulates the hidden activation vector $\mathbf{a}^{[1]}$ from the weighted sum $\mathbf{z}^{[1]}$ using the `sigmoid()` function (Equation 4.7).

$$\mathbf{W}^{[1]} \in \mathbb{R}^{10*n^{[0]}} \tag{4.4}$$

$$\mathbf{b}^{[1]} \in \mathbb{R}^{10*1} \tag{4.5}$$

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]} \cdot \mathbf{a}^{[0]} + \mathbf{b}^{[1]} = \begin{bmatrix} z_1^{[1]} & z_2^{[1]} & ... & z_{10}^{[1]} \end{bmatrix}^T \tag{4.6}$$

$$\texttt{sigmoid}(z) = \frac{1}{1 + e^{-z}} \tag{4.7}$$

$$\mathbf{a}^{[1]} = \texttt{sigmoid}(\mathbf{z}^{[1]}) = \begin{bmatrix} a_1^{[1]} & a_2^{[1]} & ... & a_{10}^{[1]} \end{bmatrix}^T \tag{4.8}$$

Equations 4.9 and 4.10 formulate the weight matrix $\mathbf{W}^{[2]}$ and the bias vector $\mathbf{b}^{[2]}$ from the hidden layer to the output layer, respectively. $n^{[2]}$ is the number of activation values in the output layer. Equation 4.11 formulates the weighted sum $\mathbf{z}^{[2]}$ from the hidden layer to the output layer, using $\mathbf{W}^{[2]}$, $\mathbf{a}^{[1]}$, and $\mathbf{b}^{[2]}$. Equation 4.12 formulates the output activation vector $\mathbf{a}^{[2]}$ from the weighted sum $\mathbf{z}^{[2]}$ using the `sigmoid()` function (Equation 4.7). Equation 4.13 formulates how to choose the proper classification $\mathbf{y}$ using the softmax algorithm on the output activation vector $\mathbf{a}^{[2]}$.

$$\mathbf{W}^{[2]} \in \mathbb{R}^{n^{[2]} * 10} \tag{4.9}$$

$$\mathbf{b}^{[2]} \in \mathbb{R}^{n^{[2]} * 1} \tag{4.10}$$

$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]} \cdot \mathbf{a}^{[1]} + \mathbf{b}^{[2]} = \begin{bmatrix} z_1^{[2]} & z_2^{[2]} & ... & z_{10}^{[2]} \end{bmatrix}^T \tag{4.11}$$

$$\mathbf{a}^{[2]} = \texttt{sigmoid}(\mathbf{z}^{[2]}) = \begin{bmatrix} a_1^{[2]} & a_2^{[2]} & ... & a_{10}^{[2]} \end{bmatrix}^T \tag{4.12}$$

$$\mathbf{y} = \text{softmax}(\mathbf{a}^{[2]}) \tag{4.13}$$

Any modelling actions will first randomly partition the input data set into the train, validate, and test sets based the ratio of 70%, 15%, and 15% respectively. Then the model will be trained with train set, where the stopping criterion is assessed with the validate set, and measure its predictive power on the test set. The model will generate confusion matrices for the train, validate, and test sets, as well for all three sets combined.

4.6 Feature Selection

The project uses the CWRU subset A to select the most discriminative feature(s) via visualisation, modelling, and combination (Figure 5).

4.6.1 Feature Selection via Visualisation

The visualisation phase of feature selection draws 7 line plots of each feature of the CWRU subset A, with the x-axis being the sample indices, and the y-axis being the feature values. Each line plot of the same feature represents 1 class label. Visually

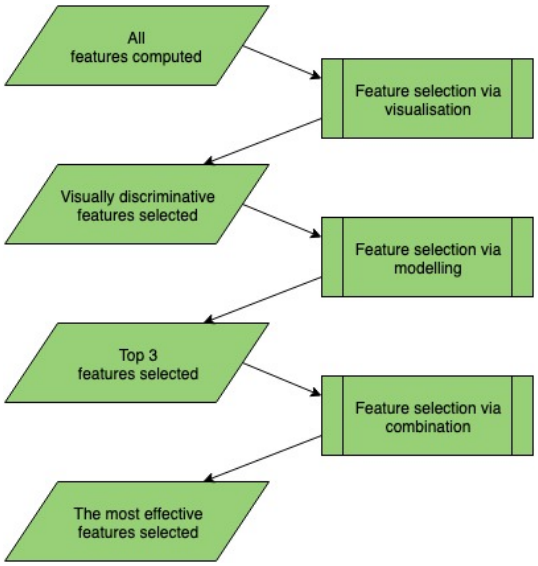

Figure 5: Block diagram of feature selection

discriminative features between all pairs of class labels are selected as candidates for further selection via modelling.

4.6.2 Feature Selection via Modelling

The modelling phase of feature selection feeds the candidate features selected from the visualisation phase to the described 3-layer ANN (Subsection 4.5) to test how the chosen classification method performs on these features. The top 3 features that yield the highest test accuracy are selected as final contenders for further selection via combination.

4.6.3 Feature Selection via Combination

The combination phase of feature selection examines whether the combination of the top 3 features ranked by the modelling phase outperforms the No. 1 candidate feature. More specifically, it stacks the vectorised representation of the top 3 features as the input, and feeds them to the 3-layer ANN (Subsection 4.5). Then, the test accuracy of the ANN based on the feature combination is compared against the test accuracy of the No. 1 candidate feature to validate the hypothesis.

4.7 Model Generalisation

The model generalisation process (Figure 6) mainly discusses two aspects: whether the feature or feature combination selected from the CWRU subset A will generalise to different data sources (Sub-subsection 4.7.1), and different load factors (Sub-subsection 4.7.2).

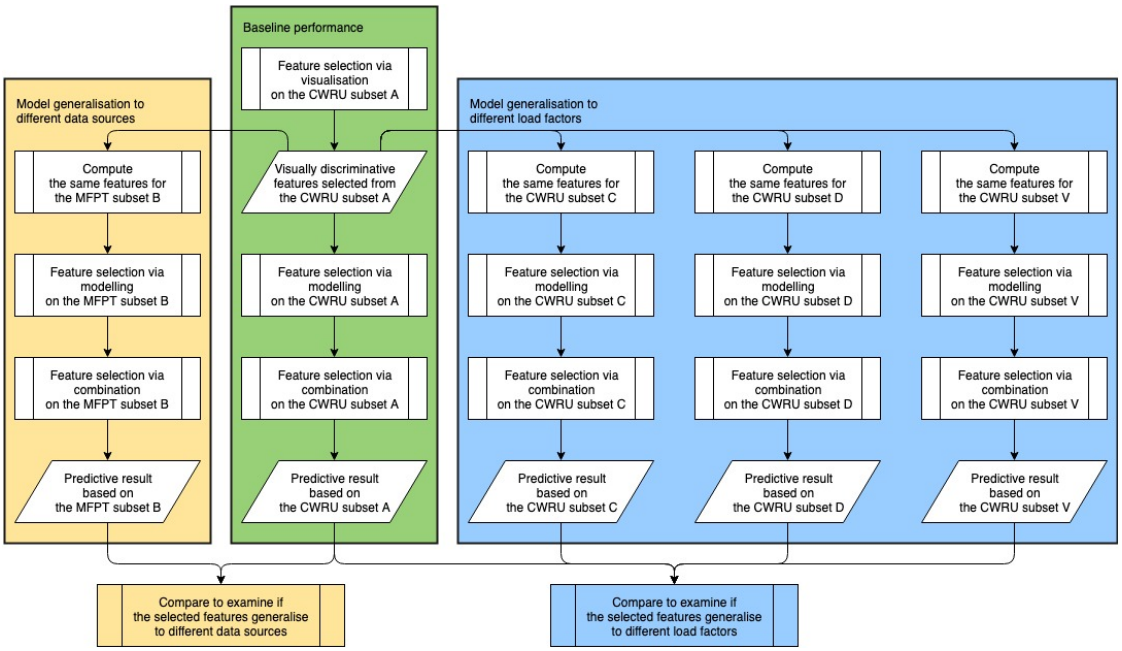

Figure 6: Block diagram of model generalisation

4.7.1 Model Generalisation to Different Data Sources

The model generalisation process first checks whether the selected features generalise to a completely different data set. This step feeds the visually discriminative features selected from the CWRU subset A, to test if the top 3 candidate features are the same for both the CWRU subset A and the MFPT subset B (Table 3). This step also compares if the combination of the top 3 features outperforms the No.1 candidate feature.

4.7.2 Model Generalisation to Different Load Factors

The next step of the model generalisation examines whether the load factors will affect the discriminativeness of the selected features. To put it in practical sense, this second step answers the question whether the fault diagnosis system needs only 1 universal model for all load factors, or 1 model for each load factor that is trained separately. This step uses the visually discriminative features selected from the CWRU subset A to compute the corresponding feature values of the CWRU subsets C, D, and V (Table 1, 2). Then, it repeats the modelling and combination phases of the features selection on the CWRU subsets C, D, and V, and compare the predictive performance against the baseline performance on the CWRU subset A.

4.8 Robustness Testing

Robustness testing in the field of software engineering examines the degree to which a software system can function as desired as a response to invalid parameters or an overloading operating environment [19]. When applying this testing process to fault diagnosis, we test the robustness of the selected features and the trained model by feeding noise-added vibration signal. The test subject that the robustness testing procedure examines is the ANN model that is trained on the most effective feature or feature combination extracted from the CWRU subset A. This testing process introduces a series of white noise with the range of $\pm 25\%$ and $\pm 50\%$ standard deviation on top of the original signals from the CWRU subset A, to create two noise-added data sets. Then, the testing procedure feeds the 2 noise-added data sets into the test subject, the trained ANN model, and assess the predictive accuracy for each noise-added data set.

5 Results

This section presents the experimental results following the procedures described in Section 4. Subsection 5.1 lists the visually discriminative features from the visualisation phase of the feature selection process. Subsection 5.2 shows the baseline performance on the CWRU subset A by feeding the visually discriminative features and their derived feature combinations into a controlled model. Subsection 5.3 discusses how well the performance of the visually discriminative features and their derived feature combinations generalise to different data sources, and different load factors. Subsection 5.4 presents the degree of robustness when dealing with noise-added data.

5.1 Visually Discriminative Features

The visually discriminative features that are identified through the visualisation phase of the feature selection process are listed below, followed by their respective abbreviations and vectorised representations:

- EEMD feature #1 (Figure 17),

$$\texttt{EEMD\_F1} = \begin{bmatrix} f_1^{[1]} & f_1^{[2]} & \dots & f_1^{[M]} \end{bmatrix} \tag{5.1}$$

- EEMD feature #5 (Figure 18, 19, 20, 21, 22),

$$\texttt{EEMD\_F5} = \begin{bmatrix} f_{z>0.0}^{[1]} & f_{z>0.0}^{[2]} & \cdots & f_{z>0.0}^{[M]} \\ f_{z>0.5}^{[1]} & f_{z>0.5}^{[2]} & \cdots & f_{z>0.5}^{[M]} \\ f_{z>1.0}^{[1]} & f_{z>1.0}^{[2]} & \cdots & f_{z>1.0}^{[M]} \\ f_{z>1.5}^{[1]} & f_{z>1.5}^{[2]} & \cdots & f_{z>1.5}^{[M]} \\ f_{z>2.0}^{[1]} & f_{z>2.0}^{[2]} & \cdots & f_{z>2.0}^{[M]} \\ f_{avg}^{[1]} & f_{avg}^{[2]} & \cdots & f_{avg}^{[M]} \end{bmatrix} \tag{5.2}$$

- EEMD feature #6 (Figure 23, 24, 25, 26, 27),

$$\texttt{EEMD\_F6} = \begin{bmatrix} f_{z>0.0}^{[1]} & f_{z>0.0}^{[2]} & \cdots & f_{z>0.0}^{[M]} \\ f_{z>0.5}^{[1]} & f_{z>0.5}^{[2]} & \cdots & f_{z>0.5}^{[M]} \\ f_{z>1.0}^{[1]} & f_{z>1.0}^{[2]} & \cdots & f_{z>1.0}^{[M]} \\ f_{z>1.5}^{[1]} & f_{z>1.5}^{[2]} & \cdots & f_{z>1.5}^{[M]} \\ f_{z>2.0}^{[1]} & f_{z>2.0}^{[2]} & \cdots & f_{z>2.0}^{[M]} \\ f_{avg}^{[1]} & f_{avg}^{[2]} & \cdots & f_{avg}^{[M]} \end{bmatrix} \tag{5.3}$$

- statistical feature #14 (Figure 28, 29, 30, 31, 32),

$$\texttt{stats\_F14} = \begin{bmatrix} f_{z>0.0}^{[1]} & f_{z>0.0}^{[2]} & \cdots & f_{z>0.0}^{[M]} \\ f_{z>0.5}^{[1]} & f_{z>0.5}^{[2]} & \cdots & f_{z>0.5}^{[M]} \\ f_{z>1.0}^{[1]} & f_{z>1.0}^{[2]} & \cdots & f_{z>1.0}^{[M]} \\ f_{z>1.5}^{[1]} & f_{z>1.5}^{[2]} & \cdots & f_{z>1.5}^{[M]} \\ f_{z>2.0}^{[1]} & f_{z>2.0}^{[2]} & \cdots & f_{z>2.0}^{[M]} \\ f_{avg}^{[1]} & f_{avg}^{[2]} & \cdots & f_{avg}^{[M]} \end{bmatrix} \tag{5.4}$$

- statistical feature #19 (Figure 33, 34, 35, 36, 37).

$$\texttt{stats\_F19} = \begin{bmatrix} f_{z>0.0}^{[1]} & f_{z>0.0}^{[2]} & \cdots & f_{z>0.0}^{[M]} \\ f_{z>0.5}^{[1]} & f_{z>0.5}^{[2]} & \cdots & f_{z>0.5}^{[M]} \\ f_{z>1.0}^{[1]} & f_{z>1.0}^{[2]} & \cdots & f_{z>1.0}^{[M]} \\ f_{z>1.5}^{[1]} & f_{z>1.5}^{[2]} & \cdots & f_{z>1.5}^{[M]} \\ f_{z>2.0}^{[1]} & f_{z>2.0}^{[2]} & \cdots & f_{z>2.0}^{[M]} \\ f_{avg}^{[1]} & f_{avg}^{[2]} & \cdots & f_{avg}^{[M]} \end{bmatrix} \tag{5.5}$$

These 5 visually discriminative features are the candidates that will proceed to the modelling and combination phase of the feature selection process.

5.2 Baseline Performance

The baseline performance (Green row in Table 4) is the performance of the visually discriminative features from the CWRU subset A that are fed into the 3-layer ANN with 10 hidden units. Due to its time-domain property, the modelling on the `EEMD_F1` feature, extracted from CWRU data, will use the model architecture in Figure 7; due to their frequency-domain property, the modelling on the `EEMD_F5`/`EEMD_F6`/`stats_F14`/`stats_F19` feature, extracted from CWRU data, will use the model architecture in Figure 8. Based on the separate modelling performance of each candidate feature, the top 3 candidate features are EEMD feature #5, EEMD feature #6, and statistical feature #14. The combination phase creates the combination of these top 3 candidate features, by stacking the vectorised representation of the top 3 candidate features, to create `top3_combo` (Equation 5.6). Because the `EEMD_F5`, and `EEMD_F6`, and `stats_F14` are all in the frequency domain, the modelling on the `top3_combo` feature combination, extracted from CWRU data, will use the model architecture in Figure 9. Based on baseline performance, it is still inconclusive which feature or feature combination yields the best performance, since statistical feature #14 and `top3_combo` both produces 100% test accuracy.

$$\texttt{top3\_combo} = \begin{bmatrix} \texttt{EEMD\_F5} \\ \texttt{EEMD\_F6} \\ \texttt{stats\_F14} \end{bmatrix} \tag{5.6}$$

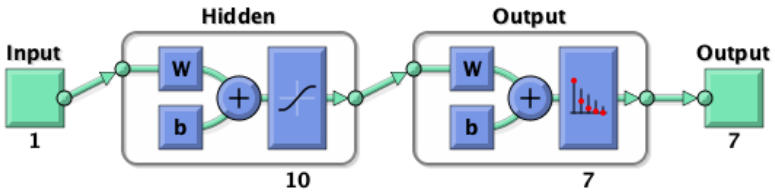

Figure 7: Block diagram of a 3-layer ANN model with 1 input neuron, 10 hidden neurons, and 7 output neurons, using the time-domain `EEMD_F1` feature, extracted from CWRU data

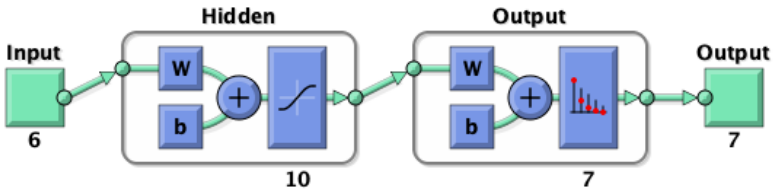

Figure 8: Block diagram of a 3-layer ANN model with 1 input neuron, 10 hidden neurons, and 7 output neurons, using the frequency-domain `EEMD_F5` / `EEMD_F6` / `stats_F14` / `stats_F19` feature, extracted from CWRU data

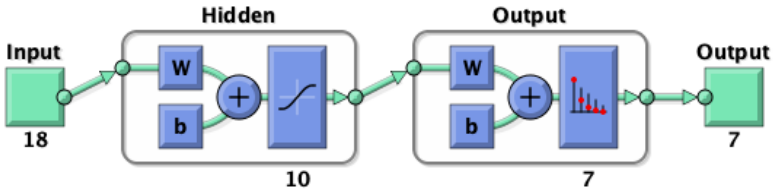

Figure 9: Block diagram of a 3-layer ANN model with 1 input neuron, 10 hidden neurons, and 7 output neurons, using the `top3_combo` feature combination, extracted from CWRU data

5.3 Generalisation Performance

The generalisation performance (Yellow and blue rows Table 4) is the performance of generalisation to different data sources, and different load factors, using the 5 visually discriminative features (Subsection 5.1) separately, and the `top3_combo` feature combination. Due to its time-domain property, the modelling on the `EEMD_F1` feature, extracted from MFPT data, will use the model architecture in Figure 10; due to their frequency-domain property, the modelling on the `EEMD_F5`/`EEMD_F6`/`stats_F14`/`stats_F19` feature, extracted from MFPT data, will use the model architecture in Figure 11; because the

EEMD_F5, and EEMD_F6, and stats_F14 are all in the frequency domain, the modelling on the top3_combo feature combination, extracted from CWRU data, will use the model architecture in Figure 12.. For any data sets, the top3_combo feature combination performs the best among all features and feature combinations. When comparing the the performance on the CWRU subset A, and that on the MFPT subset B, the top3_combo feature combination yields 100% test accuracy for both data sets. Therefore, it concludes that the top3_combo feature combination can generalise to different data sources. When comparing the performance on the CWRU subset A, C, D, and V, the CWRU subset V underperforms any other CWRU subsets. Therefore, it is discouraged to categorise the fault types of vibration signals under different load factor. In practical terms, it is preferred that the proposed diagnosis system should train different models for different load factors.

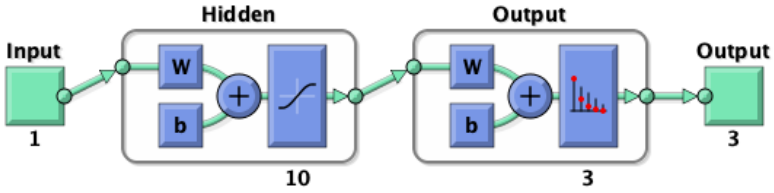

Figure 10: Block diagram of a 3-layer ANN model with 1 input neuron, 10 hidden neurons, and 7 output neurons, using the time-domain EEMD_F1 feature, extracted from MFPT data

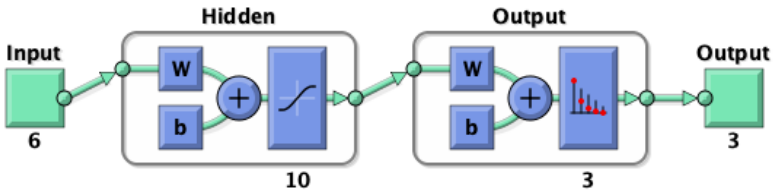

Figure 11: Block diagram of a 3-layer ANN model with 1 input neuron, 10 hidden neurons, and 7 output neurons, using the frequency-domain EEMD_F5 / EEMD_F6 / stats_F14 / stats_F19 feature, extracted from MFPT data

5.4 Robustness Performance

The robustness performance is the performance of robustness testing using the ±25%-noise-added and ±50%-noise-added data sets. According to the findings from the baseline performance and the generalisation performance, the top3_combo (Equation 5.6) feature combination is the most effective among all candidates. The robustness testing reports 100.0% and 93.9% predictive accuracy on the the ±25%-noise-added and ±50%-noise-added data sets, respectively. Therefore, the top3_combo feature combination not

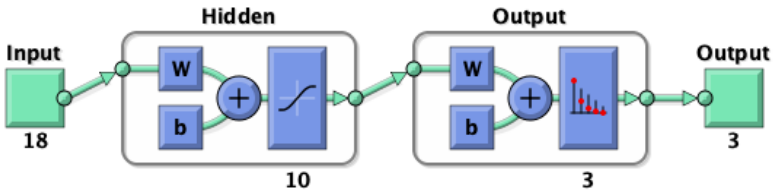

Figure 12: Block diagram of a 3-layer ANN model with 1 input neuron, 10 hidden neurons, and 7 output neurons, using the `top3_combo` feature combination, extracted from MFPT data

Table 4: Testing accuracy of visually discriminative features and combinations of top 3 candidate features on different data subsets

	Features or feature combinations					
Data	`EEMD_F1`	`EEMD_F5`	`EEMD_F6`	`stats_F14`	`stats_F19`	`top3_combo`
CWRU subset A	90.5%	98.1%	98.1%	100.0%	92.4%	100.0%
MFPT subset B	55.6%	90.7%	90.7%	87.0%	79.6%	100.0%
CWRU subset C	84.8%	99.0%	99.0%	96.2%	82.9%	100.0%
CWRU subset D	85.7%	94.3%	99.0%	99.0%	89.5%	100.0%
CWRU subset V	88.3%	84.4%	90.8%	91.4%	86.3%	97.5%

only has a high degree of generalisation, but also excels on the scale of robustness.

5.5 Proposed Diagnosis System

Based on the above findings on the effectiveness of candidates features, this subsection proposes a method to construct an automated diagnosis system based on the `top3_combo` feature combination in Sub-section 5.5.1, and describes the workflow of the proposed system to diagnose a new series of vibration signal in Sub-subsection 5.5.2.

5.5.1 Construction of the System

The construction of the proposed diagnosis system (Figure 13) starts with the collection of experimental data of vibration signals that cover the full spectrum of classes the system is supposed to identify. The proposed method extracts the `top3_combo` feature combination. The next step decides how many models the system needs, which corresponds to the number of intervals of load factors the system operates on. Figure 14 demonstrates the process of choosing the proper number of models in the systems: if the domain of load factors is discrete, then the number of models equals the number of discrete load factors; if the domain of load factors is finitely continuous, then the builder of the system should choose an arbitrary number of models for equal-length intervals of load factors; if the domain of the load factors is infinitely continuous, then the system

only needs 1 universal model for any value of load factors. Once the number of load factor intervals R is confirmed, the system partitions the extracted features of the experimental data into their appropriate load factor intervals. The system then trains a 3-layer ANN with 10 hidden units for each of the R load factor intervals based on their corresponding data partition. The outputs of this construction process are the R trained models, each of which corresponds to one of the R load factor intervals

5.5.2 Workflow of the System

The workflow of the proposed diagnosis system (Figure 15) starts with the measurement of the load factor, and the collection of the vibration signal series. The system picks the appropriate model to make the fault categorisation according to which load factor interval the measured load factor falls into. After the choice of the model, the system extracts the `top3_combo` feature combination from the collected vibration signal series. The system then feeds the combination of the extracted features into the chosen model, and predicts the fault category.

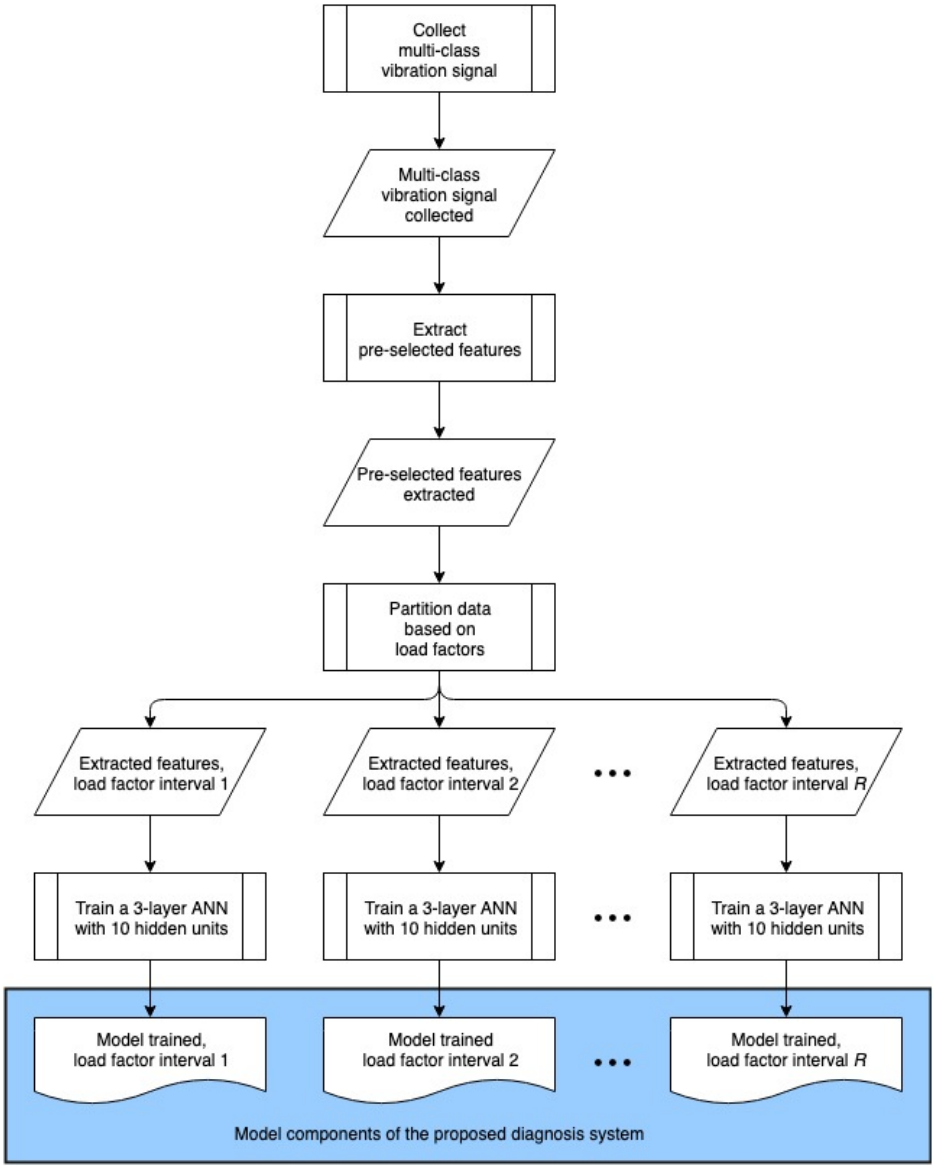

Figure 13: Construction of the proposed diagnosis system

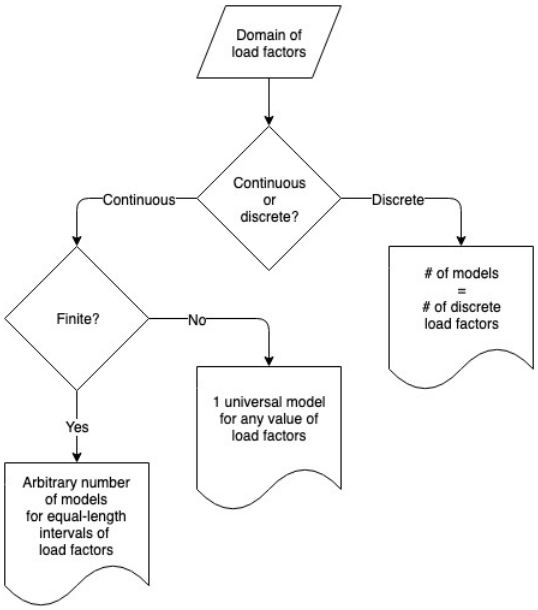

Figure 14: Process of choosing the number of models

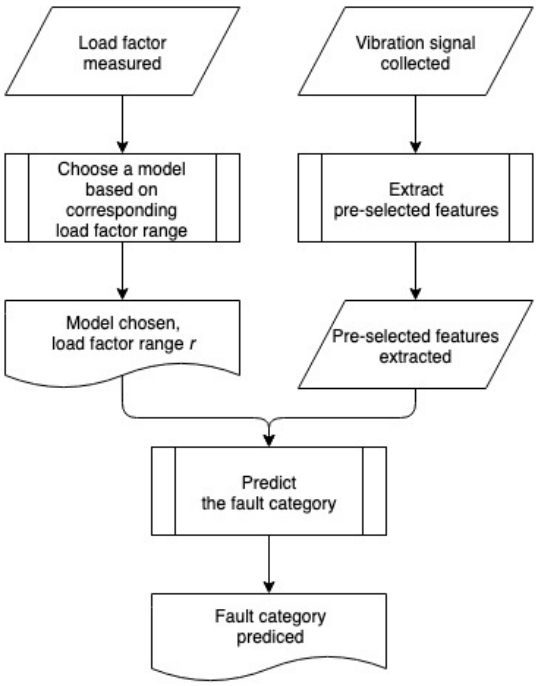

Figure 15: Workflow of the proposed diagnosis system

6 Conclusion

Most of the previous researches on the topic of intelligent fault diagnosis system focus on finding an effective feature extraction methods, and achieving accurate prediction using shallow classification methods. This paper compares the effectiveness of different feature extraction methods in a controlled experimental setting, specified by a 3-layer ANN model with 10 hidden units for any selected features. The experiment shows that the combination of a few certain features from different feature extraction methods can achieve accurate prediction, and can generalise well to different data sources as well as different load factors. Furthermore, this feature combination can perfectly withstand $\pm 25\%$ noise, and perform fairly accurately on $\pm 50\%$-noise-added data. Based on these findings, this paper proposes a multi-model fault diagnosis system that uses the measured load factor to choose the appropriate model component, and feeds the selected features combination of the new vibration signal sample into the chosen model to categorise its fault type. Further research can investigate how to implement a universal model for the entire spectrum of operational load factors for the aim of simplicity of the system architecture.

References

[1] The MathWorks, Inc., "Fast fourier transform." `https://uk.mathworks.com/help/matlab/ref/fft.html`, 2019. [Online].

[2] H. Jiang, C. Li, and H. Li, "An improved eemd with multiwavelet packet for rotating machinery multi-fault diagnosis," *Mechanical Systems and Signal Processing*, vol. 36, p. 225–239, 04 2013.

[3] Y. Lei, Z. He, and Y. Zi, "Application of a novel hybrid intelligent method to compound fault diagnosis of locomotive roller bearings," *Journal of Vibration and Acoustics*, vol. 130, 06 2008.

[4] XiaoLi Zhang, Baojian Wang, XueFeng Chen, "Intelligent fault diagnosis of roller bearings with multivariable ensemble-based incremental support vector machine," *Knowlege-Based System*, vol. 89, pp. 56–85, 2015.

[5] Yaguo Lei, Zhengjia He, Yanyang Zi, "EEMD method and WNN for fault diagnosis of locomotive roller bearings," *Expert Systems with Applications*, vol. 38, pp. 7334–7341, 2011.

[6] B. Boashash, *Time frequency signal analysis and processing : a comprehensive reference*. Amsterdam Boston: Elsevier, 2003.

[7] R. Bracewell, *The Fourier transform and its applications*. Boston: McGraw Hill, 2000.

[8] J. W. Cooley, P. A. W. Lewis, Peter, and D. Welch, "Historical notes on the fast fourier transform," *IEEE Trans. Audio Electroacoust*, 1967.

[9] Y. Z. Q. H. Yaguo Lei, Zhengjia He, "Fault diagnosis of rotating machinery based on multiple ANFIS combination with GAs," *Mechanical Systems and Signal Processing*, vol. 21, no. 5, pp. 2280 – 2294, 2007.

[10] N. Huang, Z. Shen, S. Long, M. Wu, H. Shih, Q. Zheng, N. Yen, C.-C. Tung, and H. Liu, "The empirical mode decomposition and the hilbert spectrum for nonlinear and non-stationary time series analysis," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 454, pp. 903–995, 03 1998.

[11] Yang Yu, YuDejie, Cheng Junsheng, "A roller bearing fault diagnosis method based on EMD energy entropy and ANN," *Journal of Sound and Vibration*, vol. 294, pp. 269–277, 2006.

[12] Z. Wu and N. Huang, "Ensemble empirical mode decomposition: a noise-assisted data analysis method," *Advances in Adaptive Data Analysis*, vol. 1, pp. 1–41, 01 2009.

[13] S. Russell, *Artificial intelligence : a modern approach*. Upper Saddle River, New Jersey: Prentice Hall, 2010.

[14] Huanhuan Liu, Minghong Han, "A fault diagnosis method based on local mean decomposition and multi-scale entropy for roller bearings," *Mechanism and Machine Theory*, vol. 75, pp. 67–78, 2014.

[15] Bubathi Muruganatham, M.A. Sanjith, B. Krishnakumar, S.A.V. SSatya Murty, "Roller element bearing fault diagnosis using singular spectrum analysis," *Mechanical Systems and Signal Processing*, vol. 2013, pp. 150–166, 2013.

[16] Hussein Al-Bugharbee, Irina Trendafilova, "A fault diagnosis methodology for rolling element bearings based on advanced signal pretreatment and autoregressive modelling," *Journal of Sound and Vibration*, vol. 369, pp. 246–265, 2016.

[17] Case Western Reserve Univeristy Bearing Data Center, "Seeded fault test data." `https://csegroups.case.edu/bearingdatacenter/home`. [Online].

[18] Society For Machinery Failure Prevention Technology, "Fault data set." `https://mfpt.org/fault-data-sets/`, 2019. [Online].

[19] IEEE, "IEEE standard glossary of software engineering terminology," *IEEE Std 610.12-1990*, pp. 1–84, Dec 1990.

A Appendix - Block diagram of the entire process of experimentation

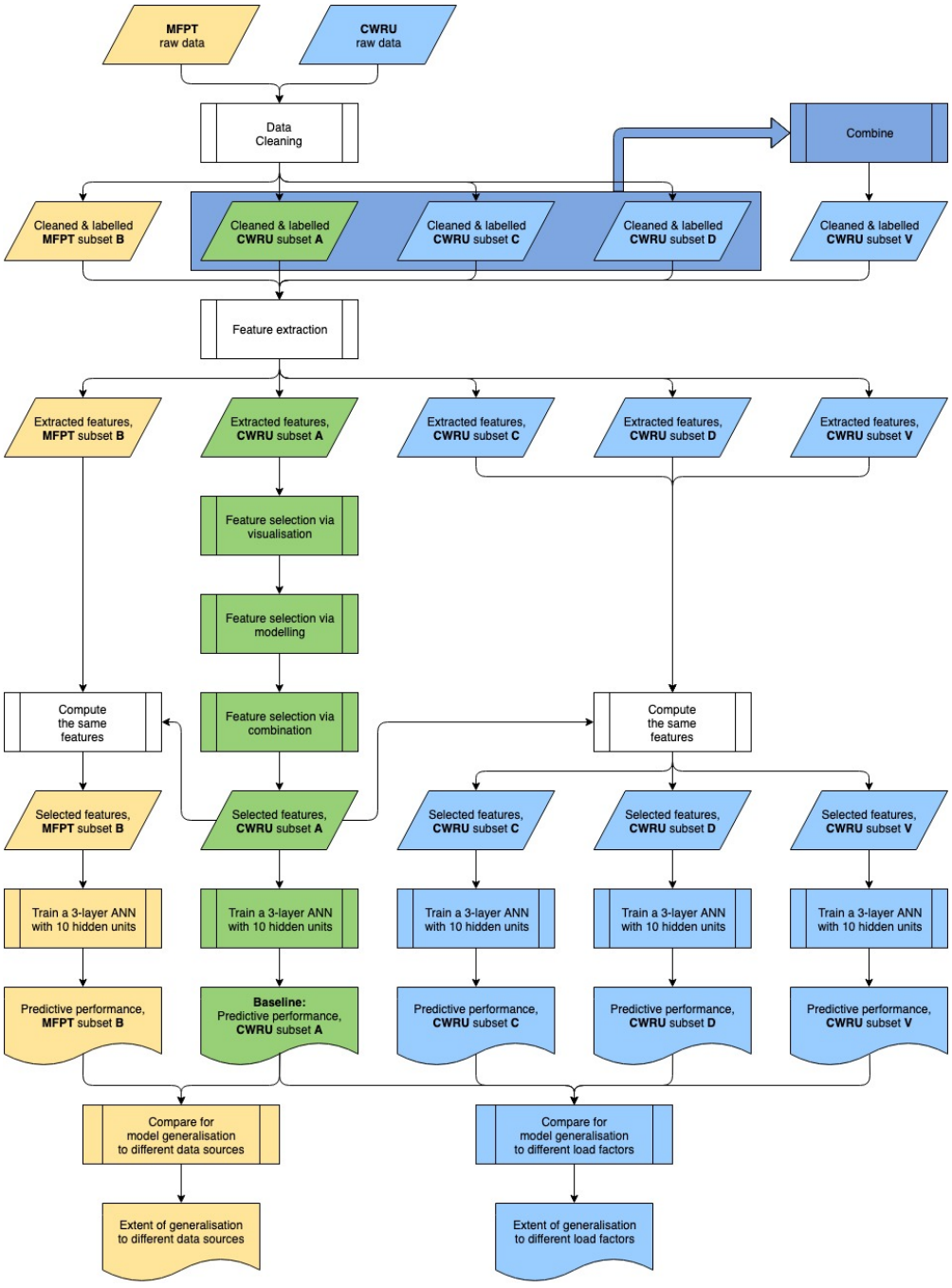

Figure 16: Block diagram of the entire process of experimentation

B Appendix - Visualisation of selected features on CWRU subset A

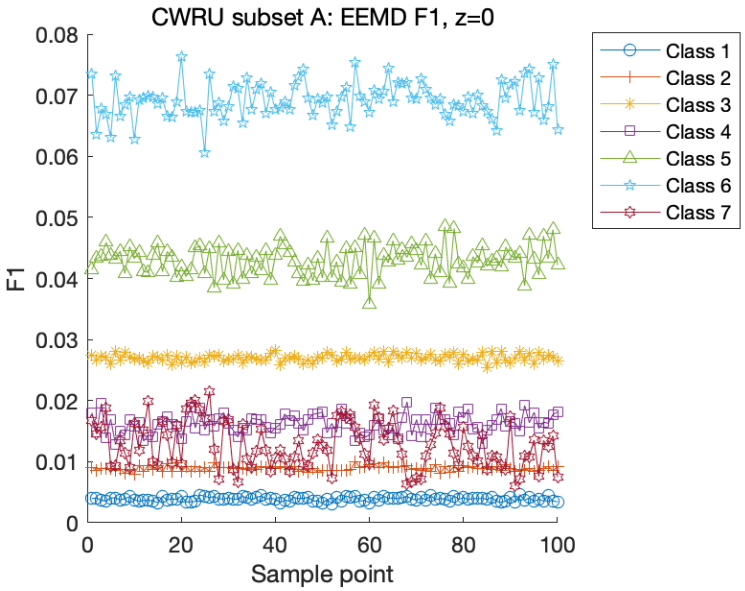

Figure 17: Discriminativeness of EEMD feature #1 on CWRU subset A

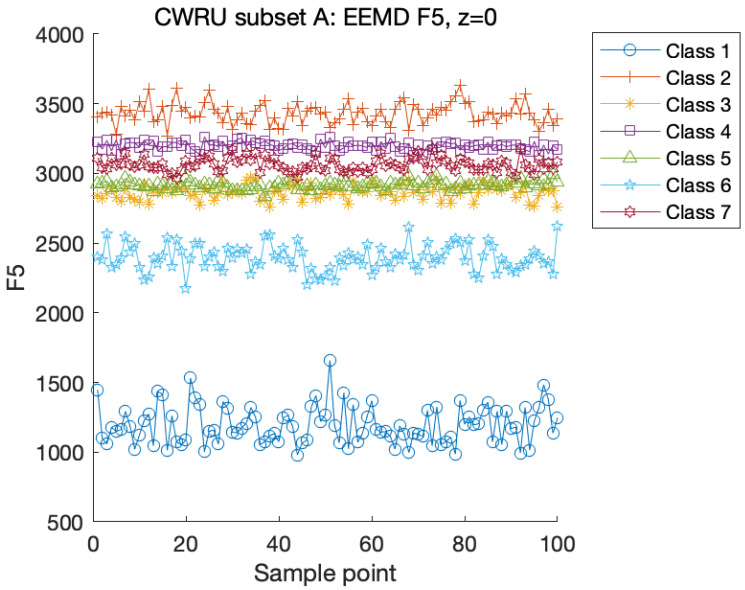

Figure 18: Discriminativeness of EEMD feature #5 with z-score=0.0 on CWRU subset A

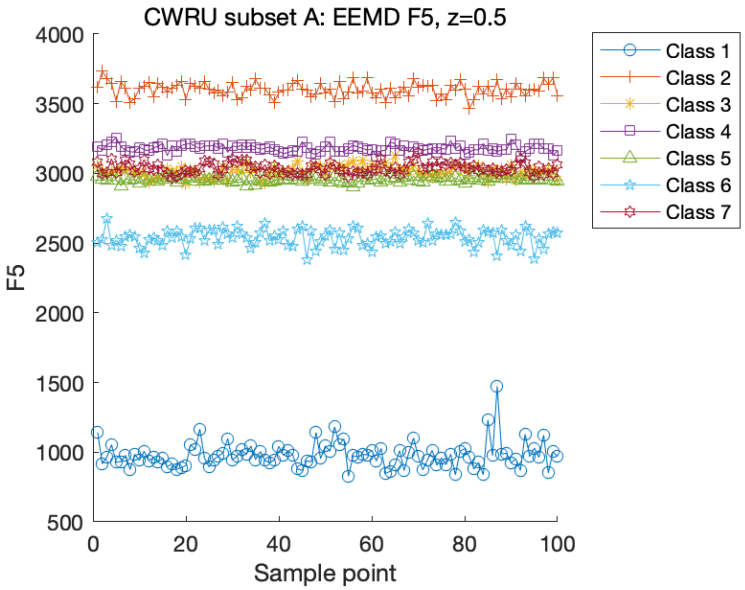

Figure 19: Discriminativeness of EEMD feature #5 with z-score=0.5 on CWRU subset A

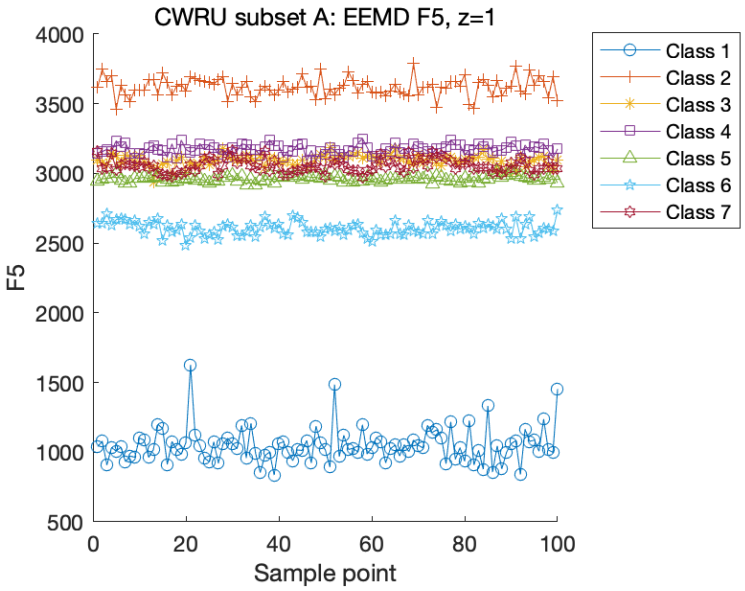

Figure 20: Discriminativeness of EEMD feature #5 with z-score=1.0 on CWRU subset A

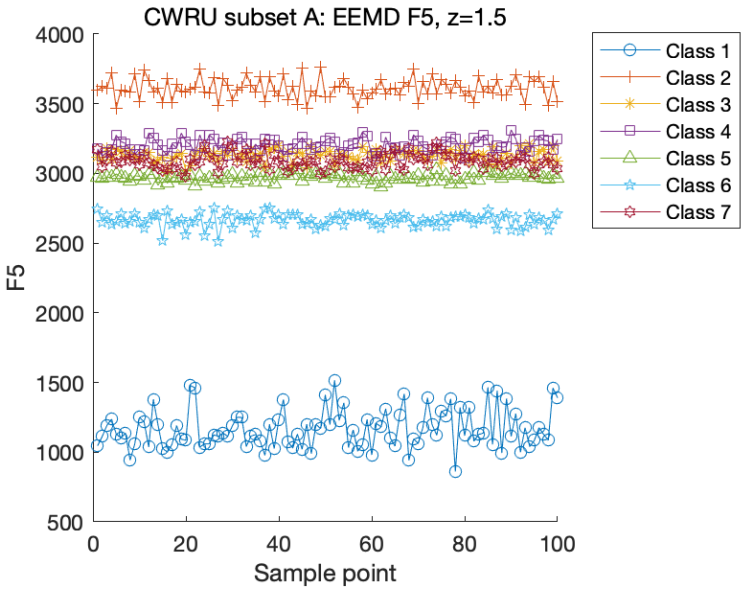

Figure 21: Discriminativeness of EEMD feature #5 with z-score=1.5 on CWRU subset A

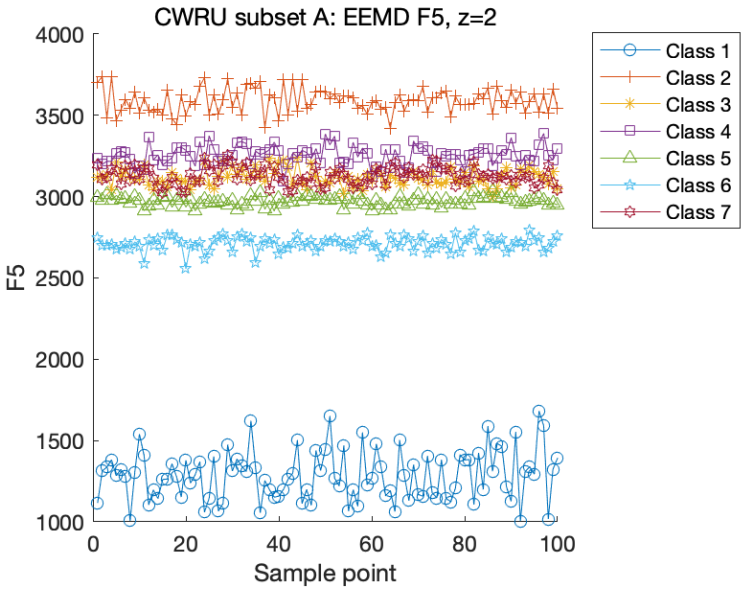

Figure 22: Discriminativeness of EEMD feature #5 with z-score=2.0 on CWRU subset A

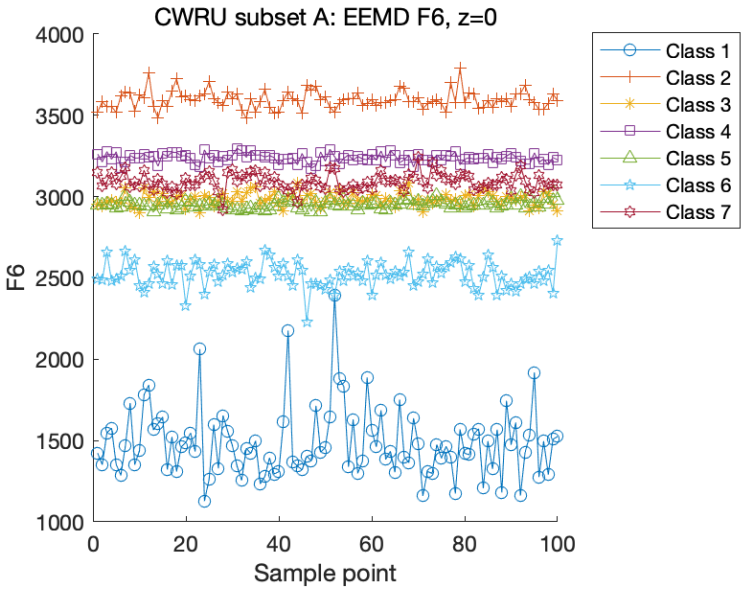

Figure 23: Discriminativeness of EEMD feature #6 with z-score=0.0 on CWRU subset A

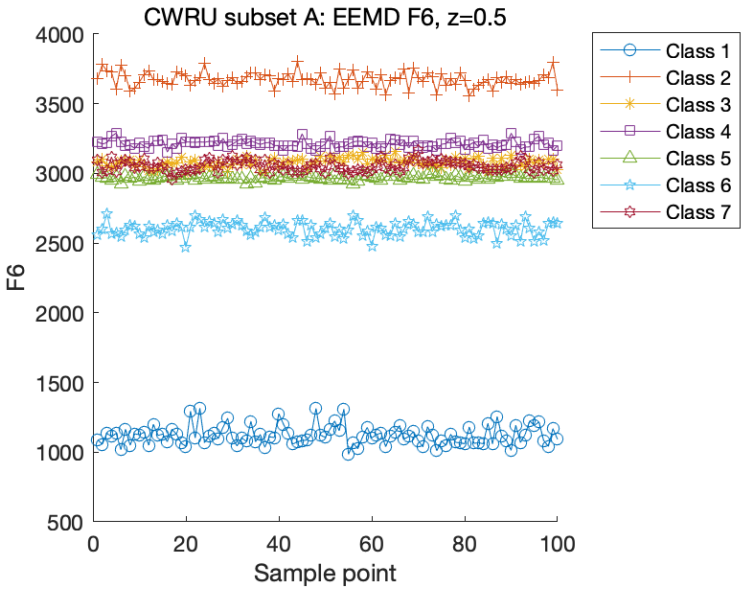

Figure 24: Discriminativeness of EEMD feature #6 with z-score=0.5 on CWRU subset A

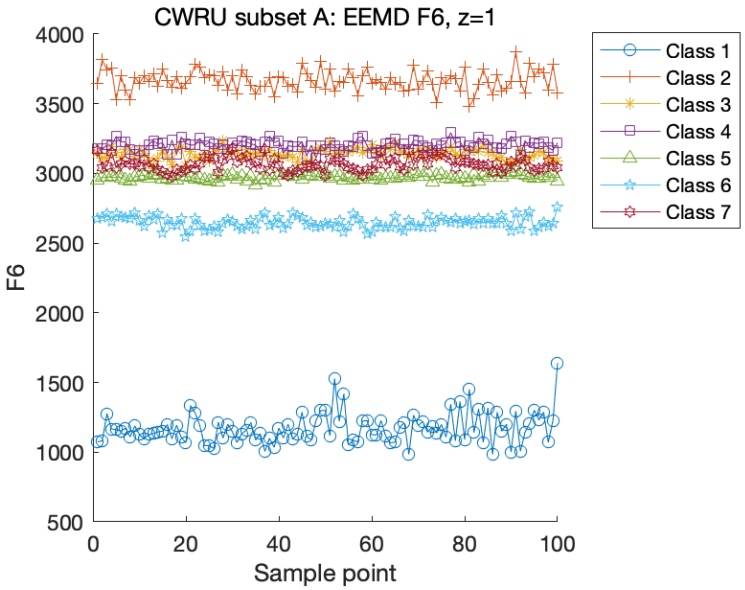

Figure 25: Discriminativeness of EEMD feature #6 with z-score=1.0 on CWRU subset A

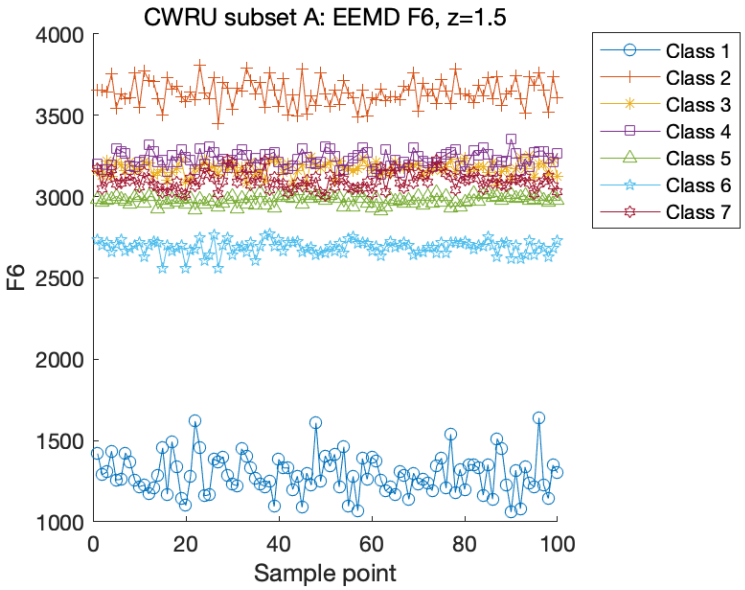

Figure 26: Discriminativeness of EEMD feature #6 with z-score=1.5 on CWRU subset A

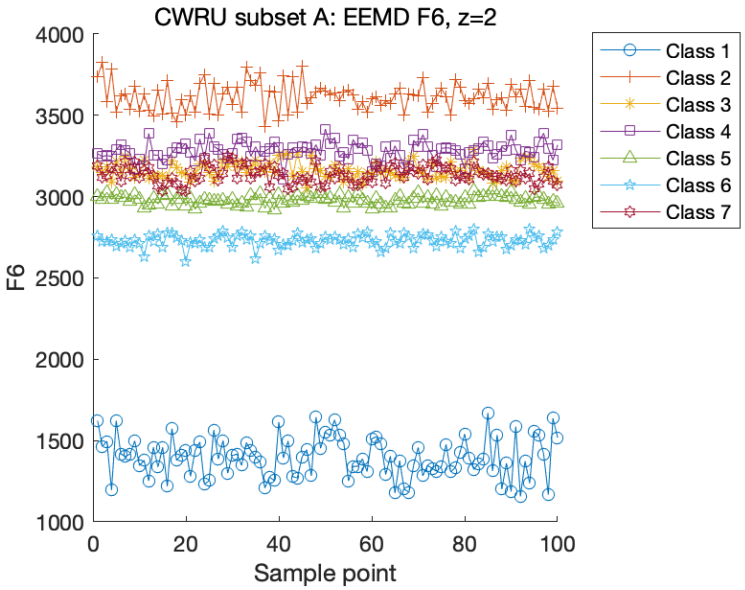

Figure 27: Discriminativeness of EEMD feature #6 with z-score=2.0 on CWRU subset A

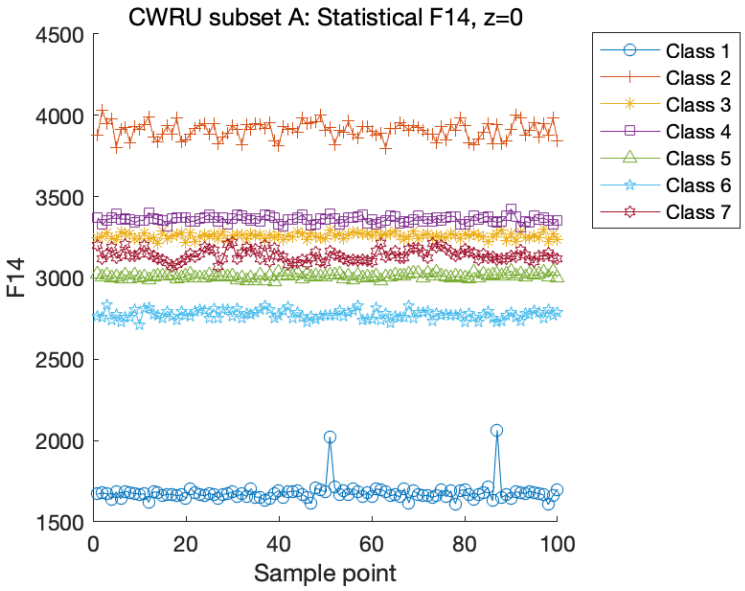

Figure 28: Discriminativeness of statistical feature #14 with z-score=0.0 on CWRU subset A

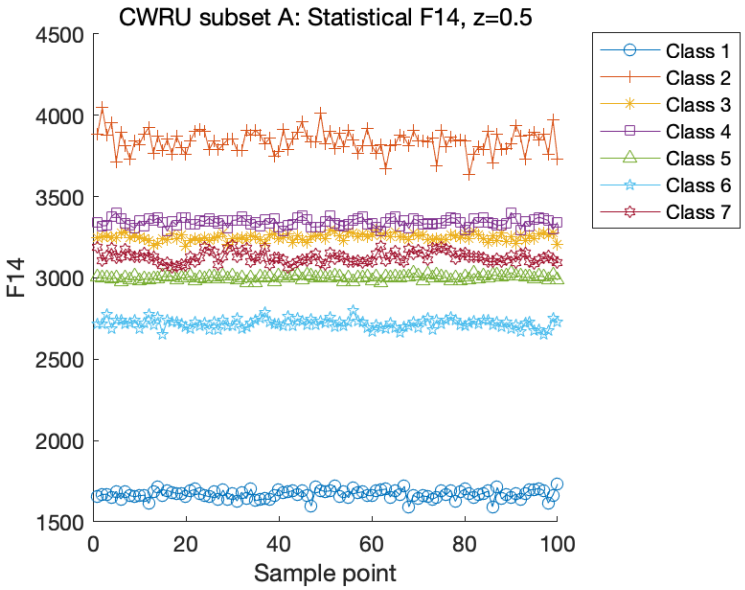

Figure 29: Discriminativeness of statistical feature #14 with z-score=0.5 on CWRU subset A

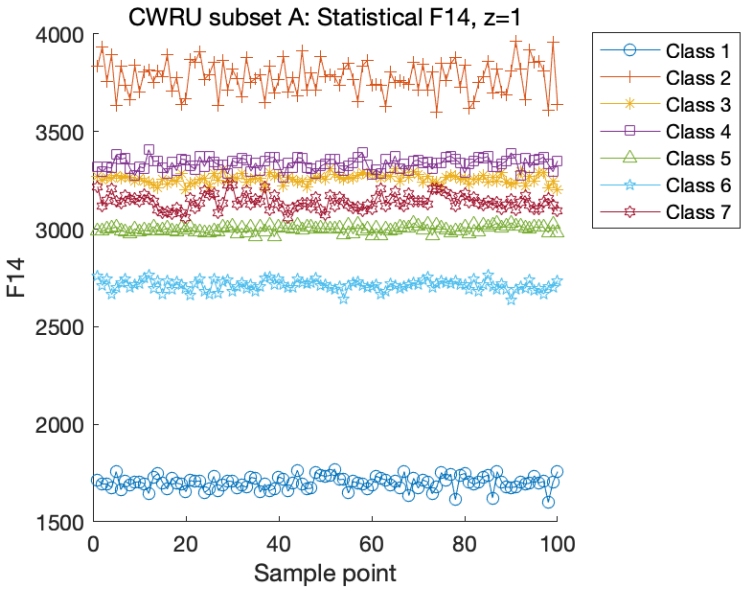

Figure 30: Discriminativeness of statistical feature #14 with z-score=1.0 on CWRU subset A

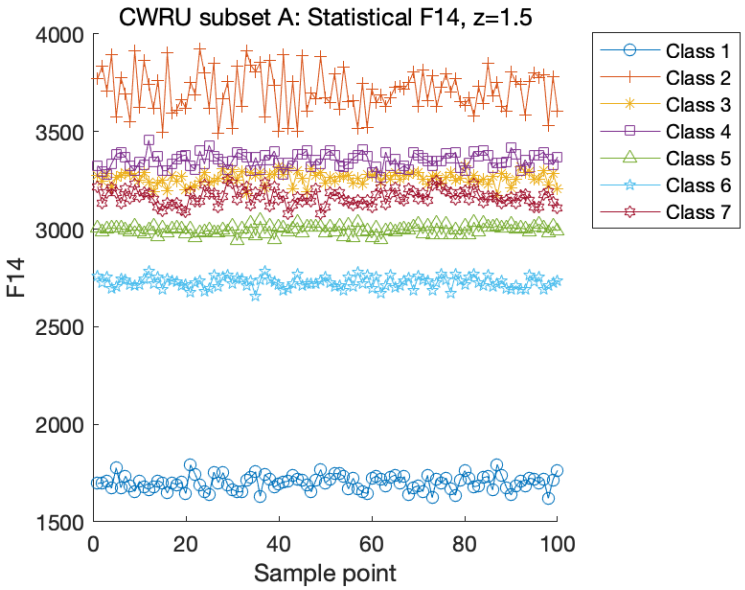

Figure 31: Discriminativeness of statistical feature #14 with z-score=1.5 on CWRU subset A

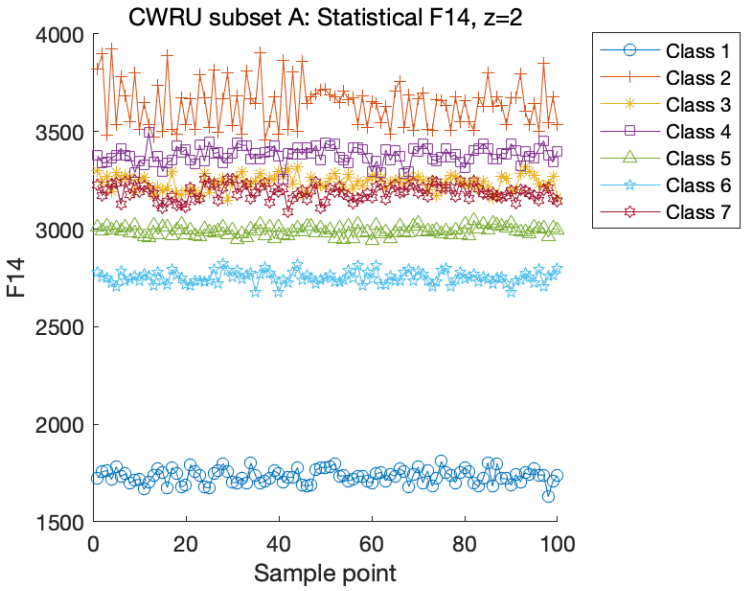

Figure 32: Discriminativeness of statistical feature #14 with z-score=2.0 on CWRU subset A

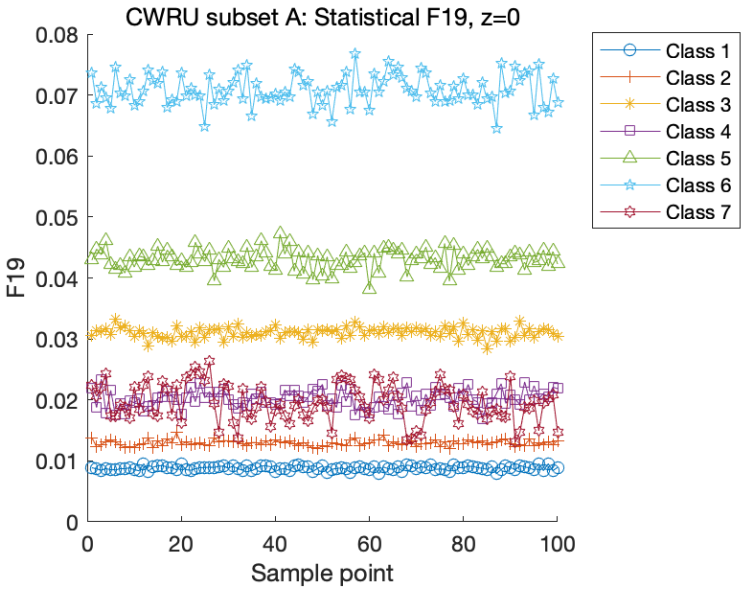

Figure 33: Discriminativeness of statistical feature #19 with z-score=0.0 on CWRU subset A

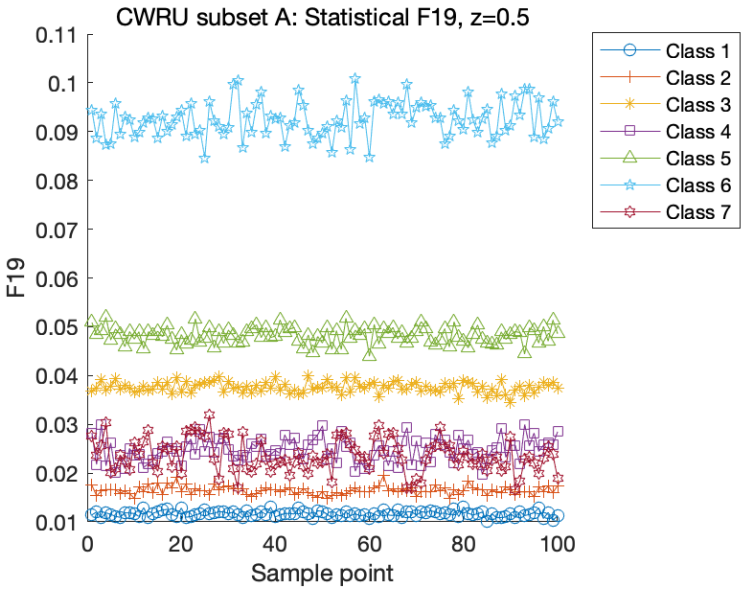

Figure 34: Discriminativeness of statistical feature #19 with z-score=0.5 on CWRU subset A

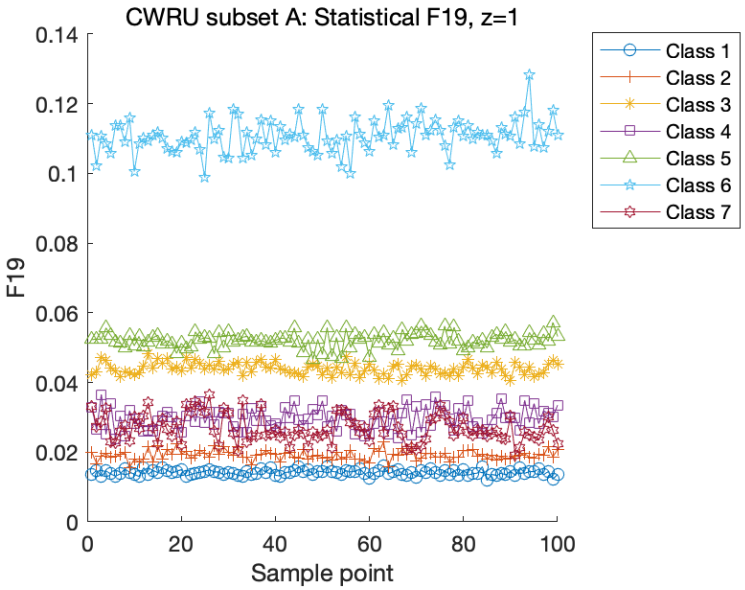

Figure 35: Discriminativeness of statistical feature #19 with z-score=1.0 on CWRU subset A

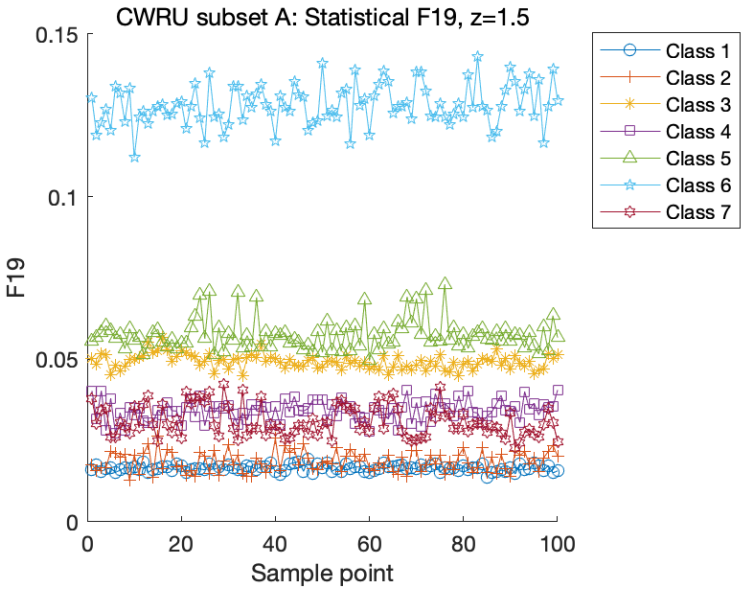

Figure 36: Discriminativeness of statistical feature #19 with z-score=1.5 on CWRU subset A

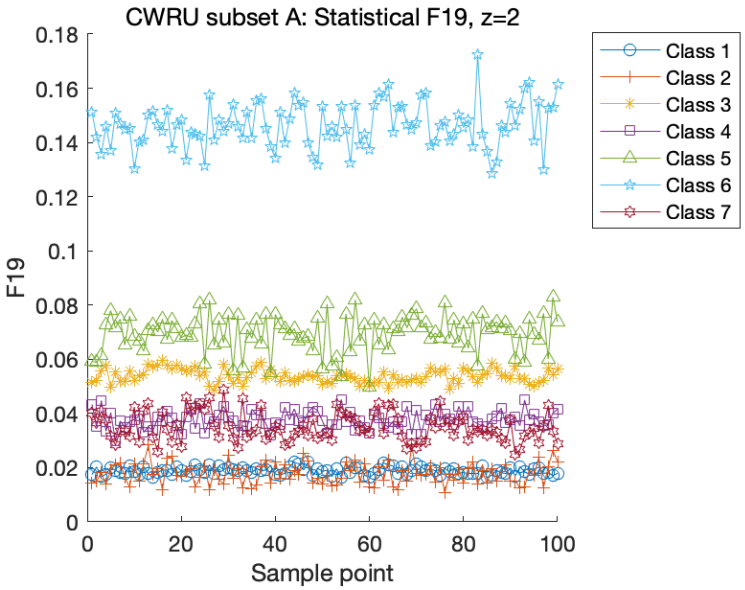

Figure 37: Discriminativeness of statistical feature #19 with z-score=2.0 on CWRU subset A

C Appendix - Confusion matrices of selected features on CWRU subset A

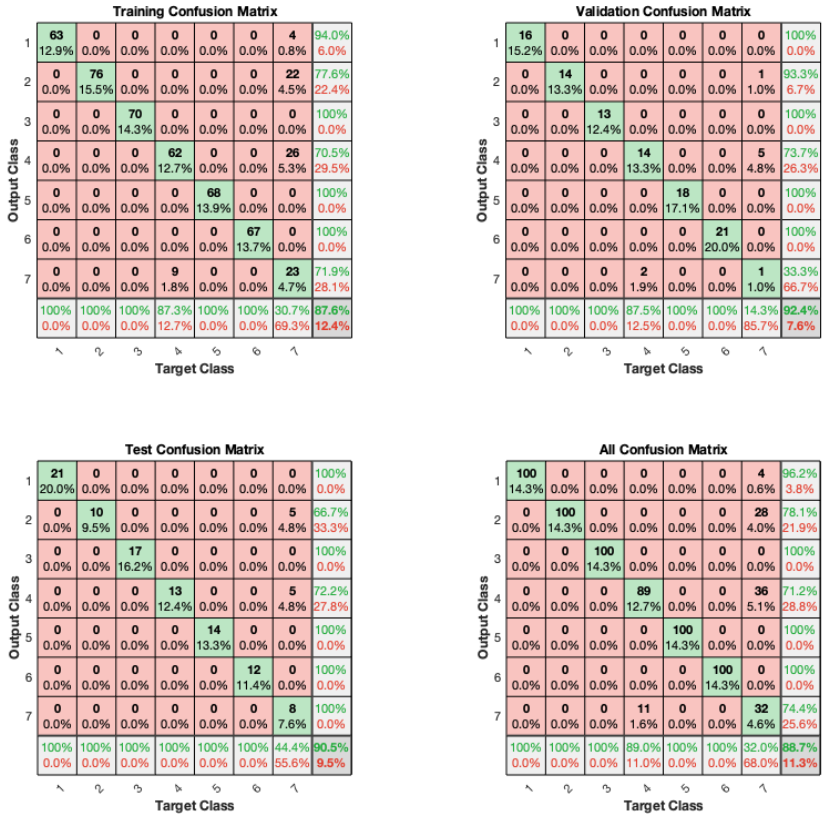

Figure 38: Confusion matrices of the train, validate, and test partitions of EEMD feature #1 on the CWRU subset A

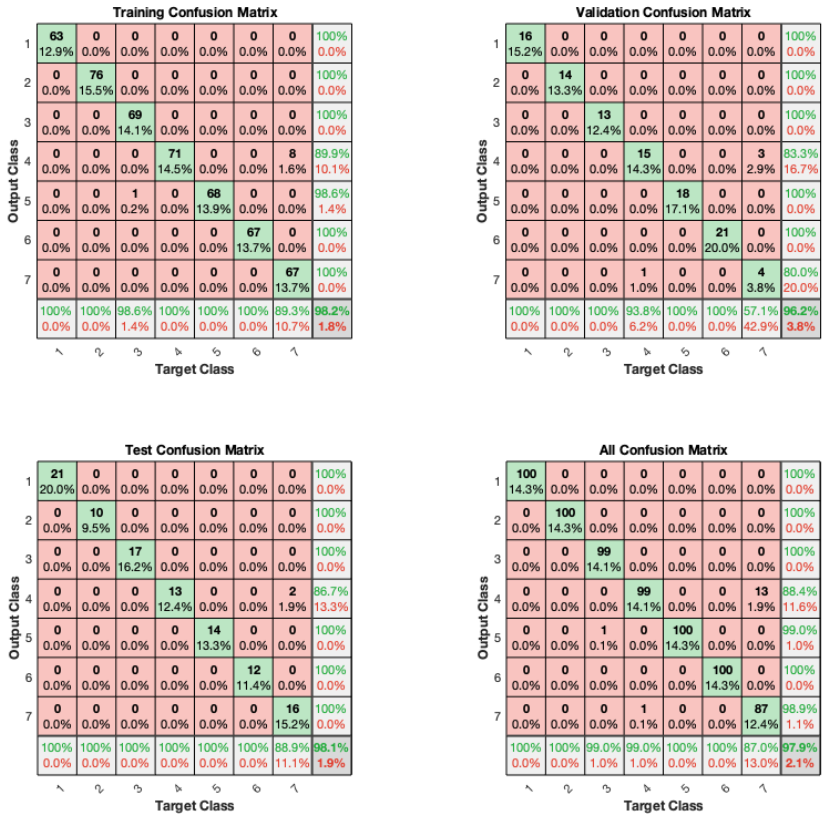

Figure 39: Confusion matrices of the train, validate, and test partitions of EEMD feature #5 on the CWRU subset A

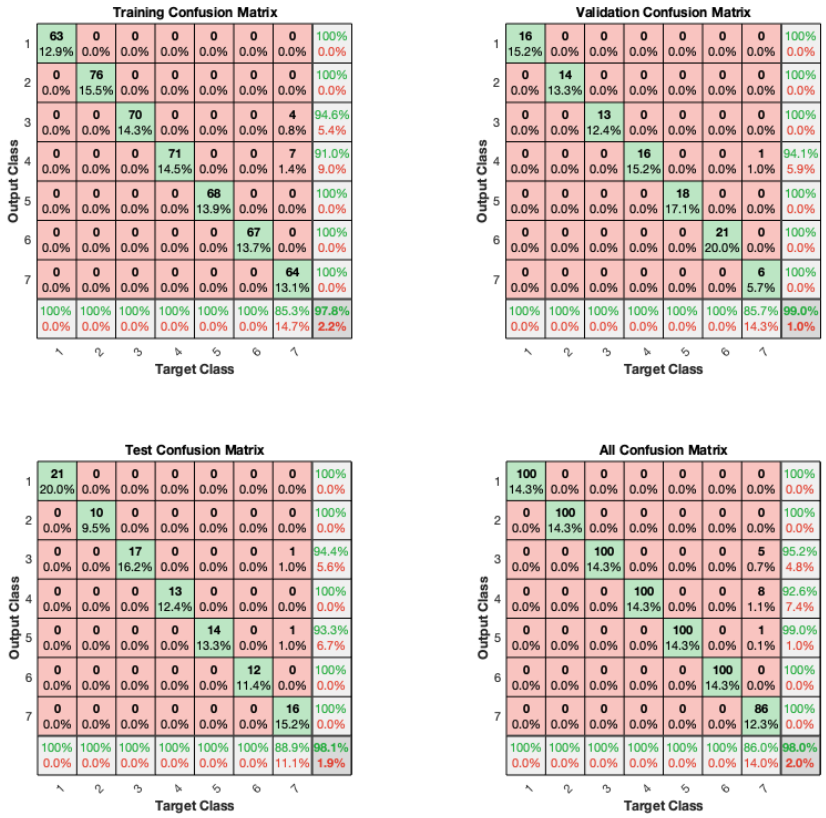

Figure 40: Confusion matrices of the train, validate, and test partitions of EEMD feature #6 on the CWRU subset A

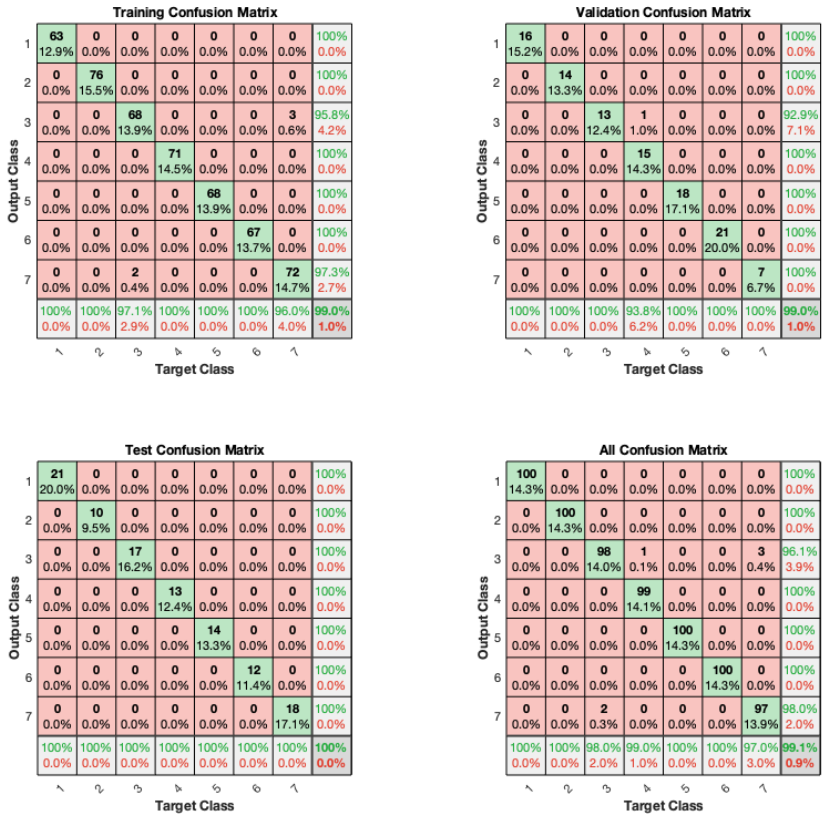

Figure 41: Confusion matrices of the train, validate, and test partitions of Statistical feature #14 on the CWRU subset A

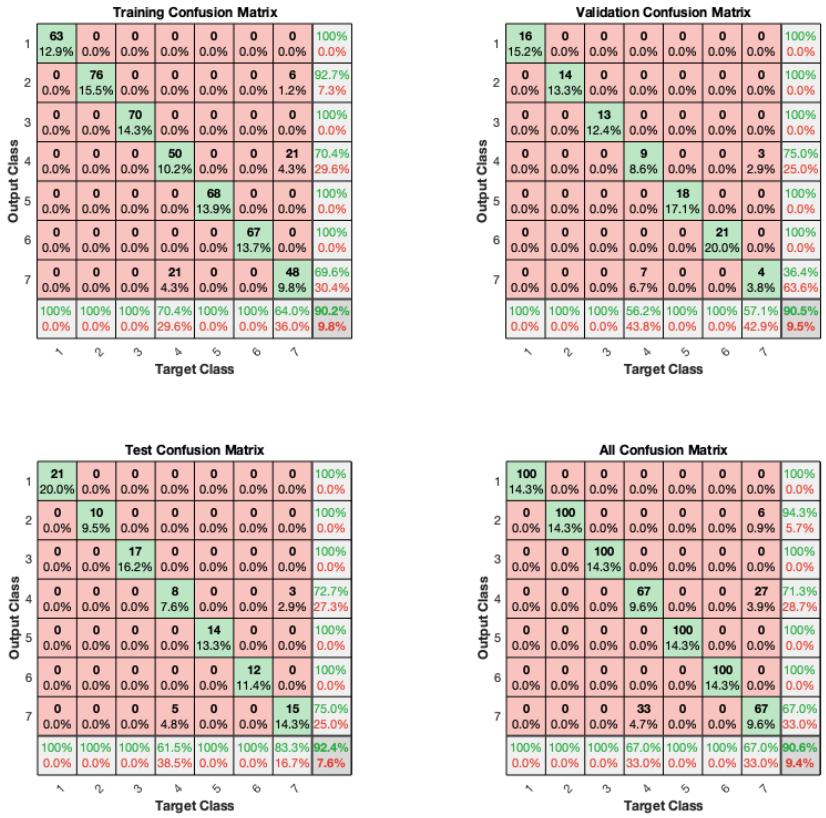

Figure 42: Confusion matrices of the train, validate, and test partitions of Statistical feature #19 on the CWRU subset A

D Appendix - Confusion matrices of robustness testing

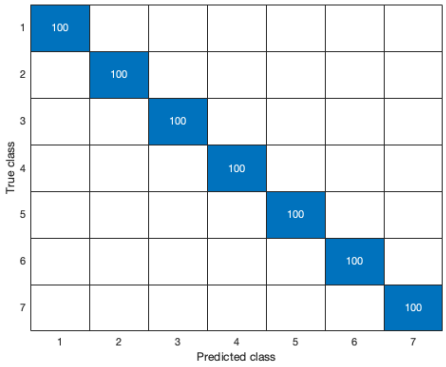

Figure 43: Confusion matrix of the robustness test using the ±25%-noise-added data set

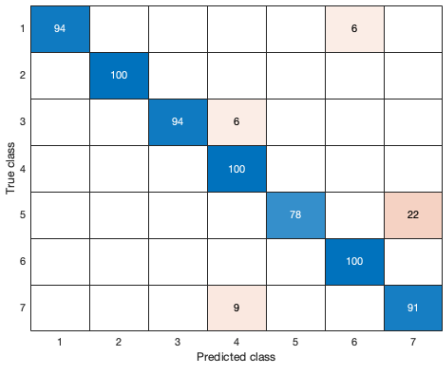

Figure 44: Confusion matrix of the robustness test using the ±50%-noise-added data set

E Appendix - Functions of traditional statistical analysis

```
%{
    Feature 1: the shape factor
    @param x: a signal series
    @param N: the number of data points
    @return F1: feature 1, the shape factor
%}
function F1 = f1_shape_factor(x, N)
    F1 = sqrt(sum(x .^ 2) / N) / (sum(abs(x)) / N);
end
```

```
%{
    Feature 2: the crest factor
    @param x: a signal series
    @param N: the number of data points
    @return F2: feature 2, the crest factor
%}
function F2 = f2_crest_factor(x, N)
    F2 = max(abs(x)) / sqrt(sum(x .^ 2) / N);
end
```

```
%{
    Feature 3: the impulse factor
    @param x: a signal series
    @param N: the number of data points
    @return F3: feature 3, the impulse
%}
function F3 = f3_impulse_factor(x, N)
    F3 = max(abs(x)) / (sum(abs(x)) / N);
end
```

```
%{
    Feature 4: the margin factor
    @param x: a signal series
    @param N: the number of data points
    @return F4: feature 4, the margin factor
%}
function F4 = f4_margin_factor(x, N)
    F4 = max(abs(x)) / (sum(sqrt(abs(x))) / N) .^ 2;
end
```

```
%{
    Feature 5: the kurtosis factor
    @param x: a signal series
    @param N: the number of data points
    @return F5: feature 5, the kurtosis factor
%}
function F5 = f5_kurtosis_factor(x, N)
    x_avg = mean(x);
    F5 = (sum((x - x_avg) .^ 4) / N) / (sqrt(sum((x - x_avg) .^ 2) / N) .^ 4);
end
```

```
%{
    Feature 6: the skewness factor
    @param x: a signal series
    @param N: the number of data points
    @return F6: feature 6, the skewness factor
%}
```

```
function F6 = f6_skewness_factor(x, N)
    x_avg = mean(x);
    F6 = (sum((x − x_avg) .^ 3) / N) / (sqrt(sum((x − x_avg) .^ 2) / N) .^ 3);
end
```

```
%{
    Combine the time−domain features
    @param x: a signal series
    @param N: the number of data points
    @return F_time: the time−domain feature vector
%}
function F_time = combine_time_features(x, N)
    % combine the shape factor
    F_time(1, :) = f1_shape_factor(x, N);
    % combine the crest factor
    F_time(2, :) = f2_crest_factor(x, N);
    % combine the impulse factor
    F_time(3, :) = f3_impulse_factor(x, N);
    % combine the margin factor
    F_time(4, :) = f4_margin_factor(x, N);
    % combine the kurtosis factor
    F_time(5, :) = f5_kurtosis_factor(x, N);
    % combine the skewness factor
    F_time(6, :) = f6_skewness_factor(x, N);
end
```

```
%{
    Compute the spectrum
    @param x: a signal series
    @param N: the number of data points
    @param sf: the sampling frequency
    @param z: the proportation of standard deviation from mean of spectrum
    @return f: the frequency
    @return s: the spectrum
%}
function [f, s] = compute_spectrum(x, N, sf, z)
    % compute the frequency domain
    fd = (sf * (0:(N/2)) / N)';
    % compute the Fourier transform of the signal
    Y = fft(x);
    % compute the two−sided spectrum P2
    P2 = abs(Y / N);
    % compute the single−sided spectrum P1
    P1 = P2(1:(N / 2 + 1));
    P1(2:end−1) = 2 * P1(2:end−1);
    % filter spikes
    s_avg = mean(P1);
    s_std = std(P1);
    filtered = P1 > (s_avg + z * s_std);
```

```
    f = fd(filtered);
    s = P1(filtered);
end
```

```
%{
    Feature 7: the vibration energy in the frequency domain
    @param s: a spectrum
    @param K: the number of spectrum lines
    @return F7: feature 7, the vibration energy in the frequency domain
%}
function F7 = f7(s, K)
    F7 = sum(s) / K;
end
```

```
%{
    Feature 8
    @param s: a spectrum
    @param K: the number of spectrum lines
    @param F7: feature 7, the vibration energy in the frequency domain
    @return F8: feature 8
%}
function F8 = f8(s, K, F7)
    F8 = sum((s - F7) .^ 2) / (K - 1);
end
```

```
%{
    Feature 9
    @param s: a spectrum
    @param K: the number of spectrum lines
    @param F7: feature 7, the vibration energy in the frequency domain
    @param F8: feature 8
    @return F9: feature 9
%}
function F9 = f9(s, K, F7, F8)
    F9 = sum((s - F7) .^ 3) / (K * (sqrt(F8) .^ 3));
end
```

```
%{
    Feature 10
    @param s: a spectrum
    @param K: the number of spectrum lines
    @param F7: feature 7, the vibration energy in the frequency domain
    @param F8: feature 8
    @return F10: feature 10
%}
function F10 = f10(s, K, F7, F8)
    F10 = sum((s - F7) .^ 4) / (K * (F8 .^ 2));
end
```

```
%{
    Feature 11
    @param s: the spectrums
    @param K: the number of spectrum lines
    @param f: the frequency values of the spectrum lines
    @return F11: feature 11
%}
function F11 = f11(s, K, f)
    F11 = sum(f .* s) / sum(s);
end
```

```
%{
    Feature 12
    @param s: the spectrums
    @param K: the number of spectrum lines
    @param f: the frequency values of the spectrum lines
    @param F11: feature 11
    @return F12: feature 12
%}
function F12 = f12(s, K, f, F11)
    F12 = sqrt(sum(((f - F11) .^ 2) .* s) / K);
end
```

```
%{
    Feature 13
    @param s: the spectrums
    @param K: the number of spectrum lines
    @param f: the frequency values of the spectrum lines
    @return F13: feature 13
%}
function F13 = f13(s, K, f)
    F13 = sqrt(sum((f .^ 2) .* s) / sum(s));
end
```

```
%{
    Feature 14
    @param s: the spectrums
    @param K: the number of spectrum lines
    @param f: the frequency values of the spectrum lines
    @return F14: feature 14
%}
function F14 = f14(s, K, f)
    F14 = sqrt(sum((f .^ 4) .* s) / sum((f .^ 2) .* s));
end
```

```
%{
    Feature 15
```

```
    @param s: the spectrums
    @param K: the number of spectrum lines
    @param f: the frequency values of the spectrum lines
    @return F15: feature 15
%}
function F15 = f15(s, K, f)
    F15 = sum((f .^ 2) .* s) / sqrt(sum(s) * sum((f .^ 4) .* s));
end
```

```
%{
    Feature 16
    @param F11: feature 11
    @param F12: feature 12
    @return F16: feature 16
%}
function F16 = f16(F11, F12)
    F16 = F12 / F11;
end
```

```
%{
    Feature 17
    @param s: the spectrums
    @param K: the number of spectrum lines
    @param f: the frequency values of the spectrum lines
    @param F11: feature 11
    @param F12: feature 12
    @return F17: feature 17
%}
function F17 = f17(s, K, f, F11, F12)
    F17 = sum(((f - F11) .^ 3) .* s) / (K * F12 .^ 3);
end
```

```
%{
    Feature 18
    @param s: the spectrums
    @param K: the number of spectrum lines
    @param f: the frequency values of the spectrum lines
    @param F11: feature 11
    @param F12: feature 12
    @return F18: feature 18
%}
function F18 = f18(s, K, f, F11, F12)
    F18 = sum(((f - F11) .^ 4) .* s) / (K * F12 .^ 4);
end
```

```
%{
    Feature 19
    @param s: the spectrums
```

```
    @param K: the number of spectrum lines
    @param f: the frequency values of the spectrum lines
    @param F11: feature 11
    @param F12: feature 12
    @return F19: feature 19
%}
function F19 = f19(s, K, f, F11, F12)
    F19 = sum(((f - F11) .^ 0.5) .* s) / (K * sqrt(F12));
end
```

```
%{
    Combine the frequency-domain features
    @param s: the spectrums
    @param K: the number of spectrum lines
    @param f: the frequency values of the spectrum lines
    @return F_freq: the frequency-domain feature vector
%}
function F_freq = combine_freq_features(s, K, f)
    F_freq(1, :) = f7(s, K);
    F_freq(2, :) = f8(s, K, F_freq(1, :));
    F_freq(3, :) = f9(s, K, F_freq(1, :), F_freq(2, :));
    F_freq(4, :) = f10(s, K, F_freq(1, :), F_freq(2, :));
    F_freq(5, :) = f11(s, K, f);
    F_freq(6, :) = f12(s, K, f, F_freq(5, :));
    F_freq(7, :) = f13(s, K, f);
    F_freq(8, :) = f14(s, K, f);
    F_freq(9, :) = f15(s, K, f);
    F_freq(10, :) = f16(F_freq(5, :), F_freq(6, :));
    F_freq(11, :) = f17(s, K, f, F_freq(5, :), F_freq(6, :));
    F_freq(12, :) = f18(s, K, f, F_freq(5, :), F_freq(6, :));
    F_freq(13, :) = f19(s, K, f, F_freq(5, :), F_freq(6, :));
    F_freq = real(F_freq);
end
```

```
%{
    Compute 19 statistical features for MEISVM
    @param x: a signal series
    @param N: the number of data points
    @param sf: the sampling frequency
    @return F: the feature vector
%}
function F = combine_19_features(x, N, sf)
    % compute the spectrum
    [f, s] = compute_spectrum(x, N, sf);
    K = size(s, 1);
    % combine the time-domain features
    F(1:6, :) = combine_time_features(x, N);
    % combine the frequency-domain features
    F(7:19, :) = combine_freq_freatures(s, K, f);
```

```
end
```

F Appendix - Functions of EEMD feature extraction

```
%{
    Compute the shape factor
    @param x: a signal series
    @return f: the shape factor
%}
function f = compute_shape_factor(x)
    N = size(x, 1);
    f = sqrt(sum(x .^ 2) / N) / (sum(abs(x)) / N);
end
```

```
%{
    Compute the impulse factor
    @param x: a signal series
    @return f: the impulse factor
%}
function f = compute_impulse_factor(x)
    N = size(x, 1);
    f = max(abs(x)) / (sum(abs(x)) / N);
end
```

```
%{
    Compute the mean frequency
    @param x: a signal series
    @param sf: the sampling frequency
    @param z: the z-score threshold to filter underlying frequencies
    @return f_avg: the mean frequency
%}
function f_avg = compute_mean_frequency(x, sf, z)
    N = size(x, 1);
    [f, ~] = compute_spectrum(x, N, sf, z);
    f_avg = mean(f);
end
```

```
%{
    Compute the root mean square frequency
    @param x: a signal series
    @param sf: the sampling frequency
    @param z: the z-score threshold to filter underlying frequencies
    @return f_rms: the root mean square frequency
%}
function f_rms = compute_root_mean_square_frequency(x, sf, z)
    N = size(x, 1);
```

```
    [f, ~] = compute_spectrum(x, N, sf, z);
    f_rms = rms(f);
end
```

```
%{
    Compute the standard deviation frequency
    @param x: a signal series
    @param sf: the sampling frequency
    @param z: the z-score threshold to filter underlying frequencies
    @return f_std: the standard deviation frequency
%}
function f_std = compute_standard_deviation_frequency(x, sf, z)
    N = size(x, 1);
    [f, ~] = compute_spectrum(x, N, sf, z);
    f_std = std(f);
end
```

```
%{
    EEMD algorithm: generate ensemble IMFs
    @param x: a noise series
    @param E: the number of ensemble
    @param I: the number of intrinsic mode functions (IMFs)
    @return a: the means of each of the IMFs
    @return f: a column vector of normalised energy of IMFs
%}
function [a, f] = generate_ensemble_IMF(x, E, I)
    % N: the number of data points in each sample
    [N, ~] = size(x);
    % initialise the IMF matrix
    imf = zeros(N, I, E);
    % for each ensemble
    for m=1:E
        % add a different white noise series $n_m with the given amplitude to the investigate
            ↪ signal $x
        n_m = randn(size(x)) / 100;
        x_m = x + n_m;
        % decompose the noise-added signal $x_m into I IMFs
        [imf(:, :, E), ~] = emd(x_m, 'MaxNumIMF', I);
    end
    % calculate the ensemble mean of the M trials for each IMF
    a = zeros(N, I);
    for i=1:I
        a(:, i) = mean(imf(:, i, :), 3);
    end
    % construct a column vector with the energy as element
    e = zeros(I, 1);
    for i=1:I
        e(i, :) = sumsqr(a(:, i));
    end
```

```
    % normalise the feature vector, and make it a column vector
    total = sqrt(sumsqr(e));
    f = e / total;
end
```

```
%{
    Select sensitive IMF after EEMD algoirithm
    @param A: the means of ensemble IMFs, size=(N, I, M)
    @param faulty: the boolean value which indicates whether the bearings
    are faulty
    @return A_sensitive: the sensitive IMF
%}
function [A_sensitive, i_min] = select_sensitive_IMF(A, faulty)
    % N: the number of data points in each sample
    % I: the number of IMFs
    % M: the number of samples
    [N, I, M] = size(A);
    % compute the kurtosis values of IMFs
    k = zeros(I, M);
    for i=1:I
        for m=1:M
            k(i, m) = kurtosis(A(:, i, m));
        end
    end
    % compute the mean of kurtosis values of IMFs
    k_avg = mean(k, 2);
    % compute the standard deviation of kurtosis values of IMFs
    k_std = std(k, 0, 2);
    % compute the criterion
    if faulty
        f = k_std ./ k_avg;
    else
        f = k_avg .* k_std;
    end
    % select the sensitive IMF
    [~, i_min] = min(f);
    A_sensitive = A(:, i_min, :);
end
```

```
%{
    Extract EEMD features from the sensitive IMF
    @param A: the sensitive IMF, size=(N, 1, M)
    @param sf: the sampling frequency
    @param z: the z−score threshold to filter underlying frequencies
    @return F: the feature matrix, size=(7, M)
%}
function F = extract_EEMD_feature(A, sf, z)
    % N: the number of data points in each sample
    % M: the number of samples
```

```
    [N, ~, M] = size(A);
    % initialise the feature matrix
    F = zeros(7, M);
    % extract the feature vector for each sample
    for m=1:M
        % feature 1: standard deviation
        F(1, m) = std(A(:, 1, m));
        % feature 2: kurtosis
        F(2, m) = kurtosis(A(:, 1, m));
        % feature 3: shape factor
        F(3, m) = sqrt(sum(A(:, 1, m) .^ 2) / N) / (sumabs(A(:, 1, m)) / N);
        % feature 4: impulse factor
        F(4, m) = max(abs(A(:, 1, m))) / (sum(abs(A(:, 1, m))) / N);
        % feature 5: mean frequency
        F(5, m) = compute_mean_frequency(A(:, 1, m), sf, z);
        % feature 6: root mean square frequency
        F(6, m) = compute_root_mean_square_frequency(A(:, 1, m), sf, z);
        % feature 7: standard deviation frequency
        F(7, m) = compute_standard_deviation_frequency(A(:, 1, m), sf, z);
    end
end
```

G Appendix - Code to clean and process CWRU subset A

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/normal_baseline/normal_1.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X098_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/a1_normal.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/OR014@6_1.mat');
for m=1:M
```

```
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X198_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/a2_or_fault_014.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/IR007_1.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X106_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/a3_ir_fault_007.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/IR014_1.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X170_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/a4_ir_fault_014.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/IR021_1.mat');
for m=1:M
```

```
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X210_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/a5_ir_fault_021.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/IR028_1.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X057_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/a6_ir_fault_028.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/B014_1.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X186_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/a7_b_fault_014.mat';
save(data_sink_path, 'x');
```

```
a1_normal_cleaning;
a2_or_fault_014_cleaning;
a3_ir_fault_007_cleaning;
a4_ir_fault_014_cleaning;
a5_ir_fault_021_cleaning;
a6_ir_fault_028_cleaning;
a7_b_fault_014_cleaning;
```

```
clear;
N = 1024;
M = 100 * 7;
X = zeros(N, M);

load('data/interim/CWRU/a1_normal.mat');
X(:, 001:100) = x;

load('data/interim/CWRU/a2_or_fault_014.mat');
X(:, 101:200) = x;

load('data/interim/CWRU/a3_ir_fault_007.mat');
X(:, 201:300) = x;

load('data/interim/CWRU/a4_ir_fault_014.mat');
X(:, 301:400) = x;

load('data/interim/CWRU/a5_ir_fault_021.mat');
X(:, 401:500) = x;

load('data/interim/CWRU/a6_ir_fault_028.mat');
X(:, 501:600) = x;

load('data/interim/CWRU/a7_b_fault_014.mat');
X(:, 601:700) = x;

% generate numerical representation of target value
T_number = zeros(1, M);
T_number(:, 001:100) = 1;
T_number(:, 101:200) = 2;
T_number(:, 201:300) = 3;
T_number(:, 301:400) = 4;
T_number(:, 401:500) = 5;
T_number(:, 501:600) = 6;
T_number(:, 601:700) = 7;

% generate one-hot encoding of target value
T_onehot = zeros(7, M);
for m=1:M
    t = T_number(:, m);
    T_onehot(t, m) = 1;
end

data_sink_path = 'data/interim/CWRU/a_combined.mat';
save(data_sink_path, 'X', 'T_number', 'T_onehot');
```

H Appendix - Code to clean and process MFPT subset B

```
clear;
N = 1024;
M = 120;

load('data/external/MFPT/1 - Three Baseline Conditions/baseline_1.mat');
for m=1:40
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = bearing.gs(head:tail, :);
end

load('data/external/MFPT/1 - Three Baseline Conditions/baseline_2.mat');
for m=41:80
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = bearing.gs(head:tail, :);
end

load('data/external/MFPT/1 - Three Baseline Conditions/baseline_3.mat');
for m=81:120
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = bearing.gs(head:tail, :);
end

data_sink_path = 'data/interim/MFPT/b1_normal.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 120;
x = zeros(N, M);

load('data/external/MFPT/3 - Seven More Outer Race Fault Conditions/
    ↪ OuterRaceFault_vload_6.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = bearing.gs(head:tail, :);
end

data_sink_path = 'data/interim/MFPT/b2_or_fault_250lb.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
```

```
M = 120;
x = zeros(N, M);

load('data/external/MFPT/4_-_Seven_Inner_Race_Fault_Conditions/InnerRaceFault_vload_6.
    ↪ mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = bearing.gs(head:tail, :);
end

data_sink_path = 'data/interim/MFPT/b3_ir_fault_250lb.mat';
save(data_sink_path, 'x');
```

```
b1_normal_cleaning;
b2_or_fault_250lb_cleaning;
b3_ir_fault_250lb_cleaning;

clear;
N = 1024;
M = 120 * 3;
X = zeros(N, M);

load('data/interim/MFPT/b1_normal.mat');
X(:, 001:120) = x;
load('data/interim/MFPT/b2_or_fault_250lb.mat');
X(:, 121:240) = x;
load('data/interim/MFPT/b3_ir_fault_250lb.mat');
X(:, 241:360) = x;

% generate numerical representation of target value
T_number = zeros(1, M);
T_number(:, 001:120) = 1;
T_number(:, 121:240) = 2;
T_number(:, 241:360) = 3;

% generate one-hot encoding of target value
T_onehot = zeros(3, M);
for m = 1:M
    t = T_number(:, m);
    T_onehot(t, m) = 1;
end

data_sink_path = 'data/interim/MFPT/b_combined.mat';
save(data_sink_path, 'X', 'T_number', 'T_onehot');
```

I Appendix - Code to clean and process CWRU subset C

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/normal_baseline/normal_2.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X099_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/c1_normal.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/OR014@6_2.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X199_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/c2_or_fault_014.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/IR007_2.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X107_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/c3_ir_fault_007.mat';
```

```
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/IR014_2.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X171_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/c4_ir_fault_014.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/IR021_2.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X211_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/c5_ir_fault_021.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/IR028_2.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X058_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/c6_ir_fault_028.mat';
```

```
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/B014_2.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X187_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/c7_b_fault_014.mat';
save(data_sink_path, 'x');
```

```
c1_normal_cleaning;
c2_or_fault_014_cleaning;
c3_ir_fault_007_cleaning;
c4_ir_fault_014_cleaning;
c5_ir_fault_021_cleaning;
c6_ir_fault_028_cleaning;
c7_b_fault_014_cleaning;

clear;
N = 1024;
M = 100 * 7;
X = zeros(N, M);

load('data/interim/CWRU/c1_normal.mat');
X(:, 001:100) = x;

load('data/interim/CWRU/c2_or_fault_014.mat');
X(:, 101:200) = x;

load('data/interim/CWRU/c3_ir_fault_007.mat');
X(:, 201:300) = x;

load('data/interim/CWRU/c4_ir_fault_014.mat');
X(:, 301:400) = x;

load('data/interim/CWRU/c5_ir_fault_021.mat');
X(:, 401:500) = x;

load('data/interim/CWRU/c6_ir_fault_028.mat');
X(:, 501:600) = x;
```

```
load('data/interim/CWRU/c7_b_fault_014.mat');
X(:, 601:700) = x;

% generate numerical representation of target value
T_number = zeros(1, M);
T_number(:, 001:100) = 1;
T_number(:, 101:200) = 2;
T_number(:, 201:300) = 3;
T_number(:, 301:400) = 4;
T_number(:, 401:500) = 5;
T_number(:, 501:600) = 6;
T_number(:, 601:700) = 7;

% generate one-hot encoding of target value
T_onehot = zeros(7, M);
for m=1:M
    t = T_number(:, m);
    T_onehot(t, m) = 1;
end

data_sink_path = 'data/interim/CWRU/c_combined.mat';
save(data_sink_path, 'X', 'T_number', 'T_onehot');
```

J Appendix - Code to clean and process CWRU subset D

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/normal_baseline/normal_3.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X100_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/d1_normal.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/OR014@6_3.mat');
```

```
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X200_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/d2_or_fault_014.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/IR007_3.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X108_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/d3_ir_fault_007.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/IR014_3.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X172_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/d4_ir_fault_014.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/IR021_3.mat');
```

```
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X212_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/d5_ir_fault_021.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/IR028_3.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X059_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/d6_ir_fault_028.mat';
save(data_sink_path, 'x');
```

```
clear;
N = 1024;
M = 100;
x = zeros(N, M);

load('data/external/CWRU/12k_drive_end_bearing_fault/B014_3.mat');
for m=1:M
    head = N * (m - 1) + 1;
    tail = N * m;
    x(:, m) = X188_DE_time(head:tail, :);
end
clearvars -except x;

data_sink_path = 'data/interim/CWRU/d7_b_fault_014.mat';
save(data_sink_path, 'x');
```

```
d1_normal_cleaning;
d2_or_fault_014_cleaning;
d3_ir_fault_007_cleaning;
d4_ir_fault_014_cleaning;
d5_ir_fault_021_cleaning;
d6_ir_fault_028_cleaning;
```

```
d7_b_fault_014_cleaning;

clear;
N = 1024;
M = 100 * 7;
X = zeros(N, M);

load('data/interim/CWRU/d1_normal.mat');
X(:, 001:100) = x;

load('data/interim/CWRU/d2_or_fault_014.mat');
X(:, 101:200) = x;

load('data/interim/CWRU/d3_ir_fault_007.mat');
X(:, 201:300) = x;

load('data/interim/CWRU/d4_ir_fault_014.mat');
X(:, 301:400) = x;

load('data/interim/CWRU/d5_ir_fault_021.mat');
X(:, 401:500) = x;

load('data/interim/CWRU/d6_ir_fault_028.mat');
X(:, 501:600) = x;

load('data/interim/CWRU/d7_b_fault_014.mat');
X(:, 601:700) = x;

% generate numerical representation of target value
T_number = zeros(1, M);
T_number(:, 001:100) = 1;
T_number(:, 101:200) = 2;
T_number(:, 201:300) = 3;
T_number(:, 301:400) = 4;
T_number(:, 401:500) = 5;
T_number(:, 501:600) = 6;
T_number(:, 601:700) = 7;

% generate one-hot encoding of target value
T_onehot = zeros(7, M);
for m=1:M
    t = T_number(:, m);
    T_onehot(t, m) = 1;
end

data_sink_path = 'data/interim/CWRU/d_combined.mat';
save(data_sink_path, 'X', 'T_number', 'T_onehot');
```

K Appendix - Code to clean and process CWRU subset Z

```
N = 1024;
M = 100;
Z = 0.5;

load('data/interim/CWRU/a_combined.mat');

% set random seed
setdemorandstream(491218382);

% add white noise to class 1
std1 = std(X(:, 001:100), 1, 'all');
X(:, 001:100) = X(:, 001:100) + ((rand(N, M) - 0.5) .* Z .* std1);

% add white noise to class 2
std2 = std(X(:, 101:200), 1, 'all');
X(:, 101:200) = X(:, 101:200) + ((rand(N, M) - 0.5) .* Z .* std2);

% add white noise to class 3
std3 = std(X(:, 201:300), 1, 'all');
X(:, 201:300) = X(:, 201:300) + ((rand(N, M) - 0.5) .* Z .* std3);

% add white noise to class 4
std4 = std(X(:, 301:400), 1, 'all');
X(:, 301:400) = X(:, 301:400) + ((rand(N, M) - 0.5) .* Z .* std4);

% add white noise to class 5
std5 = std(X(:, 401:500), 1, 'all');
X(:, 401:500) = X(:, 401:500) + ((rand(N, M) - 0.5) .* Z .* std5);

% add white noise to class 6
std6 = std(X(:, 501:600), 1, 'all');
X(:, 501:600) = X(:, 501:600) + ((rand(N, M) - 0.5) .* Z .* std6);

% add white noise to class 7
std7 = std(X(:, 601:700), 1, 'all');
X(:, 601:700) = X(:, 601:700) + ((rand(N, M) - 0.5) .* Z .* std7);

data_sink_path = 'data/interim/CWRU/z_combined.mat';
save(data_sink_path, 'X', 'T_number', 'T_onehot');
```

L Appendix - Code to visualise all statistical and EEMD-derived features extracted from CWRU subset A

```
clear;
```

```
F = 6;

% combine time-domain features for normal
load('data/interim/CWRU/a1_normal.mat');
[N, M] = size(x);
F_time_a1 = zeros(F, M);
for m=1:M
    F_time_a1(:, m) = combine_time_features(x(:, m), N);
end

% combine time-domain features for 014 outer race fault
load('data/interim/CWRU/a2_or_fault_014.mat');
[N, M] = size(x);
F_time_a2 = zeros(F, M);
for m=1:M
    F_time_a2(:, m) = combine_time_features(x(:, m), N);
end

% combine time-domain features for 007 inner race fault
load('data/interim/CWRU/a3_ir_fault_007.mat');
[N, M] = size(x);
F_time_a3 = zeros(F, M);
for m=1:M
    F_time_a3(:, m) = combine_time_features(x(:, m), N);
end

% combine time-domain features for 014 inner race fault
load('data/interim/CWRU/a4_ir_fault_014.mat');
[N, M] = size(x);
F_time_a4 = zeros(F, M);
for m=1:M
    F_time_a4(:, m) = combine_time_features(x(:, m), N);
end

% combine time-domain features for 021 inner race fault
load('data/interim/CWRU/a5_ir_fault_021.mat');
[N, M] = size(x);
F_time_a5 = zeros(F, M);
for m=1:M
    F_time_a5(:, m) = combine_time_features(x(:, m), N);
end

% combine time-domain features for 028 inner race fault
load('data/interim/CWRU/a6_ir_fault_028.mat');
[N, M] = size(x);
F_time_a6 = zeros(F, M);
for m=1:M
    F_time_a6(:, m) = combine_time_features(x(:, m), N);
end
```

```
% combine time-domain features for 014 ball fault
load('data/interim/CWRU/a7_b_fault_014.mat');
[N, M] = size(x);
F_time_a7 = zeros(F, M);
for m=1:M
    F_time_a7(:, m) = combine_time_features(x(:, m), N);
end

for f=1:F
    fig = figure('Position', [0 0 400 300]);hold on
    plot(F_time_a1(f, :), 'Marker' , 'o');
    plot(F_time_a2(f, :), 'Marker' , '+');
    plot(F_time_a3(f, :), 'Marker' , '*');
    plot(F_time_a4(f, :), 'Marker' , 's');
    plot(F_time_a5(f, :), 'Marker' , '^');
    plot(F_time_a6(f, :), 'Marker' , 'p');
    plot(F_time_a7(f, :), 'Marker' , 'h');
    title(strcat('CWRU Data: Time-Domain Feature #', num2str(f)));
    xlabel('Sample point');
    ylabel(strcat('F', num2str(f)));
    legend('Normal', 'OR 014', 'IR 007', 'IR 014', 'IR 021', 'IR 028', 'B 014', 'Location', '
        ↪ northeastoutside');
    saveas(fig, strcat('reports/figures/CWRU_stats_F', num2str(f), '_time.png'));
end
```

```
clear;
F = 13;
sf = 12000;
z = 2;

% combine freq-domain features for normal
load('data/interim/CWRU/a1_normal.mat');
[N, M] = size(x);
F_freq_a1 = zeros(F, M);
for m=1:M
    [f, s] = compute_spectrum(x(:, m), N, sf, z);
    [K, ~] = size(s);
    F_freq_a1(:, m) = combine_freq_features(s, K, f);
end

% combine freq-domain features for 014 outer race fault
load('data/interim/CWRU/a2_or_fault_014.mat');
[N, M] = size(x);
F_freq_a2 = zeros(F, M);
for m=1:M
    [f, s] = compute_spectrum(x(:, m), N, sf, z);
    [K, ~] = size(s);
    F_freq_a2(:, m) = combine_freq_features(s, K, f);
```

```
end

% combine freq-domain features for 007 inner race fault
load('data/interim/CWRU/a3_ir_fault_007.mat');
[N, M] = size(x);
F_freq_a3 = zeros(F, M);
for m=1:M
    [f, s] = compute_spectrum(x(:, m), N, sf, z);
    [K, ~] = size(s);
    F_freq_a3(:, m) = combine_freq_features(s, K, f);
end

% combine freq-domain features for 014 inner race fault
load('data/interim/CWRU/a4_ir_fault_014.mat');
[N, M] = size(x);
F_freq_a4 = zeros(F, M);
for m=1:M
    [f, s] = compute_spectrum(x(:, m), N, sf, z);
    [K, ~] = size(s);
    F_freq_a4(:, m) = combine_freq_features(s, K, f);
end

% combine freq-domain features for 021 inner race fault
load('data/interim/CWRU/a5_ir_fault_021.mat');
[N, M] = size(x);
F_freq_a5 = zeros(F, M);
for m=1:M
    [f, s] = compute_spectrum(x(:, m), N, sf, z);
    [K, ~] = size(s);
    F_freq_a5(:, m) = combine_freq_features(s, K, f);
end

% combine freq-domain features for 028 inner race fault
load('data/interim/CWRU/a6_ir_fault_028.mat');
[N, M] = size(x);
F_freq_a6 = zeros(F, M);
for m=1:M
    [f, s] = compute_spectrum(x(:, m), N, sf, z);
    [K, ~] = size(s);
    F_freq_a6(:, m) = combine_freq_features(s, K, f);
end

% combine freq-domain features for 014 ball fault
load('data/interim/CWRU/a7_b_fault_014.mat');
[N, M] = size(x);
F_freq_a7 = zeros(F, M);
for m=1:M
    [f, s] = compute_spectrum(x(:, m), N, sf, z);
    [K, ~] = size(s);
```

```
    F_freq_a7(:, m) = combine_freq_features(s, K, f);
end

for f=1:F
    fig = figure('Position', [0 0 400 300]);hold on
    plot(F_freq_a1(f, :), 'Marker' , 'o');
    plot(F_freq_a2(f, :), 'Marker' , '+');
    plot(F_freq_a3(f, :), 'Marker' , '*');
    plot(F_freq_a4(f, :), 'Marker' , 's');
    plot(F_freq_a5(f, :), 'Marker' , '^');
    plot(F_freq_a6(f, :), 'Marker' , 'p');
    plot(F_freq_a7(f, :), 'Marker' , 'h');
    title(strcat('CWRU Data: Freq-Domain Feature #', num2str(6 + f), ', z=', num2str(
        ↪ z)));
    xlabel('Sample point');
    ylabel(strcat('F', num2str(6 + f)));
    legend('Normal', 'OR 014', 'IR 007', 'IR 014', 'IR 021', 'IR 028', 'B 014', 'Location', '
        ↪ northeastoutside');
    saveas(fig, strcat('reports/figures/CWRU_stats_F', num2str(6 + f), '_freq_z', num2str(z),
        ↪  '.png'));
end
```

```
clear;
m_EEMD = 7;
M_ens = 10;
I_imf = 4;
sf = 12000;
z = 2;

% combine EEMD features for normal
load('data/interim/CWRU/a1_normal.mat');
[N, M] = size(x);
A_a1 = zeros(N, I_imf, M);
for m=1:M
    [A_a1(:, :, m), ~] = generate_ensemble_IMF(x(:, m), M_ens, I_imf);
end
A_sen_a1 = A_a1(:, 1, :);
F_EEMD_a1 = extract_EEMD_feature(A_sen_a1, sf, z);

% combine EEMD features for 014 outer race fault
load('data/interim/CWRU/a2_or_fault_014.mat');
[N, M] = size(x);
A_a2 = zeros(N, I_imf, M);
for m=1:M
    [A_a2(:, :, m), ~] = generate_ensemble_IMF(x(:, m), M_ens, I_imf);
end
A_sen_a2 = A_a2(:, 1, :);
F_EEMD_a2 = extract_EEMD_feature(A_sen_a2, sf, z);
```

```
% combine EEMD features for 007 inner race fault
load('data/interim/CWRU/a3_ir_fault_007.mat');
[N, M] = size(x);
A_a3 = zeros(N, I_imf, M);
for m=1:M
    [A_a3(:, :, m), ~] = generate_ensemble_IMF(x(:, m), M_ens, I_imf);
end
A_sen_a3 = A_a3(:, 1, :);
F_EEMD_a3 = extract_EEMD_feature(A_sen_a3, sf, z);

% combine EEMD features for 014 inner race fault
load('data/interim/CWRU/a4_ir_fault_014.mat');
[N, M] = size(x);
A_a4 = zeros(N, I_imf, M);
for m=1:M
    [A_a4(:, :, m), ~] = generate_ensemble_IMF(x(:, m), M_ens, I_imf);
end
A_sen_a4 = A_a4(:, 1, :);
F_EEMD_a4 = extract_EEMD_feature(A_sen_a4, sf, z);

% combine EEMD features for 021 inner race fault
load('data/interim/CWRU/a5_ir_fault_021.mat');
[N, M] = size(x);
A_a5 = zeros(N, I_imf, M);
for m=1:M
    [A_a5(:, :, m), ~] = generate_ensemble_IMF(x(:, m), M_ens, I_imf);
end
A_sen_a5 = A_a5(:, 1, :);
F_EEMD_a5 = extract_EEMD_feature(A_sen_a5, sf, z);

% combine EEMD features for 028 inner race fault
load('data/interim/CWRU/a6_ir_fault_028.mat');
[N, M] = size(x);
A_a6 = zeros(N, I_imf, M);
for m=1:M
    [A_a6(:, :, m), ~] = generate_ensemble_IMF(x(:, m), M_ens, I_imf);
end
A_sen_a6 = A_a6(:, 1, :);
F_EEMD_a6 = extract_EEMD_feature(A_sen_a6, sf, z);

% combine EEMD features for 014 ball race fault
load('data/interim/CWRU/a7_b_fault_014.mat');
[N, M] = size(x);
A_a7 = zeros(N, I_imf, M);
for m=1:M
    [A_a7(:, :, m), ~] = generate_ensemble_IMF(x(:, m), M_ens, I_imf);
end
A_sen_a7 = A_a7(:, 1, :);
F_EEMD_a7 = extract_EEMD_feature(A_sen_a7, sf, z);
```

```
for f=1:m_EEMD
    fig = figure('Position', [0 0 400 300]);hold on
    plot(F_EEMD_a1(f, :), 'Marker' , 'o');
    plot(F_EEMD_a2(f, :), 'Marker' , '+');
    plot(F_EEMD_a3(f, :), 'Marker' , '*');
    plot(F_EEMD_a4(f, :), 'Marker' , 's');
    plot(F_EEMD_a5(f, :), 'Marker' , '^');
    plot(F_EEMD_a6(f, :), 'Marker' , 'p');
    plot(F_EEMD_a7(f, :), 'Marker' , 'h');
    title(strcat('CWRU Data: EEMD Feature #', num2str(f), ', z=', num2str(z)));
    xlabel('Sample point');
    ylabel(strcat('F', num2str(f)));
    legend('Normal', 'OR 014', 'IR 007', 'IR 014', 'IR 021', 'IR 028', 'B 014', 'Location', '
        ↪ northeastoutside');
    saveas(fig, strcat('reports/figures/CWRU_EEMD_F', num2str(f), '_z', num2str(z), '.png')
        ↪ );
end
```

M Appendix - Code to visualise visually discriminative features extracted from CWRU subset A using different z-score threshold

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z, M);

load('data/processed/CWRU/a_EEMD_F1_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/a_EEMD_F1_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/a_EEMD_F1_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/a_EEMD_F1_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/a_EEMD_F1_z2.mat');
F_stacked(5, :) = F;

F_avg = mean(F_stacked, 1);
F_med = median(F_stacked, 1);
z_score = '0005101520';

for z=1:Z
    fig = figure('Position', [0 0 400 300]);hold on
    plot(F_stacked(z, 001:100), 'Marker' , 'o');
```

```
    plot(F_stacked(z, 101:200), 'Marker' , '+');
    plot(F_stacked(z, 201:300), 'Marker' , '*');
    plot(F_stacked(z, 301:400), 'Marker' , 's');
    plot(F_stacked(z, 401:500), 'Marker' , '^');
    plot(F_stacked(z, 501:600), 'Marker' , 'p');
    plot(F_stacked(z, 601:700), 'Marker' , 'h');
    title(strcat('CWRU subset A: EEMD F1, z=', num2str(0.5 * (z - 1))));
    xlabel('Sample point');
    ylabel('F1');
    legend('Class 1', 'Class 2', 'Class 3', 'Class 4', 'Class 5', 'Class 6', 'Class 7', 'Location', '
        ↪ northeastoutside');
    saveas(fig, strcat('reports/figures/CWRU_features/CWRU_EEMD_F1_z', z_score(z*2-1:z
        ↪ *2), '.png'));
end

fig = figure('Position', [0 0 400 300]);hold on
plot(F_avg(:, 001:100), 'Marker' , 'o');
plot(F_avg(:, 101:200), 'Marker' , '+');
plot(F_avg(:, 201:300), 'Marker' , '*');
plot(F_avg(:, 301:400), 'Marker' , 's');
plot(F_avg(:, 401:500), 'Marker' , '^');
plot(F_avg(:, 501:600), 'Marker' , 'p');
plot(F_avg(:, 601:700), 'Marker' , 'h');
title('CWRU Data: EEMD Feature #1, average');
xlabel('Sample point');
ylabel('F1');
legend('Class 1', 'Class 2', 'Class 3', 'Class 4', 'Class 5', 'Class 6', 'Class 7', 'Location', '
    ↪ northeastoutside');

fig = figure('Position', [0 0 400 300]);hold on
plot(F_med(:, 001:100), 'Marker' , 'o');
plot(F_med(:, 101:200), 'Marker' , '+');
plot(F_med(:, 201:300), 'Marker' , '*');
plot(F_med(:, 301:400), 'Marker' , 's');
plot(F_med(:, 401:500), 'Marker' , '^');
plot(F_med(:, 501:600), 'Marker' , 'p');
plot(F_med(:, 601:700), 'Marker' , 'h');
title('CWRU Data: EEMD Feature #1, median');
xlabel('Sample point');
ylabel('F1');
legend('Class 1', 'Class 2', 'Class 3', 'Class 4', 'Class 5', 'Class 6', 'Class 7', 'Location', '
    ↪ northeastoutside');
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z, M);

load('data/processed/CWRU/a_EEMD_F5_z0.mat');
```

```
F_stacked(1, :) = F;
load('data/processed/CWRU/a_EEMD_F5_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/a_EEMD_F5_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/a_EEMD_F5_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/a_EEMD_F5_z2.mat');
F_stacked(5, :) = F;

F_avg = mean(F_stacked, 1);
F_med = median(F_stacked, 1);
z_score = '0005101520';

for z=1:Z
    fig = figure('Position', [0 0 400 300]);hold on
    plot(F_stacked(z, 001:100), 'Marker' , 'o');
    plot(F_stacked(z, 101:200), 'Marker' , '+');
    plot(F_stacked(z, 201:300), 'Marker' , '*');
    plot(F_stacked(z, 301:400), 'Marker' , 's');
    plot(F_stacked(z, 401:500), 'Marker' , '^');
    plot(F_stacked(z, 501:600), 'Marker' , 'p');
    plot(F_stacked(z, 601:700), 'Marker' , 'h');
    title(strcat('CWRU_subset_A:_EEMD_F5,_z=', num2str(0.5 * (z - 1))));
    xlabel('Sample_point');
    ylabel('F5');
    legend('Class_1', 'Class_2', 'Class_3', 'Class_4', 'Class_5', 'Class_6', 'Class_7', 'Location', '
        ↪ northeastoutside');
    saveas(fig, strcat('reports/figures/CWRU_features/CWRU_EEMD_F5_z', z_score(z*2-1:z
        ↪ *2), '.png'));
end

fig = figure('Position', [0 0 400 300]);hold on
plot(F_avg(:, 001:100), 'Marker' , 'o');
plot(F_avg(:, 101:200), 'Marker' , '+');
plot(F_avg(:, 201:300), 'Marker' , '*');
plot(F_avg(:, 301:400), 'Marker' , 's');
plot(F_avg(:, 401:500), 'Marker' , '^');
plot(F_avg(:, 501:600), 'Marker' , 'p');
plot(F_avg(:, 601:700), 'Marker' , 'h');
title('CWRU_Data:_EEMD_Feature_#5,_average');
xlabel('Sample_point');
ylabel('F5');
legend('Normal', 'OR_014', 'IR_007', 'IR_014', 'IR_021', 'IR_028', 'B_014', 'Location', '
    ↪ northeastoutside');

fig = figure('Position', [0 0 400 300]);hold on
plot(F_med(:, 001:100), 'Marker' , 'o');
plot(F_med(:, 101:200), 'Marker' , '+');
```

```
plot(F_med(:, 201:300), 'Marker' , '*');
plot(F_med(:, 301:400), 'Marker' , 's');
plot(F_med(:, 401:500), 'Marker' , '^');
plot(F_med(:, 501:600), 'Marker' , 'p');
plot(F_med(:, 601:700), 'Marker' , 'h');
title('CWRU Data: EEMD Feature #5, median');
xlabel('Sample point');
ylabel('F1');
legend('Normal', 'OR 014', 'IR 007', 'IR 014', 'IR 021', 'IR 028', 'B 014', 'Location', '
    ↪ northeastoutside');
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z, M);

load('data/processed/CWRU/a_EEMD_F6_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/a_EEMD_F6_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/a_EEMD_F6_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/a_EEMD_F6_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/a_EEMD_F6_z2.mat');
F_stacked(5, :) = F;

F_avg = mean(F_stacked, 1);
F_med = median(F_stacked, 1);
z_score = '0005101520';

for z=1:Z
    fig = figure('Position', [0 0 400 300]);hold on
    plot(F_stacked(z, 001:100), 'Marker' , 'o');
    plot(F_stacked(z, 101:200), 'Marker' , '+');
    plot(F_stacked(z, 201:300), 'Marker' , '*');
    plot(F_stacked(z, 301:400), 'Marker' , 's');
    plot(F_stacked(z, 401:500), 'Marker' , '^');
    plot(F_stacked(z, 501:600), 'Marker' , 'p');
    plot(F_stacked(z, 601:700), 'Marker' , 'h');
    title(strcat('CWRU subset A: EEMD F6, z=', num2str(0.5 * (z - 1))));
    xlabel('Sample point');
    ylabel('F6');
    legend('Class 1', 'Class 2', 'Class 3', 'Class 4', 'Class 5', 'Class 6', 'Class 7', 'Location', '
        ↪ northeastoutside');
    saveas(fig, strcat('reports/figures/CWRU_features/CWRU_EEMD_F6_z', z_score(z*2−1:z
        ↪ *2), '.png'));
end
```

```
fig = figure('Position', [0 0 400 300]);hold on
plot(F_avg(:, 001:100), 'Marker' , 'o');
plot(F_avg(:, 101:200), 'Marker' , '+');
plot(F_avg(:, 201:300), 'Marker' , '*');
plot(F_avg(:, 301:400), 'Marker' , 's');
plot(F_avg(:, 401:500), 'Marker' , '^');
plot(F_avg(:, 501:600), 'Marker' , 'p');
plot(F_avg(:, 601:700), 'Marker' , 'h');
title('CWRU_Data:_EEMD_Feature_#6,_average');
xlabel('Sample_point');
ylabel('F6');
legend('Normal', 'OR_014', 'IR_007', 'IR_014', 'IR_021', 'IR_028', 'B_014', 'Location', '
    ↪ northeastoutside');

fig = figure('Position', [0 0 400 300]);hold on
plot(F_med(:, 001:100), 'Marker' , 'o');
plot(F_med(:, 101:200), 'Marker' , '+');
plot(F_med(:, 201:300), 'Marker' , '*');
plot(F_med(:, 301:400), 'Marker' , 's');
plot(F_med(:, 401:500), 'Marker' , '^');
plot(F_med(:, 501:600), 'Marker' , 'p');
plot(F_med(:, 601:700), 'Marker' , 'h');
title('CWRU_Data:_EEMD_Feature_#6,_median');
xlabel('Sample_point');
ylabel('F1');
legend('Normal', 'OR_014', 'IR_007', 'IR_014', 'IR_021', 'IR_028', 'B_014', 'Location', '
    ↪ northeastoutside');
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z, M);

load('data/processed/CWRU/a_stats_F14_freq_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/a_stats_F14_freq_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/a_stats_F14_freq_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/a_stats_F14_freq_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/a_stats_F14_freq_z2.mat');
F_stacked(5, :) = F;

F_avg = mean(F_stacked, 1);
F_med = median(F_stacked, 1);
z_score = '0005101520';

for z=1:Z
```

```
    fig = figure('Position', [0 0 400 300]);hold on
    plot(F_stacked(z, 001:100), 'Marker' , 'o');
    plot(F_stacked(z, 101:200), 'Marker' , '+');
    plot(F_stacked(z, 201:300), 'Marker' , '*');
    plot(F_stacked(z, 301:400), 'Marker' , 's');
    plot(F_stacked(z, 401:500), 'Marker' , '^');
    plot(F_stacked(z, 501:600), 'Marker' , 'p');
    plot(F_stacked(z, 601:700), 'Marker' , 'h');
    title(strcat('CWRU subset A: Statistical F14, z=', num2str(0.5 * (z - 1))));
    xlabel('Sample point');
    ylabel('F14');
    legend('Class 1', 'Class 2', 'Class 3', 'Class 4', 'Class 5', 'Class 6', 'Class 7', 'Location', '
        ↪ northeastoutside');
    saveas(fig, strcat('reports/figures/CWRU_features/CWRU_stats_F14_z', z_score(z*2-1:z*2)
        ↪ , '.png'));
end

fig = figure('Position', [0 0 400 300]);hold on
plot(F_avg(:, 001:100), 'Marker' , 'o');
plot(F_avg(:, 101:200), 'Marker' , '+');
plot(F_avg(:, 201:300), 'Marker' , '*');
plot(F_avg(:, 301:400), 'Marker' , 's');
plot(F_avg(:, 401:500), 'Marker' , '^');
plot(F_avg(:, 501:600), 'Marker' , 'p');
plot(F_avg(:, 601:700), 'Marker' , 'h');
title('CWRU Data: Freq-Domain Feature #14, average');
xlabel('Sample point');
ylabel('F14');
legend('Normal', 'OR 014', 'IR 007', 'IR 014', 'IR 021', 'IR 028', 'B 014', 'Location', '
    ↪ northeastoutside');

fig = figure('Position', [0 0 400 300]);hold on
plot(F_med(:, 001:100), 'Marker' , 'o');
plot(F_med(:, 101:200), 'Marker' , '+');
plot(F_med(:, 201:300), 'Marker' , '*');
plot(F_med(:, 301:400), 'Marker' , 's');
plot(F_med(:, 401:500), 'Marker' , '^');
plot(F_med(:, 501:600), 'Marker' , 'p');
plot(F_med(:, 601:700), 'Marker' , 'h');
title('CWRU Data: Freq-Domain Feature #14, median');
xlabel('Sample point');
ylabel('F1');
legend('Normal', 'OR 014', 'IR 007', 'IR 014', 'IR 021', 'IR 028', 'B 014', 'Location', '
    ↪ northeastoutside');
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z, M);
```

```
load('data/processed/CWRU/a_stats_F19_freq_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/a_stats_F19_freq_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/a_stats_F19_freq_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/a_stats_F19_freq_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/a_stats_F19_freq_z2.mat');
F_stacked(5, :) = F;

F_avg = mean(F_stacked, 1);
F_med = median(F_stacked, 1);
z_score = '0005101520';

for z=1:Z
    fig = figure('Position', [0 0 400 300]);hold on
    plot(F_stacked(z, 001:100), 'Marker' , 'o');
    plot(F_stacked(z, 101:200), 'Marker' , '+');
    plot(F_stacked(z, 201:300), 'Marker' , '*');
    plot(F_stacked(z, 301:400), 'Marker' , 's');
    plot(F_stacked(z, 401:500), 'Marker' , '^');
    plot(F_stacked(z, 501:600), 'Marker' , 'p');
    plot(F_stacked(z, 601:700), 'Marker' , 'h');
    title(strcat('CWRU_subset_A:_Statistical_F19,_z=', num2str(0.5 * (z - 1))));
    xlabel('Sample_point');
    ylabel('F19');
    legend('Class_1', 'Class_2', 'Class_3', 'Class_4', 'Class_5', 'Class_6', 'Class_7', 'Location', '
        ↪ northeastoutside');
    saveas(fig, strcat('reports/figures/CWRU_features/CWRU_stats_F19_z', z_score(z*2-1:z*2)
        ↪ , '.png'));
end

fig = figure('Position', [0 0 400 300]);hold on
plot(F_avg(:, 001:100), 'Marker' , 'o');
plot(F_avg(:, 101:200), 'Marker' , '+');
plot(F_avg(:, 201:300), 'Marker' , '*');
plot(F_avg(:, 301:400), 'Marker' , 's');
plot(F_avg(:, 401:500), 'Marker' , '^');
plot(F_avg(:, 501:600), 'Marker' , 'p');
plot(F_avg(:, 601:700), 'Marker' , 'h');
title('CWRU_Data:_Freq-Domain_Feature_#19,_average');
xlabel('Sample_point');
ylabel('F19');
legend('Normal', 'OR_014', 'IR_007', 'IR_014', 'IR_021', 'IR_028', 'B_014', 'Location', '
    ↪ northeastoutside');

fig = figure('Position', [0 0 400 300]);hold on
```

```
plot(F_med(:, 001:100), 'Marker' , 'o');
plot(F_med(:, 101:200), 'Marker' , '+');
plot(F_med(:, 201:300), 'Marker' , '*');
plot(F_med(:, 301:400), 'Marker' , 's');
plot(F_med(:, 401:500), 'Marker' , '^');
plot(F_med(:, 501:600), 'Marker' , 'p');
plot(F_med(:, 601:700), 'Marker' , 'h');
title('CWRU Data: Freq-Domain Feature #19, median');
xlabel('Sample point');
ylabel('F1');
legend('Normal', 'OR_014', 'IR_007', 'IR_014', 'IR_021', 'IR_028', 'B_014', 'Location', '
    ↪ northeastoutside');
```

N Appendix - Code to extract visually discriminative features from CWRU subset A

```
clear;

load('data/interim/CWRU/a_combined.mat');
[N, M] = size(X);
M_ens = 10;
I_imf = 4;
sf = 12000;

% extract EEMD feature #1
for z=[0 0.5 1 1.5 2]
    A = zeros(N, I_imf, M);
    for m=1:M
        [A(:, :, m), ~] = generate_ensemble_IMF(X(:, m), M_ens, I_imf);
    end
    A_sen = A(:, 1, :);
    F_EEMD = extract_EEMD_feature(A_sen, sf, z);
    F = real(F_EEMD(1, :));
    data_sink_path = strcat('data/processed/CWRU/a_EEMD_F1_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
end
```

```
clear;

load('data/interim/CWRU/a_combined.mat');
[N, M] = size(X);
M_ens = 10;
I_imf = 4;
sf = 12000;

% extract EEMD feature #5
```

```
for z=[0 0.5 1 1.5 2]
    A = zeros(N, I_imf, M);
    for m=1:M
        [A(:, :, m), ~] = generate_ensemble_IMF(X(:, m), M_ens, I_imf);
    end
    A_sen = A(:, 1, :);
    F_EEMD = extract_EEMD_feature(A_sen, sf, z);
    F = real(F_EEMD(5, :));
    data_sink_path = strcat('data/processed/CWRU/a_EEMD_F5_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
end
```

```
clear;

load('data/interim/CWRU/a_combined.mat');
[N, M] = size(X);
M_ens = 10;
I_imf = 4;
sf = 12000;

% extract EEMD feature #6
for z=[0 0.5 1 1.5 2]
    A = zeros(N, I_imf, M);
    for m=1:M
        [A(:, :, m), ~] = generate_ensemble_IMF(X(:, m), M_ens, I_imf);
    end
    A_sen = A(:, 1, :);
    F_EEMD = extract_EEMD_feature(A_sen, sf, z);
    F = real(F_EEMD(6, :));
    data_sink_path = strcat('data/processed/CWRU/a_EEMD_F6_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
end
```

```
clear;

load('data/interim/CWRU/a_combined.mat');
[N, M] = size(X);
sf = 12000;

% extract frequency−domain feature #14
for z=[0 0.5 1 1.5 2]
    F = zeros(1, M);
    for m=1:M
        [f, s] = compute_spectrum(X(:, m), N, sf, z);
        [K, ~] = size(s);
        F(:, m) = real(f14(s, K, f));
    end
    data_sink_path = strcat('data/processed/CWRU/a_stats_F14_freq_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
```

```
end
```

```
clear;

load('data/interim/CWRU/a_combined.mat');
[N, M] = size(X);
sf = 12000;

% extract frequency-domain feature #19
for z=[0 0.5 1 1.5 2]
    F = zeros(1, M);
    for m=1:M
        [f, s] = compute_spectrum(X(:, m), N, sf, z);
        [K, ~] = size(s);
        F11 = f11(s, K, f);
        F12 = f12(s, K, f, F11);
        F(:, m) = real(f19(s, K, f, F11, F12));
    end
    data_sink_path = strcat('data/processed/CWRU/a_stats_F19_freq_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
end
```

O Appendix - Code to extract visually discriminative features from MFPT subset B

```
clear;

load('data/interim/MFPT/b_combined.mat');
[N, M] = size(X);
M_ens = 10;
I_imf = 4;
sf = zeros(1, M);
sf(:, 001:120) = 97656;
sf(:, 121:360) = 48828;

% extract EEMD feature #1 #5 #6
for z=[0 0.5 1 1.5 2]
    A = zeros(N, I_imf, M);
    for m=1:M
        [A(:, :, m), ~] = generate_ensemble_IMF(X(:, m), M_ens, I_imf);
    end
    A_sen = A(:, 1, :);
    F_EEMD = zeros(7, M);
    F_EEMD(:, 001:120) = extract_EEMD_feature(A_sen(:, :, 001:120), 97656, z);
    F_EEMD(:, 121:360) = extract_EEMD_feature(A_sen(:, :, 121:360), 48828, z);
    F = real(F_EEMD(1, :));
```

```
    data_sink_path = strcat('data/processed/MFPT/b_EEMD_F1_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
    F = real(F_EEMD(5, :));
    data_sink_path = strcat('data/processed/MFPT/b_EEMD_F5_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
    F = real(F_EEMD(6, :));
    data_sink_path = strcat('data/processed/MFPT/b_EEMD_F6_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
end
```

```
clear;

load('data/interim/MFPT/b_combined.mat');
[N, M] = size(X);
M_ens = 10;
I_imf = 4;
sf = zeros(1, M);
sf(:, 001:120) = 97656;
sf(:, 121:360) = 48828;

% extract frequency−domain feature #14 #19
for z=[0 0.5 1 1.5 2]
    F14 = zeros(1, M);
    F19 = zeros(1, M);
    for m=1:M
        [f, s] = compute_spectrum(X(:, m), N, sf(:, m), z);
        [K, ~] = size(s);
        F11 = f11(s, K, f);
        F12 = f12(s, K, f, F11);
        F14(:, m) = real(f14(s, K, f));
        F19(:, m) = real(f19(s, K, f, F11, F12));
    end
    F = F14;
    data_sink_path = strcat('data/processed/MFPT/b_stats_F14_freq_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
    F = F19;
    data_sink_path = strcat('data/processed/MFPT/b_stats_F19_freq_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
end
```

P Appendix - Code to extract visually discriminative features from CWRU subset C

```
clear;

load('data/interim/CWRU/c_combined.mat');
```

```
[N, M] = size(X);
M_ens = 10;
I_imf = 4;
sf = 12000;

% extract EEMD feature #1 #5 #6
for z=[0 0.5 1 1.5 2]
    A = zeros(N, I_imf, M);
    for m=1:M
        [A(:, :, m), ~] = generate_ensemble_IMF(X(:, m), M_ens, I_imf);
    end
    A_sen = A(:, 1, :);
    F_EEMD = extract_EEMD_feature(A_sen, sf, z);
    F = real(F_EEMD(1, :));
    data_sink_path = strcat('data/processed/CWRU/c_EEMD_F1_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
    F = real(F_EEMD(5, :));
    data_sink_path = strcat('data/processed/CWRU/c_EEMD_F5_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
    F = real(F_EEMD(6, :));
    data_sink_path = strcat('data/processed/CWRU/c_EEMD_F6_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
end
```

```
clear;

load('data/interim/CWRU/c_combined.mat');
[N, M] = size(X);
M_ens = 10;
I_imf = 4;
sf = 12000;

% extract frequency-domain feature #14 #19
for z=[0 0.5 1 1.5 2]
    F14 = zeros(1, M);
    F19 = zeros(1, M);
    for m=1:M
        [f, s] = compute_spectrum(X(:, m), N, sf, z);
        [K, ~] = size(s);
        F11 = f11(s, K, f);
        F12 = f12(s, K, f, F11);
        F14(:, m) = real(f14(s, K, f));
        F19(:, m) = real(f19(s, K, f, F11, F12));
    end
    F = F14;
    data_sink_path = strcat('data/processed/CWRU/c_stats_F14_freq_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
    F = F19;
    data_sink_path = strcat('data/processed/CWRU/c_stats_F19_freq_z', num2str(z), '.mat');
```

```
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
end
```

Q Appendix - Code to extract visually discriminative features from CWRU subset D

```
clear;

load('data/interim/CWRU/d_combined.mat');
[N, M] = size(X);
M_ens = 10;
I_imf = 4;
sf = 12000;

% extract EEMD feature #1 #5 #6
for z=[0 0.5 1 1.5 2]
    A = zeros(N, I_imf, M);
    for m=1:M
        [A(:, :, m), ~] = generate_ensemble_IMF(X(:, m), M_ens, I_imf);
    end
    A_sen = A(:, 1, :);
    F_EEMD = extract_EEMD_feature(A_sen, sf, z);
    F = real(F_EEMD(1, :));
    data_sink_path = strcat('data/processed/CWRU/d_EEMD_F1_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
    F = real(F_EEMD(5, :));
    data_sink_path = strcat('data/processed/CWRU/d_EEMD_F5_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
    F = real(F_EEMD(6, :));
    data_sink_path = strcat('data/processed/CWRU/d_EEMD_F6_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
end
```

```
clear;

load('data/interim/CWRU/d_combined.mat');
[N, M] = size(X);
M_ens = 10;
I_imf = 4;
sf = 12000;

% extract frequency-domain feature #14 #19
for z=[0 0.5 1 1.5 2]
    F14 = zeros(1, M);
    F19 = zeros(1, M);
    for m=1:M
```

```
        [f, s] = compute_spectrum(X(:, m), N, sf, z);
        [K, ~] = size(s);
        F11 = f11(s, K, f);
        F12 = f12(s, K, f, F11);
        F14(:, m) = real(f14(s, K, f));
        F19(:, m) = real(f19(s, K, f, F11, F12));
    end
    F = F14;
    data_sink_path = strcat('data/processed/CWRU/d_stats_F14_freq_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
    F = F19;
    data_sink_path = strcat('data/processed/CWRU/d_stats_F19_freq_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
end
```

R Appendix - Code to extract visually discriminative features from CWRU subset Z

```
clear;

% set random seed
setdemorandstream(491218382);

load('data/interim/CWRU/z_combined.mat');
[N, M] = size(X);
M_ens = 10;
I_imf = 4;
sf = 12000;

% extract EEMD feature #1 #5 #6
for z=[0 0.5 1 1.5 2]
    A = zeros(N, I_imf, M);
    for m=1:M
        [A(:, :, m), ~] = generate_ensemble_IMF(X(:, m), M_ens, I_imf);
    end
    A_sen = A(:, 1, :);
    F_EEMD = extract_EEMD_feature(A_sen, sf, z);
    F = real(F_EEMD(1, :));
    data_sink_path = strcat('data/processed/CWRU/z_EEMD_F1_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
    F = real(F_EEMD(5, :));
    data_sink_path = strcat('data/processed/CWRU/z_EEMD_F5_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
    F = real(F_EEMD(6, :));
    data_sink_path = strcat('data/processed/CWRU/z_EEMD_F6_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
end
```

```
clear;

% set random seed
setdemorandstream(491218382);

load('data/interim/CWRU/z_combined.mat');
[N, M] = size(X);
M_ens = 10;
I_imf = 4;
sf = 12000;

% extract frequency-domain feature #14 #19
for z=[0 0.5 1 1.5 2]
    F14 = zeros(1, M);
    F19 = zeros(1, M);
    for m=1:M
        [f, s] = compute_spectrum(X(:, m), N, sf, z);
        [K, ~] = size(s);
        F11 = f11(s, K, f);
        F12 = f12(s, K, f, F11);
        F14(:, m) = real(f14(s, K, f));
        F19(:, m) = real(f19(s, K, f, F11, F12));
    end
    F = F14;
    data_sink_path = strcat('data/processed/CWRU/z_stats_F14_freq_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
    F = F19;
    data_sink_path = strcat('data/processed/CWRU/z_stats_F19_freq_z', num2str(z), '.mat');
    save(data_sink_path, 'F', 'T_number', 'T_onehot');
end
```

S Appendix - Code to assess baseline performance

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M);

% load the CWRU data set, and stack features
load('data/processed/CWRU/a_EEMD_F1_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/a_EEMD_F1_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/a_EEMD_F1_z1.mat');
F_stacked(3, :) = F;
```

```
load('data/processed/CWRU/a_EEMD_F1_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/a_EEMD_F1_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked, 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M);

% load the CWRU data set, and stack features
load('data/processed/CWRU/a_EEMD_F5_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/a_EEMD_F5_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/a_EEMD_F5_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/a_EEMD_F5_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/a_EEMD_F5_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked, 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M);
```

```
% load the CWRU data set, and stack features
load('data/processed/CWRU/a_EEMD_F6_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/a_EEMD_F6_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/a_EEMD_F6_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/a_EEMD_F6_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/a_EEMD_F6_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked, 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M);

% load the CWRU data set, and stack features
load('data/processed/CWRU/a_stats_F14_freq_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/a_stats_F14_freq_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/a_stats_F14_freq_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/a_stats_F14_freq_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/a_stats_F14_freq_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked, 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
```

```
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M);

% load the CWRU data set, and stack features
load('data/processed/CWRU/a_stats_F19_freq_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/a_stats_F19_freq_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/a_stats_F19_freq_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/a_stats_F19_freq_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/a_stats_F19_freq_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked, 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros((Z + 1) * 3, M);

% load the CWRU data set, and stack EEMD feature #5
load('data/processed/CWRU/a_EEMD_F5_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/a_EEMD_F5_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/a_EEMD_F5_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/a_EEMD_F5_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/a_EEMD_F5_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked(1:5, :), 1);

% load the CWRU data set, and stack EEMD feature #6
```

```
load('data/processed/CWRU/a_EEMD_F6_z0.mat');
F_stacked(7, :) = F;
load('data/processed/CWRU/a_EEMD_F6_z0.5.mat');
F_stacked(8, :) = F;
load('data/processed/CWRU/a_EEMD_F6_z1.mat');
F_stacked(9, :) = F;
load('data/processed/CWRU/a_EEMD_F6_z1.5.mat');
F_stacked(10, :) = F;
load('data/processed/CWRU/a_EEMD_F6_z2.mat');
F_stacked(11, :) = F;
F_stacked(12, :) = mean(F_stacked(7:11, :), 1);

% load the CWRU data set, and stack frequency−domain feature #14
load('data/processed/CWRU/a_stats_F14_freq_z0.mat');
F_stacked(13, :) = F;
load('data/processed/CWRU/a_stats_F14_freq_z0.5.mat');
F_stacked(14, :) = F;
load('data/processed/CWRU/a_stats_F14_freq_z1.mat');
F_stacked(15, :) = F;
load('data/processed/CWRU/a_stats_F14_freq_z1.5.mat');
F_stacked(16, :) = F;
load('data/processed/CWRU/a_stats_F14_freq_z2.mat');
F_stacked(17, :) = F;
F_stacked(18, :) = mean(F_stacked(13:17, :), 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

T Appendix - Code to assess generalisation performance to different data sources

```
clear;
Z = 5;
M = 360;
F_stacked = zeros(Z + 1, M);

% load the MFPT data set, and stack EEMD feature #1
load('data/processed/MFPT/b_EEMD_F1_z0.mat');
F_stacked(1, :) = F;
load('data/processed/MFPT/b_EEMD_F1_z0.5.mat');
```

```
F_stacked(2, :) = F;
load('data/processed/MFPT/b_EEMD_F1_z1.mat');
F_stacked(3, :) = F;
load('data/processed/MFPT/b_EEMD_F1_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/MFPT/b_EEMD_F1_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked(1:5, :), 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 360;
F_stacked = zeros(Z + 1, M);

% load the MFPT data set, and stack EEMD feature #5
load('data/processed/MFPT/b_EEMD_F5_z0.mat');
F_stacked(1, :) = F;
load('data/processed/MFPT/b_EEMD_F5_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/MFPT/b_EEMD_F5_z1.mat');
F_stacked(3, :) = F;
load('data/processed/MFPT/b_EEMD_F5_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/MFPT/b_EEMD_F5_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked(1:5, :), 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
```

```
M = 360;
F_stacked = zeros(Z + 1, M);

% load the MFPT data set, and stack EEMD feature #6
load('data/processed/MFPT/b_EEMD_F6_z0.mat');
F_stacked(1, :) = F;
load('data/processed/MFPT/b_EEMD_F6_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/MFPT/b_EEMD_F6_z1.mat');
F_stacked(3, :) = F;
load('data/processed/MFPT/b_EEMD_F6_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/MFPT/b_EEMD_F6_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked(1:5, :), 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 360;
F_stacked = zeros(Z + 1, M);

% load the MFPT data set, and stack frequency−domain feature #14
load('data/processed/MFPT/b_stats_F14_freq_z0.mat');
F_stacked(1, :) = F;
load('data/processed/MFPT/b_stats_F14_freq_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/MFPT/b_stats_F14_freq_z1.mat');
F_stacked(3, :) = F;
load('data/processed/MFPT/b_stats_F14_freq_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/MFPT/b_stats_F14_freq_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked(1:5, :), 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');
```

```
% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 360;
F_stacked = zeros(Z + 1, M);

% load the MFPT data set, and stack frequency−domain feature #19
load('data/processed/MFPT/b_stats_F19_freq_z0.mat');
F_stacked(1, :) = F;
load('data/processed/MFPT/b_stats_F19_freq_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/MFPT/b_stats_F19_freq_z1.mat');
F_stacked(3, :) = F;
load('data/processed/MFPT/b_stats_F19_freq_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/MFPT/b_stats_F19_freq_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked(1:5, :), 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 360;
F_stacked = zeros((Z + 1) * 3, M);

% load the MFPT data set, and stack EEMD feature #5
load('data/processed/MFPT/b_EEMD_F5_z0.mat');
F_stacked(1, :) = F;
load('data/processed/MFPT/b_EEMD_F5_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/MFPT/b_EEMD_F5_z1.mat');
F_stacked(3, :) = F;
load('data/processed/MFPT/b_EEMD_F5_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/MFPT/b_EEMD_F5_z2.mat');
F_stacked(5, :) = F;
```

```
F_stacked(6, :) = mean(F_stacked(1:5, :), 1);

% load the MFPT data set, and stack EEMD feature #6
load('data/processed/MFPT/b_EEMD_F6_z0.mat');
F_stacked(7, :) = F;
load('data/processed/MFPT/b_EEMD_F6_z0.5.mat');
F_stacked(8, :) = F;
load('data/processed/MFPT/b_EEMD_F6_z1.mat');
F_stacked(9, :) = F;
load('data/processed/MFPT/b_EEMD_F6_z1.5.mat');
F_stacked(10, :) = F;
load('data/processed/MFPT/b_EEMD_F6_z2.mat');
F_stacked(11, :) = F;
F_stacked(12, :) = mean(F_stacked(7:11, :), 1);

% load the MFPT data set, and stack frequency−domain feature #14
load('data/processed/MFPT/b_stats_F14_freq_z0.mat');
F_stacked(13, :) = F;
load('data/processed/MFPT/b_stats_F14_freq_z0.5.mat');
F_stacked(14, :) = F;
load('data/processed/MFPT/b_stats_F14_freq_z1.mat');
F_stacked(15, :) = F;
load('data/processed/MFPT/b_stats_F14_freq_z1.5.mat');
F_stacked(16, :) = F;
load('data/processed/MFPT/b_stats_F14_freq_z2.mat');
F_stacked(17, :) = F;
F_stacked(18, :) = mean(F_stacked(13:17, :), 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

U Appendix - Code to assess generalisation performance to different load factors

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M);

% load the CWRU data set, load=2HP, and stack features
```

```
load('data/processed/CWRU/c_EEMD_F1_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/c_EEMD_F1_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/c_EEMD_F1_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/c_EEMD_F1_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/c_EEMD_F1_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked, 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M);

% load the CWRU data set, load=2HP, and stack features
load('data/processed/CWRU/c_EEMD_F5_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/c_EEMD_F5_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/c_EEMD_F5_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/c_EEMD_F5_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/c_EEMD_F5_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked, 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M);

% load the CWRU data set, load=2HP, and stack features
load('data/processed/CWRU/c_EEMD_F6_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/c_EEMD_F6_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/c_EEMD_F6_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/c_EEMD_F6_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/c_EEMD_F6_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked, 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M);

% load the CWRU data set, and stack features
load('data/processed/CWRU/c_stats_F14_freq_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/c_stats_F14_freq_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/c_stats_F14_freq_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/c_stats_F14_freq_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/c_stats_F14_freq_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked, 1);

% set random seed
```

```
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M);

% load the CWRU data set, load=2HP, and stack features
load('data/processed/CWRU/c_stats_F19_freq_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/c_stats_F19_freq_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/c_stats_F19_freq_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/c_stats_F19_freq_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/c_stats_F19_freq_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked, 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros((Z + 1) * 3, M);

% load the CWRU data set, and stack EEMD feature #5
load('data/processed/CWRU/c_EEMD_F5_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/c_EEMD_F5_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/c_EEMD_F5_z1.mat');
F_stacked(3, :) = F;
```

```
load('data/processed/CWRU/c_EEMD_F5_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/c_EEMD_F5_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked(1:5, :), 1);

% load the CWRU data set, and stack EEMD feature #6
load('data/processed/CWRU/c_EEMD_F6_z0.mat');
F_stacked(7, :) = F;
load('data/processed/CWRU/c_EEMD_F6_z0.5.mat');
F_stacked(8, :) = F;
load('data/processed/CWRU/c_EEMD_F6_z1.mat');
F_stacked(9, :) = F;
load('data/processed/CWRU/c_EEMD_F6_z1.5.mat');
F_stacked(10, :) = F;
load('data/processed/CWRU/c_EEMD_F6_z2.mat');
F_stacked(11, :) = F;
F_stacked(12, :) = mean(F_stacked(7:11, :), 1);

% load the CWRU data set, and stack frequency−domain feature #14
load('data/processed/CWRU/c_stats_F14_freq_z0.mat');
F_stacked(13, :) = F;
load('data/processed/CWRU/c_stats_F14_freq_z0.5.mat');
F_stacked(14, :) = F;
load('data/processed/CWRU/c_stats_F14_freq_z1.mat');
F_stacked(15, :) = F;
load('data/processed/CWRU/c_stats_F14_freq_z1.5.mat');
F_stacked(16, :) = F;
load('data/processed/CWRU/c_stats_F14_freq_z2.mat');
F_stacked(17, :) = F;
F_stacked(18, :) = mean(F_stacked(13:17, :), 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M);

% load the CWRU data set, load=3HP, and stack features
load('data/processed/CWRU/d_EEMD_F1_z0.mat');
```

```
F_stacked(1, :) = F;
load('data/processed/CWRU/d_EEMD_F1_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/d_EEMD_F1_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/d_EEMD_F1_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/d_EEMD_F1_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked, 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M);

% load the CWRU data set, load=3HP, and stack features
load('data/processed/CWRU/d_EEMD_F5_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/d_EEMD_F5_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/d_EEMD_F5_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/d_EEMD_F5_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/d_EEMD_F5_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked, 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M);

% load the CWRU data set, load=3HP, and stack features
load('data/processed/CWRU/d_EEMD_F6_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/d_EEMD_F6_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/d_EEMD_F6_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/d_EEMD_F6_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/d_EEMD_F6_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked, 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M);

% load the CWRU data set, load=3HP, and stack features
load('data/processed/CWRU/d_stats_F14_freq_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/d_stats_F14_freq_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/d_stats_F14_freq_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/d_stats_F14_freq_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/d_stats_F14_freq_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked, 1);

% set random seed
setdemorandstream(491218382)
```

```
% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M);

% load the CWRU data set, load=3HP, and stack features
load('data/processed/CWRU/d_stats_F19_freq_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/d_stats_F19_freq_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/d_stats_F19_freq_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/d_stats_F19_freq_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/d_stats_F19_freq_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked, 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros((Z + 1) * 3, M);

% load the CWRU data set, load=3HP, and stack EEMD feature #5
load('data/processed/CWRU/d_EEMD_F5_z0.mat');
F_stacked(1, :) = F;
load('data/processed/CWRU/d_EEMD_F5_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/d_EEMD_F5_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/d_EEMD_F5_z1.5.mat');
```

```
F_stacked(4, :) = F;
load('data/processed/CWRU/d_EEMD_F5_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked(1:5, :), 1);

% load the CWRU data set, load=3HP, and stack EEMD feature #6
load('data/processed/CWRU/d_EEMD_F6_z0.mat');
F_stacked(7, :) = F;
load('data/processed/CWRU/d_EEMD_F6_z0.5.mat');
F_stacked(8, :) = F;
load('data/processed/CWRU/d_EEMD_F6_z1.mat');
F_stacked(9, :) = F;
load('data/processed/CWRU/d_EEMD_F6_z1.5.mat');
F_stacked(10, :) = F;
load('data/processed/CWRU/d_EEMD_F6_z2.mat');
F_stacked(11, :) = F;
F_stacked(12, :) = mean(F_stacked(7:11, :), 1);

% load the CWRU data set, load=3HP, and stack frequency-domain feature #14
load('data/processed/CWRU/d_stats_F14_freq_z0.mat');
F_stacked(13, :) = F;
load('data/processed/CWRU/d_stats_F14_freq_z0.5.mat');
F_stacked(14, :) = F;
load('data/processed/CWRU/d_stats_F14_freq_z1.mat');
F_stacked(15, :) = F;
load('data/processed/CWRU/d_stats_F14_freq_z1.5.mat');
F_stacked(16, :) = F;
load('data/processed/CWRU/d_stats_F14_freq_z2.mat');
F_stacked(17, :) = F;
F_stacked(18, :) = mean(F_stacked(13:17, :), 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M * 3);
T_stacked = zeros(7, M * 3);

% load the CWRU data set, load=1HP & load=2HP & load=3HP, and stack features
load('data/processed/CWRU/a_EEMD_F1_z0.mat');
```

```
F_stacked(1, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F1_z0.5.mat');
F_stacked(2, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F1_z1.mat');
F_stacked(3, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F1_z1.5.mat');
F_stacked(4, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F1_z2.mat');
F_stacked(5, 1:M) = F;
load('data/processed/CWRU/c_EEMD_F1_z0.mat');
F_stacked(1, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F1_z0.5.mat');
F_stacked(2, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F1_z1.mat');
F_stacked(3, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F1_z1.5.mat');
F_stacked(4, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F1_z2.mat');
F_stacked(5, 701:M*2) = F;
load('data/processed/CWRU/d_EEMD_F1_z0.mat');
F_stacked(1, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F1_z0.5.mat');
F_stacked(2, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F1_z1.mat');
F_stacked(3, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F1_z1.5.mat');
F_stacked(4, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F1_z2.mat');
F_stacked(5, 1401:M*3) = F;
F_stacked(6, :) = mean(F_stacked, 1);

T_stacked(:, 1:M) = T_onehot;
T_stacked(:, 701:M*2) = T_onehot;
T_stacked(:, 1401:M*3) = T_onehot;

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_stacked);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M * 3);
```

```
T_stacked = zeros(7, M * 3);

% load the CWRU data set, load=1HP & load=2HP & load=3HP, and stack features
load('data/processed/CWRU/a_EEMD_F5_z0.mat');
F_stacked(1, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F5_z0.5.mat');
F_stacked(2, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F5_z1.mat');
F_stacked(3, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F5_z1.5.mat');
F_stacked(4, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F5_z2.mat');
F_stacked(5, 1:M) = F;
load('data/processed/CWRU/c_EEMD_F5_z0.mat');
F_stacked(1, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F5_z0.5.mat');
F_stacked(2, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F5_z1.mat');
F_stacked(3, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F5_z1.5.mat');
F_stacked(4, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F5_z2.mat');
F_stacked(5, 701:M*2) = F;
load('data/processed/CWRU/d_EEMD_F5_z0.mat');
F_stacked(1, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F5_z0.5.mat');
F_stacked(2, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F5_z1.mat');
F_stacked(3, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F5_z1.5.mat');
F_stacked(4, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F5_z2.mat');
F_stacked(5, 1401:M*3) = F;
F_stacked(6, :) = mean(F_stacked, 1);

T_stacked(:, 1:M) = T_onehot;
T_stacked(:, 701:M*2) = T_onehot;
T_stacked(:, 1401:M*3) = T_onehot;

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_stacked);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M * 3);
T_stacked = zeros(7, M * 3);

% load the CWRU data set, load=1HP & load=2HP & load=3HP, and stack features
load('data/processed/CWRU/a_EEMD_F6_z0.mat');
F_stacked(1, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F6_z0.5.mat');
F_stacked(2, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F6_z1.mat');
F_stacked(3, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F6_z1.5.mat');
F_stacked(4, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F6_z2.mat');
F_stacked(5, 1:M) = F;
load('data/processed/CWRU/c_EEMD_F6_z0.mat');
F_stacked(1, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F6_z0.5.mat');
F_stacked(2, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F6_z1.mat');
F_stacked(3, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F6_z1.5.mat');
F_stacked(4, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F6_z2.mat');
F_stacked(5, 701:M*2) = F;
load('data/processed/CWRU/d_EEMD_F6_z0.mat');
F_stacked(1, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F6_z0.5.mat');
F_stacked(2, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F6_z1.mat');
F_stacked(3, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F6_z1.5.mat');
F_stacked(4, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F6_z2.mat');
F_stacked(5, 1401:M*3) = F;
F_stacked(6, :) = mean(F_stacked, 1);

T_stacked(:, 1:M) = T_onehot;
T_stacked(:, 701:M*2) = T_onehot;
T_stacked(:, 1401:M*3) = T_onehot;

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');
```

```
% train the pattern neural network
[net,tr] = train(net, F_stacked, T_stacked);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M * 3);
T_stacked = zeros(7, M * 3);

% load the CWRU data set, load=1HP & load=2HP & load=3HP, and stack features
load('data/processed/CWRU/a_stats_F14_freq_z0.mat');
F_stacked(1, 1:M) = F;
load('data/processed/CWRU/a_stats_F14_freq_z0.5.mat');
F_stacked(2, 1:M) = F;
load('data/processed/CWRU/a_stats_F14_freq_z1.mat');
F_stacked(3, 1:M) = F;
load('data/processed/CWRU/a_stats_F14_freq_z1.5.mat');
F_stacked(4, 1:M) = F;
load('data/processed/CWRU/a_stats_F14_freq_z2.mat');
F_stacked(5, 1:M) = F;
load('data/processed/CWRU/c_stats_F14_freq_z0.mat');
F_stacked(1, 701:M*2) = F;
load('data/processed/CWRU/c_stats_F14_freq_z0.5.mat');
F_stacked(2, 701:M*2) = F;
load('data/processed/CWRU/c_stats_F14_freq_z1.mat');
F_stacked(3, 701:M*2) = F;
load('data/processed/CWRU/c_stats_F14_freq_z1.5.mat');
F_stacked(4, 701:M*2) = F;
load('data/processed/CWRU/c_stats_F14_freq_z2.mat');
F_stacked(5, 701:M*2) = F;
load('data/processed/CWRU/d_stats_F14_freq_z0.mat');
F_stacked(1, 1401:M*3) = F;
load('data/processed/CWRU/d_stats_F14_freq_z0.5.mat');
F_stacked(2, 1401:M*3) = F;
load('data/processed/CWRU/d_stats_F14_freq_z1.mat');
F_stacked(3, 1401:M*3) = F;
load('data/processed/CWRU/d_stats_F14_freq_z1.5.mat');
F_stacked(4, 1401:M*3) = F;
load('data/processed/CWRU/d_stats_F14_freq_z2.mat');
F_stacked(5, 1401:M*3) = F;
F_stacked(6, :) = mean(F_stacked, 1);

T_stacked(:, 1:M) = T_onehot;
T_stacked(:, 701:M*2) = T_onehot;
T_stacked(:, 1401:M*3) = T_onehot;

% set random seed
```

```
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_stacked);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros(Z + 1, M * 3);
T_stacked = zeros(7, M * 3);

% load the CWRU data set, load=1HP & load=2HP & load=3HP, and stack features
load('data/processed/CWRU/a_stats_F19_freq_z0.mat');
F_stacked(1, 1:M) = F;
load('data/processed/CWRU/a_stats_F19_freq_z0.5.mat');
F_stacked(2, 1:M) = F;
load('data/processed/CWRU/a_stats_F19_freq_z1.mat');
F_stacked(3, 1:M) = F;
load('data/processed/CWRU/a_stats_F19_freq_z1.5.mat');
F_stacked(4, 1:M) = F;
load('data/processed/CWRU/a_stats_F19_freq_z2.mat');
F_stacked(5, 1:M) = F;
load('data/processed/CWRU/c_stats_F19_freq_z0.mat');
F_stacked(1, 701:M*2) = F;
load('data/processed/CWRU/c_stats_F19_freq_z0.5.mat');
F_stacked(2, 701:M*2) = F;
load('data/processed/CWRU/c_stats_F19_freq_z1.mat');
F_stacked(3, 701:M*2) = F;
load('data/processed/CWRU/c_stats_F19_freq_z1.5.mat');
F_stacked(4, 701:M*2) = F;
load('data/processed/CWRU/c_stats_F19_freq_z2.mat');
F_stacked(5, 701:M*2) = F;
load('data/processed/CWRU/d_stats_F19_freq_z0.mat');
F_stacked(1, 1401:M*3) = F;
load('data/processed/CWRU/d_stats_F19_freq_z0.5.mat');
F_stacked(2, 1401:M*3) = F;
load('data/processed/CWRU/d_stats_F19_freq_z1.mat');
F_stacked(3, 1401:M*3) = F;
load('data/processed/CWRU/d_stats_F19_freq_z1.5.mat');
F_stacked(4, 1401:M*3) = F;
load('data/processed/CWRU/d_stats_F19_freq_z2.mat');
F_stacked(5, 1401:M*3) = F;
F_stacked(6, :) = mean(F_stacked, 1);

T_stacked(:, 1:M) = T_onehot;
```

```
T_stacked(:, 701:M*2) = T_onehot;
T_stacked(:, 1401:M*3) = T_onehot;

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_stacked);
nntraintool
```

```
clear;
Z = 5;
M = 700;
F_stacked = zeros((Z + 1) * 3, M * 3);
T_stacked = zeros(7, M * 3);

% load the CWRU data set, load=1HP & load=2HP & load=3HP, and stack EEMD feature
    ↪ #5
load('data/processed/CWRU/a_EEMD_F5_z0.mat');
F_stacked(1, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F5_z0.5.mat');
F_stacked(2, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F5_z1.mat');
F_stacked(3, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F5_z1.5.mat');
F_stacked(4, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F5_z2.mat');
F_stacked(5, 1:M) = F;
load('data/processed/CWRU/c_EEMD_F5_z0.mat');
F_stacked(1, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F5_z0.5.mat');
F_stacked(2, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F5_z1.mat');
F_stacked(3, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F5_z1.5.mat');
F_stacked(4, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F5_z2.mat');
F_stacked(5, 701:M*2) = F;
load('data/processed/CWRU/d_EEMD_F5_z0.mat');
F_stacked(1, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F5_z0.5.mat');
F_stacked(2, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F5_z1.mat');
F_stacked(3, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F5_z1.5.mat');
F_stacked(4, 1401:M*3) = F;
```

```
load('data/processed/CWRU/d_EEMD_F5_z2.mat');
F_stacked(5, 1401:M*3) = F;
F_stacked(6, :) = mean(F_stacked(1:5, :), 1);

% load the CWRU data set, load=1HP & load=2HP & load=3HP, and stack EEMD feature
    ↪ #6
load('data/processed/CWRU/a_EEMD_F6_z0.mat');
F_stacked(7, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F6_z0.5.mat');
F_stacked(8, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F6_z1.mat');
F_stacked(9, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F6_z1.5.mat');
F_stacked(10, 1:M) = F;
load('data/processed/CWRU/a_EEMD_F6_z2.mat');
F_stacked(11, 1:M) = F;
load('data/processed/CWRU/c_EEMD_F6_z0.mat');
F_stacked(7, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F6_z0.5.mat');
F_stacked(8, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F6_z1.mat');
F_stacked(9, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F6_z1.5.mat');
F_stacked(10, 701:M*2) = F;
load('data/processed/CWRU/c_EEMD_F6_z2.mat');
F_stacked(11, 701:M*2) = F;
load('data/processed/CWRU/d_EEMD_F6_z0.mat');
F_stacked(7, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F6_z0.5.mat');
F_stacked(8, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F6_z1.mat');
F_stacked(9, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F6_z1.5.mat');
F_stacked(10, 1401:M*3) = F;
load('data/processed/CWRU/d_EEMD_F6_z2.mat');
F_stacked(11, 1401:M*3) = F;
F_stacked(12, :) = mean(F_stacked(7:11, :), 1);

% load the CWRU data set, load=1HP & load=2HP & load=3HP, and stack frequency-
    ↪ domain feature #14
load('data/processed/CWRU/a_stats_F14_freq_z0.mat');
F_stacked(13, 1:M) = F;
load('data/processed/CWRU/a_stats_F14_freq_z0.5.mat');
F_stacked(14, 1:M) = F;
load('data/processed/CWRU/a_stats_F14_freq_z1.mat');
F_stacked(15, 1:M) = F;
load('data/processed/CWRU/a_stats_F14_freq_z1.5.mat');
F_stacked(16, 1:M) = F;
load('data/processed/CWRU/a_stats_F14_freq_z2.mat');
```

```
F_stacked(17, 1:M) = F;
load('data/processed/CWRU/c_stats_F14_freq_z0.mat');
F_stacked(13, 701:M*2) = F;
load('data/processed/CWRU/c_stats_F14_freq_z0.5.mat');
F_stacked(14, 701:M*2) = F;
load('data/processed/CWRU/c_stats_F14_freq_z1.mat');
F_stacked(15, 701:M*2) = F;
load('data/processed/CWRU/c_stats_F14_freq_z1.5.mat');
F_stacked(16, 701:M*2) = F;
load('data/processed/CWRU/c_stats_F14_freq_z2.mat');
F_stacked(17, 701:M*2) = F;
load('data/processed/CWRU/d_stats_F14_freq_z0.mat');
F_stacked(13, 1401:M*3) = F;
load('data/processed/CWRU/d_stats_F14_freq_z0.5.mat');
F_stacked(14, 1401:M*3) = F;
load('data/processed/CWRU/d_stats_F14_freq_z1.mat');
F_stacked(15, 1401:M*3) = F;
load('data/processed/CWRU/d_stats_F14_freq_z1.5.mat');
F_stacked(16, 1401:M*3) = F;
load('data/processed/CWRU/d_stats_F14_freq_z2.mat');
F_stacked(17, 1401:M*3) = F;
F_stacked(18, :) = mean(F_stacked(13:17, :), 1);

T_stacked(:, 1:M) = T_onehot;
T_stacked(:, 701:M*2) = T_onehot;
T_stacked(:, 1401:M*3) = T_onehot;

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_stacked);
nntraintool
```

V Appendix - Code to assess robustness performance

```
clear;
Z = 5;
M = 700;
F_stacked = zeros((Z + 1) * 3, M);

% load the CWRU data set, and stack EEMD feature #5
load('data/processed/CWRU/a_EEMD_F5_z0.mat');
F_stacked(1, :) = F;
```

```
load('data/processed/CWRU/a_EEMD_F5_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/a_EEMD_F5_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/a_EEMD_F5_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/a_EEMD_F5_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked(1:5, :), 1);

% load the CWRU data set, and stack EEMD feature #6
load('data/processed/CWRU/a_EEMD_F6_z0.mat');
F_stacked(7, :) = F;
load('data/processed/CWRU/a_EEMD_F6_z0.5.mat');
F_stacked(8, :) = F;
load('data/processed/CWRU/a_EEMD_F6_z1.mat');
F_stacked(9, :) = F;
load('data/processed/CWRU/a_EEMD_F6_z1.5.mat');
F_stacked(10, :) = F;
load('data/processed/CWRU/a_EEMD_F6_z2.mat');
F_stacked(11, :) = F;
F_stacked(12, :) = mean(F_stacked(7:11, :), 1);

% load the CWRU data set, and stack frequency−domain feature #14
load('data/processed/CWRU/a_stats_F14_freq_z0.mat');
F_stacked(13, :) = F;
load('data/processed/CWRU/a_stats_F14_freq_z0.5.mat');
F_stacked(14, :) = F;
load('data/processed/CWRU/a_stats_F14_freq_z1.mat');
F_stacked(15, :) = F;
load('data/processed/CWRU/a_stats_F14_freq_z1.5.mat');
F_stacked(16, :) = F;
load('data/processed/CWRU/a_stats_F14_freq_z2.mat');
F_stacked(17, :) = F;
F_stacked(18, :) = mean(F_stacked(13:17, :), 1);

% set random seed
setdemorandstream(491218382)

% initialise the pattern neural network
net = patternnet([10], 'traingdx', 'crossentropy');

% train the pattern neural network
[net,tr] = train(net, F_stacked, T_onehot);
nntraintool

% load the CWRU data set, and stack EEMD feature #5
load('data/processed/CWRU/z_EEMD_F5_z0.mat');
F_stacked(1, :) = F;
```

```
load('data/processed/CWRU/z_EEMD_F5_z0.5.mat');
F_stacked(2, :) = F;
load('data/processed/CWRU/z_EEMD_F5_z1.mat');
F_stacked(3, :) = F;
load('data/processed/CWRU/z_EEMD_F5_z1.5.mat');
F_stacked(4, :) = F;
load('data/processed/CWRU/z_EEMD_F5_z2.mat');
F_stacked(5, :) = F;
F_stacked(6, :) = mean(F_stacked(1:5, :), 1);

% load the CWRU data set, and stack EEMD feature #6
load('data/processed/CWRU/z_EEMD_F6_z0.mat');
F_stacked(7, :) = F;
load('data/processed/CWRU/z_EEMD_F6_z0.5.mat');
F_stacked(8, :) = F;
load('data/processed/CWRU/z_EEMD_F6_z1.mat');
F_stacked(9, :) = F;
load('data/processed/CWRU/z_EEMD_F6_z1.5.mat');
F_stacked(10, :) = F;
load('data/processed/CWRU/z_EEMD_F6_z2.mat');
F_stacked(11, :) = F;
F_stacked(12, :) = mean(F_stacked(7:11, :), 1);

% load the CWRU data set, and stack frequency−domain feature #14
load('data/processed/CWRU/z_stats_F14_freq_z0.mat');
F_stacked(13, :) = F;
load('data/processed/CWRU/z_stats_F14_freq_z0.5.mat');
F_stacked(14, :) = F;
load('data/processed/CWRU/z_stats_F14_freq_z1.mat');
F_stacked(15, :) = F;
load('data/processed/CWRU/z_stats_F14_freq_z1.5.mat');
F_stacked(16, :) = F;
load('data/processed/CWRU/z_stats_F14_freq_z2.mat');
F_stacked(17, :) = F;
F_stacked(18, :) = mean(F_stacked(13:17, :), 1);

Y_onehot = net(F_stacked);
[~, Y_number] = max(Y_onehot, [], 1);
figure
robust_confusion_matrix = confusionchart(T_number, Y_number);
```