UNIVERSITY OF SHEFFIELD

Assignment I

Part I: Multisensor & Decision Systems

AWAB SYED

4/19/2021

Name: Awabullah M. Syed

Student Number: 2002 000 33

University Name: University of Sheffield

Module Code: ACS 6124

MATLAB code can be extracted via the QR Code

Contents

Section A (Lab A.I & Problem I. A)

Introduction

Sensor measurements/vibrations data from five test rigs shown in *Appendix A* were used to investigate/evaluate faults: *Fault 1 – Bearing Defect, Fault 2 – Gear Mesh, Fault 3 – Resonance, Fault 4 – Imbalance, Fault 5 – Misalignment* via Multisensor approach. As part of *Section A*, relevant features/parameters were derived from the vibration data to indicate the presence of possible faults and monitor the health condition of the machine. The extracted features deduced in *Section A: Lab A. I* was used in *Section B* to diagnose the rotating machine via pattern classification techniques / K-nearest neighbour classifier. Additionally, the MMSE estimator was designed and a two-sided CUSUM was carried out in *Section C* to alert/detect the fault in the wind turbine blade.

Task I: Vibration Signal Analysis

Time-Domain of each fault

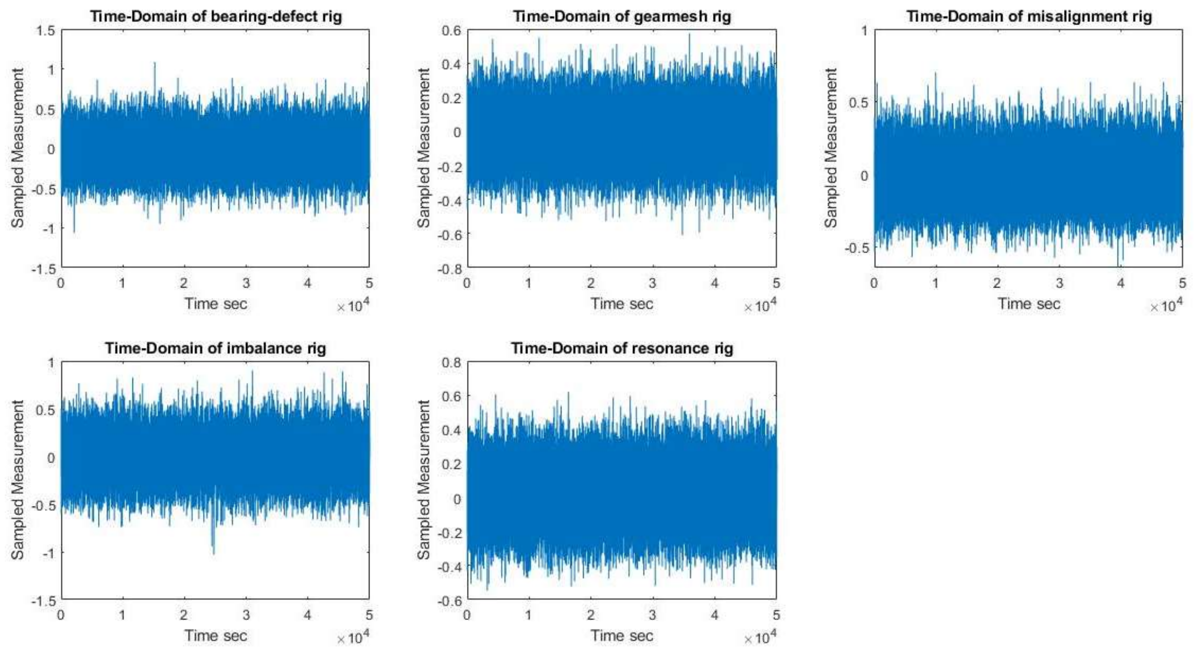


Figure 1 - Time-Domain plot of five faults; Bearing, Gear mesh, Misalignment, Imbalance, Resonance

Figure 1 shows the vibration signature of each fault: bearing, gearmesh, misalignment, imbalance, and resonance in a time-domain plot. The change in mean and variance detection are the key features to determine the type of fault that occurred i.e., abrupt fault, addictive fault, or multiplicative fault. For example, a change in mean value or the step-change in variance could suggest an abrupt fault in the sensor measurement while the increase in variance may indicate multiplicative fault i.e., machine tool wear (noise). Hence, the most important features extracted from the time-domain plot are mean and variance to differentiate the type of fault. However, the correlation between the sensor signal and faults in the time-domain was found to be rather poor and also did not show invariance in features concerning static transformation, further supported by (A. Noori-Khajavi, 1995). Consequently, power spectral densities/energy distribution of the sensor measurement were calculated using Matlab function, *'pwelch'* to evaluate the data in the frequency domain. Relevant MATLAB code is shown in *Appendix B.1: Time-Domain & Frequency Domain Analysis.*

Frequency-Domain, Power Spectral Density (PSD)

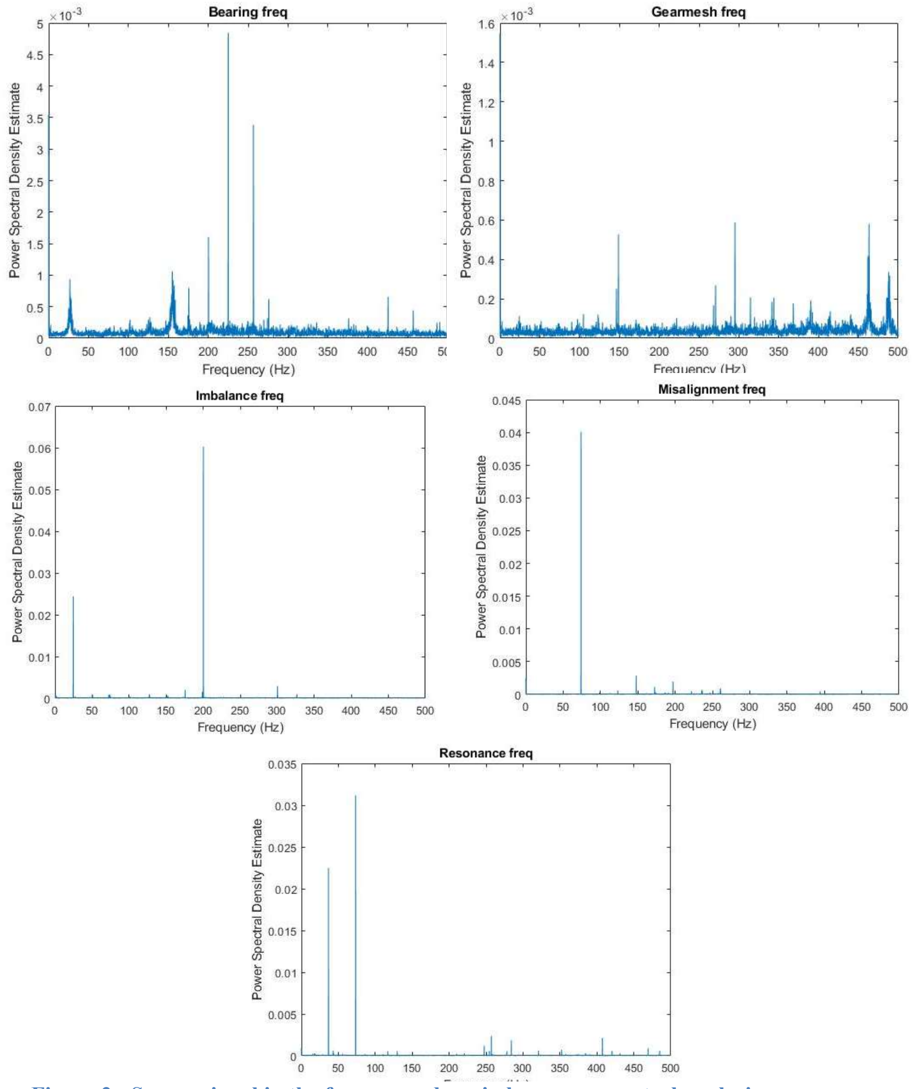


Figure 2 - Sensor signal in the frequency-domain by power spectral analysis

Figure 2 shows the power spectral density, PSD via frequency-domain analysis of individual faults used to estimate the energy distribution of the sensor signal. PSD in frequency-domain was used since it did not assume the underlying model structure of the system while relevant features including average energy within frequency bands, peak frequencies, amplitudes, and bandwidths can be deduced to determine and detect any fault i.e., bearing, gear mesh, misalignment, resonance, and imbalance from sensor measurements.

The spectral analysis shown in Figure 2 suggest a large variety of features/frequency bands extracted for fault diagnosis of rotating machinery shown in *Appendix A*.

Task II: Feature Extraction

The sensor data was divided/reshaped into $1000\ x\ 1$ blocks and four selected features shown in Table 1 were computed through the normalization of the data using Equation 1. Normalization was performed to convert the data with a common scale without affecting the differences in the ranges of values, enabling effective identification of faults. The normalized data was processed further to deduce the power spectral density of individual features using Equation 2 and then implemented appropriate *Butterworth* filter to features; f_2, f_3 and f_4 via MATLAB function, *butter.* The MATLAB code to undertake the process is shown in *Appendix B.2.* (Jaitley, 2018).

Feature Name	Detail of the Feature
f_1	$RMS\ of\ the\ signal$
f_2	$RMS\ of\ 0-50Hz$
f_3	$RM\ of\ 50-200\ Hz$
f_4	$RMS\ of\ 200-500Hz$

Table 1 - Relevant features to be extracted from sensor data.

$$x_{Normalized} = x - x_{mean} \qquad Eq.(1)$$

$$f_1 = \frac{\|P_i\|_2}{\sqrt{N}} \qquad Eq.(2)$$

$$[B, A] = butter(order, \omega_N) \qquad Eq.(3)$$

$$\omega_N = \frac{cuttoff\ frequency}{\frac{T_s}{2}} \qquad Eq.(4)$$

Butterworth filter was applied to the normalized data of the sensor using Equation 3, the output was then used in conjunction with the PSD to extract features using Equation 2. The desired four features of signal energy levels in different frequency bands for each fault were composed into a matrix, resulting in five fault matrices of each fault: bearing, gearmesh, imbalance, resonance, misalignment which was composed into a matrix, G shown in Equation 5.

Task III: Data Visualization

$$G = \begin{bmatrix} Fault\ 1(bearing) = [f1\,, f2\,, f3\,, f4] \\ Fault\ 2(gearmesh) = [f1\,, f2\,, f3\,, f4] \\ Fault\ 3(imbalance) = [f1\,, f2\,, f3\,, f4] \\ Fault\ 4(misalignment) = [f1\,, f2\,, f3\,, f4] \\ Fault\ 5(resonance) = [f1\,, f2\,, f3\,, f4] \end{bmatrix} \qquad Eq.(5)$$

Since each fault had four features/dimensions, it was unfeasible to visualize the data hence dimension reduction approach was taken. Four dimensions were mapped to two dimensions via Principal Component Analysis (PCA) in which eigendecomposition was utilized to determine the order of eigenvalues and eigenvectors. Hence, PCA allowed exploring/visualizing high-dimensional data easily. The MATLAB code to carry out the process is shown in *Appendix B.3.*

Problem I. A – Energy Levels in six frequency bands

The same process from *Lab A. I* was followed from the normalization of the sensor data to extracting energy levels but on this occasion, six frequency bands shown in Table 2 were considered.

Feature Name	Filter	Filter Order	ω_N
f_1	Low-Pass 25 Hz	7	0.05
f_2	Band-pass 25 – 50Hz	6	[0.05 0.1]
f_3	Band-pass 50 – 100Hz	9	[0.1 0.2]
f_4	Band-pass 100 – 200Hz	8	[0.2 0.4]
f_5	Band-pass 200-350Hz	9	[0.4 0.7]
f_6	High pass 350Hz	16	0.7

Table 2 - Six features extracted - Problem I. A

Principal Component Analysis, PCA was conducted as shown in *Appendix B.3* and *Appendix C* to reduce the dimensions from four and six to two dimensions, respectively, for interpretation and visualization purposes. Features, z_1 and z_2 obtained from PCA were plotted consisting of the following faults:

Fault 1 – Bearing Fault

Fault 2 – Gear Mesh

Fault 3 – Imbalance

Fault 4 – Misalignment

Fault 5 – Resonance

Critical Analysis / Comparison

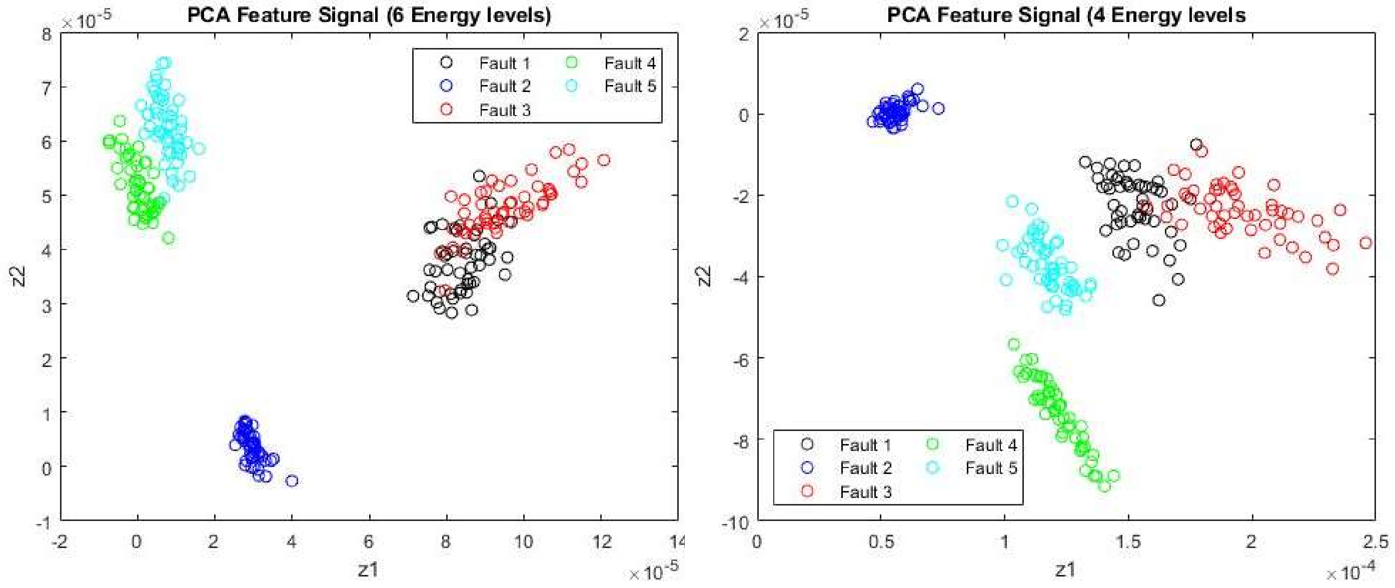


Figure 3 - 2-Dimensional feature vector, z obtained via PCA is plotted. The plot on the left is of Problem I.A with six frequency bands and the plot on the right is of Lab A.I with four frequency bands.

The plot on the *right* side of Figure 3 is obtained from *Lab I. A* where four features shown in Table 1 were reduced while the plot on the *left* of Figure 3 is obtained from *Problem I.A* where six features shown in Table 2 were reduced to two features consisting of important details via Principal Component Analysis, PCA for better interpretation/visualization.

In four features plot, each fault type is completely separatable from others except *Fault 1 - Bearing* and *Fault 3 - Imbalance* which slightly overlaps without a distinct boundary in terms of feature, z_1 while both have a similar range of feature, z_2. This may cause difficulty in distinguishing between *Fault 1* and *Fault 3,* generating 'ambiguity' as well as increasing the probability of a false alarm which may be tackled by utilizing statistical decision theory.

The faults of six features plot in comparison with 4 features are relatively (Approx. 10 times more) compact together particularly *Fault 4 - Misalignment* and *Fault 5 - Resonance* while the overlap between *Fault 1* and *Fault 3* is further increased with the overall range of each fault with respect to z_1 and z_2 being reduced. One of the major factors contributing to this is the dimension reduction from six features to two (z_1 & z_2) via PCA method as it is much more likely that important parameters/details are neglected or not considered when compared with dimension reduction from four to two. On the other hand, feature data being compact/packed closely is due to smaller frequency filter range for six features which resulted in increased precision but also increased the risk of losing or not able to consider critical information due to dimension reduction, PCA. Therefore, increasing the number of features, makes the feature data concentrated but increases the chance of losing relevant information due to PCA dimensional reduction. In other words, as the number of features increases to six, there is a trade-off observed between detecting fault (precision) and overlap (losing crucial information) as well as the computational power/execution time being increased with the number of features.

One of the most important points mentioned by the author (Svante Wold, 1987) is the scaling issue of axes which is unfortunately usually merged with Principal Component Analysis however, this can be avoided/reduced by considering the '*variance scaling',* where each variable is scaled to a unit variance as suggested by (Jollifee). Though, this is not extensively covered in this report since plots in Figure 3 are not greatly impacted by it.

Section B (Lab A.II & Problem I. B)

Introduction

Simple classifier, *'K-nearest neighbour classifier* was used to classify machine conditions using two features, z_1 and z_2 obtained from applying Principal Component Analysis (PCA) on four features shown in Table 1 *(Lab A.I)*. The features' data attained from PCA was first partitioned for training (first 35 rows) and testing (remaining 15 rows) and then the nearest neighbour classifier was implemented. The training data was used to design the algorithm while the testing data was used to evaluate the accuracy of the classifier. K-nearest neighbour algorithm was later slightly modified to consider, compare, and evaluate different k-values i.e., 1, 3 and 5 in terms of classification accuracy. Additionally, possible suggestions including alternative approaches and improvements to the nearest neighbour algorithm were made but these were not implemented.

Lab A. II - 1-Nearest Neighbors Algorithm

1-Nearest Neighbor Algorithm was first implemented on two features data from *Lab A. I* via MATLAB code shown in *Appendix D.2* incorporated with *Euclidean* distance measure function shown in *Appendix D.1.*

1-Nearest Neighbor classifies the new/coming measurement based on the class of one of its nearest neighbours. In other words, the algorithm finds the similarity between the *training* measurement and the new/*test* measurement (Devi). Following assumptions were made while considering the data for each fault:

$Fault1 \cap Fault2 \cap Fault3 \ldots . = \phi$

$Fault1 \cup Fault2 \cup Fault3 \ldots = \mathbb{U}$

It was assumed that the union of all the faults covered the entire data space and the intersection was zero, mutually exclusive.

Apart from detecting a fault, applying the 1-Nearest Neighbor Algorithm enabled to classify the type of faults/conditions based on the *training* measurement with an accuracy of 90.667%. Several factors contributed to obtaining a 90% accuracy, an important factor that significantly impacted the accuracy was the dimension reduction via the PCA approach since this reduced the amount of feature information from the data. The second factor being the slight overlap between *Fault 1 – Bearing* and *Fault 3 – Imbalance* as indicated by the plot on the *right* of Figure 3. However, the initial normalization of the data using Equation 1 allowed to significantly increase the classification accuracy.

Problem I.B - K-Nearest Neighbors Algorithm

To reduce the impact of factors affecting the accuracy (i.e., the overlap) of the classifier, the K-nearest neighbour algorithm was implemented with K being three and five, the MATLAB code is shown in *Appendix D.3*.

MATLAB function, *mink* was used to consider K nearest/smallest elements of the variable, *currentDist* and returns them to the variable, *I,* as illustrated below (for K = 3). The second argument of the *mink* function is the value of K. For example when K = 3, three closest training samples were selected based on euclidean distance and the weighting applied. If two selected nearest belongs to *Fault 4* while one to *Fault 5*, the test sample classified as *Fault 4*. Hence, K-values were deliberately chosen to be odd to avoid the stalemate after weighting applied to the nearest neighbours.

```
[shortestDistance,I] = mink(currentDist,3); % for k =3
```

MATLAB function, *mode* was then used to return the most frequently occurring value of the variable, *shortestDistanceLabel_temp* since multiple (three or five) values occur equally frequently.

```
[shortestDistanceLabel,F] = mode(shortestDistanceLabel_temp);
```

The entire MATLAB code for the K-Nearest Neighbor algorithm is shown in *Appendix D.3* but a brief explanation is provided above for only the important sections of the code.

Nearest Neighbor Algorithm	
$K - Value$	**Classifier Accuracy (%)**
1	90.337
3	93.33
5	93.33
Initial 4 feature data (PCA Not Applied)	
1	98.66%

Table 3 - Accuracy of the K-Nearest Neighbor Algorithm with respect to K-value

Table 3 shows the impact of the K-value on the accuracy of the algorithm. An apparent trend was observed, as the value of K (Number of nearest neighbour) increases from one to three, so does the accuracy, however, the accuracy plateaus to 93.33% beyond the K-value of 3. The increase in accuracy from 90.337% to 93.337% was mainly by significantly neutralizing the impact of slight overlapping between *Fault 1* and *Fault 3* since K being more than one considers numerous nearest neighbourhood values rather than only one, allowing to improve the accuracy. This does not necessarily mean that increasing the K-value completely negate the impact of various factors discussed above but it certainly works as a counteracting force. It was also observed that increasing the K-value, increased the computation time but allowed to reduce the effect of noise on the result/accuracy. However, further improvement in the accuracy was not observed by increasing the K-value. This could be due to the implementation of Principal Component Analysis, PCA on the data to reduce the number of features from four to two for interpretation purposes but consequently, it marginally affected the details/information of the data. Hence, unable to improve the accuracy further via increasing K-value. Moreover, K being more than five was not considered since it may defeat the core philosophy behind the K-nearest algorithm of the nearest neighbourhood having the same density.

K-Nearest Neighbor Algorithm was implemented on the initial features/fault data (before PCA applied) as any reduction in the accuracy due to this can be observed. Although, this is not fully covered in this report but the accuracy of the 1-Nearest Neighbor Algorithm implemented on the initial fault data was 98.33% which further supports the argument made.

Alternative Approaches & Improvements

Several ways and methods can be used to improve the Nearest Neighbor Algorithm, some being already implemented during the implementation in MATLAB while other improvements, and approaches, could be beneficial for future or industrial implementation.

Part A of *Appendix D.4* was replaced by *Part B* which allowed to significantly reduce the overall execution time of the algorithm, allowing the code to swiftly classify the data and detect faults. Hence, this was implemented while executing MATLAB code.

Furthermore, *Artificial Neural Network, ANN* which is similar to the KNN algorithm in terms of the designing process (Figure 4), both trained via *training data set* and used on the *test data* with variable, K replaced by hidden neuron count, N in ANN algorithm. In ANN, inputs are multiplied by weights via *activation* function to transform the node into an output. Activation functions are generally classified into three categories, *Ridge* (Linear, logistic, ReLU), *Radial Basis* (Gaussian, Multiquadric) and *Folding activation* functions. In terms of monitoring the health and classifying the faults of the rotating machine (bearing, gearmesh, imbalance etc.), the *Radial basis function* may be the ideal category to explore further (Adnan Hamad, 2010).

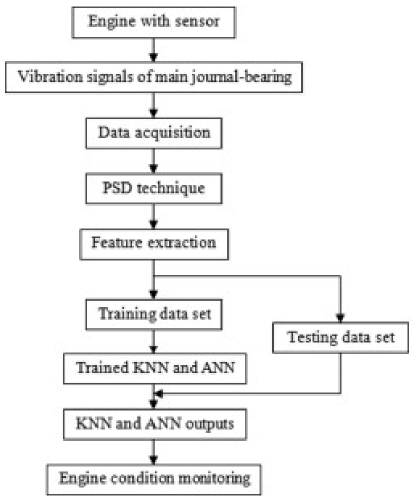


Figure 4 - General process to determining the health condition of a system (Ahmadi, 2012)

Several factors affect the performance and the accuracy of the ANN so while implementing the ANN algorithm special attention must be given to the hidden layers, network model, network size, momentum and learning rate as well as the type of *activation* function being employed, to obtain optimal accuracy.

Once effectively implemented, *Artificial Neural Network,* ANN will have higher classification accuracy (Approx. ~98%) with minimum execution time compared with KNN (~90 – 93%) largely due to hidden layers. ANN has a lower *mean absolute error* and *relative absolute error* compared with the K-Nearest Neighbor algorithm but more importantly, ANN will be able to process feature-rich data due to the complex network model. This means that ANN will be better off if considering the six features (Table 2) data (Mustafa).

Although ANN is much more complex than KNN with several parameters that need to be set before designing the ANN model i.e., learning parameters but it can be leveraged by having the option to utilize unsupervised learning which will enable the algorithm to extract features of the unlabeled data and can also be incorporated with transfer learning. This will allow the system designed for the rotating machine to be easily expandable to different machines or integrate new faults without starting from scratch while incorporating additional *training* data for better accuracy due to transfer learning.

Furthermore, it would be worth considering integrating an optimization scheme to minimize classification error or cost i.e., the *Moth Optimization* algorithm; a population-based approach to select relevant informative features from the data to improve the accuracy and reduce the computational power. MFO algorithm always moves towards the optimal solution hence, the MFO algorithm may enable to extract relevant features from the data that distinguishes the faults and reduces 'ambiguity' while the KNN algorithm acts as a classifier. This needs to be further analyzed and evaluated to come up with an approach that effectively incorporates both algorithms together, a starting point could be to critically examine the latest published article on using MFO as a wrapping method for feature selection problem (Alzaqebah, 2020).

Additionally, the *gradient descent backpropagation* method could also be utilized to train the ANN, enabling it to classify the data through evaluating the local nearest minimum class. Though this is not further rationalized but relevant material can be extracted from (Imdad), where the author used ANN and KNN to classify students results. Lastly, An alternative to all this could also be to use the in-built MATLAB function, *fitcknn* or *Classification Learner Ap* to implement KNN in MATLAB (outside the scope of this report). This will reduce the No. of code lines/the execution time but will have no impact on the accuracy. Though the accuracy can be improved by bringing more *training* measurements onboard while utilizing parallel programming may enable higher dimension visualization and accelerate computation.

Section C – Wind Turbine mfg. Co.

Introduction

Multisensor signal estimation and health monitoring system was designed and implemented for a wind turbine manufacturing company with prior knowledge of the *pitch angle* and the *sensor noise* being uniformly distributed in the range $0° \leq \widehat{\omega} \leq 30°$ and normally distributed, $v \sim N(0, 9)$, respectively. Sensor measurement vector, *encoder.mat* was used and can be extracted via QR code provided on the report cover page. Two scenarios were considered, *Case 1 – Entire measurement vector encoder.mat available* and *Case 2 – Only the first five elements of the measurement available.* The MATLAB code was written to calculate the MMSE estimator to predict the true value of the sensor (without noise) for both the scenarios and was extended with the introduction of prior knowledge of w_t, being Gaussian distributed. Lastly, the measurement of the vertical to the rotating axis bending moment of the blade (*straingauge. mat*) was extracted from the strain gauge sensor. This was used to design a two-sided CUSUM test algorithm to detect and alert for any fault under the assumption that normal operation of the system being $3000\ KNm$ with the variance of 1 and a threshold of ± 20 while leakage, γ being ignored.

Part A: MMSE Estimator – Prior Knowledge Uniformly Distributed

Estimator, x^* shown in Equation 9 was deduced that minimizes the MSE through deriving the joint PDF via Bayes' Theorem shown in Equation 6 and partially differentiating Equation 8 with respect to x^* along with algebraic manipulation.

$$p(x|y) = \frac{p(y|x)p(x)}{\int_{-\infty}^{\infty} p(y|x)p(x)dx} \qquad Eq.(6)$$

$$p(Y_{1:T}, x) = p(x|Y_{1:T})\, p(Y_{1:T}) \qquad Eq.(7)$$

<u>*Assumption*</u>*:* x and v_t are independent for all t

<u>*NOTE:*</u> Multiplication of Gaussian PDF results in Gaussian PDF with different mean value.

$$\frac{\partial}{\partial x *} \int (x - x^*)^2 p(X|Y_{1:T})dx = 0 \qquad Eq.(8)$$

$$x^* = \frac{\int_{-\infty}^{\infty} \frac{x}{\sqrt{2\pi \frac{\sigma^2}{T}}} exp\left[-\frac{1}{2\frac{\sigma^2}{T}}\left(X - \frac{1}{T}\sum_{t=1}^{T} y_t\right)^2\right] dx}{\int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi \frac{\sigma^2}{T}}} exp\left[-\frac{1}{2\frac{\sigma^2}{T}}\left(X - \frac{1}{T}\sum_{t=1}^{T} y_t\right)^2\right] dx} \qquad Eq.(9)$$

$$x^* = \mathrm{E}[(X|Y_{1:T})] \qquad Eq.(10)$$

MMSE estimator shown in Equation 9 was implemented in MATLAB for both scenarios, *Case 1,* and *Case 2.*

Part B: MMSE Estimator – Prior Knowledge Gaussian Distributed

The methodology followed in Part A was extended to calculate the MMSE estimator, x^* with the prior knowledge, ω being Gaussian distributed with a mean of 15 and variance of four.

$$p(x) = \frac{1}{\sqrt{2\pi{\sigma_x}^2}} exp\left[-\frac{1}{2{\sigma_x}^2}(X-\mu_x)^2\right] \qquad Eq.(11)$$

$$p(Y_{1:T}|X) = \frac{1}{(2\pi\sigma^2)^{T/2}} exp\left[-\frac{1}{2\sigma^2}\sum_{t=1}^{T}(y_t - X)^2\right] \qquad Eq.(12)$$

$$x^* = \mathrm{E}(X|Y_{1:T}) = \frac{\frac{1}{\sigma^2}\left(\sum_{t=1}^{T} y_t\right) + \frac{\mu_x}{{\sigma_x}^2}}{\frac{T}{\sigma^2} + \frac{1}{{\sigma_x}^2}} \qquad Eq.(13)$$

$$p(X|Y_{1:T}) = \frac{p(Y_{1:T}|X)p(X)}{\int p(Y_{1:T}|X)\, p(X)dx} = \frac{1}{\left(2\pi{\sigma^2}_{x|Y_{1:T}}\right)^{1/2}} exp\left[-\frac{1}{2\sigma^2}\sum_{t=1}^{T}(y_t - x)^2\right] \qquad Eq.(14)$$

<u>*NOTE*</u>: if no observation then $x^* = \mathrm{E}(x) = 0$

The probability density function, PDF of Gaussian distribution is shown in Equation 11 which was derived along with the likelihood function (Equation 12) through Bayes' Theorem (Equation 6) to obtain Bayesian Estimation with Gaussian priors shown in Equation 13. Equation 14 reaffirms that the algebraic posterior probability is Gaussian distributed with a different mean value of 15 and variance of 4. MMSE estimator, x^* shown in Equation 13 was implemented in MATLAB with the script shown in the next section: '*MATLAB Code'.*

MATLAB Code

```
%Observation model y(t) = x(t) + v(t) | x(t) = ax(t-1) + w(t) (Part A)
%Prior Knowledge
% sensor noise v ~ N(0,9)
% X - Uniformly distributed (0,30)
load encoder.mat
%---------------------------Part A---------------------------------------%
T = 100; % Data (Encoder) rows / measurement
var_noise = 9; % variance
max = 30;
min = 0;
k = 1;
for T = [1: 100]  % Case 1 = [1:100]  & Case 2 = [1:5]
    t = var_noise/T;
    t1 = sqrt(2*pi*t);
    x_mean = mean(encoder(1:T));

    numerator = @(x) ((x/t1) .* exp(-((x-x_mean).^2 / (2*t)))); % Eqn. (9)
    denominator = @(x) ((1/t1) .* exp(-(x-x_mean).^2 / (2*t))); % Eqn. (9)

    num_int = integral(numerator,0,30); % 0 & 30 - limits
    den_int = integral(denominator,0,30); % 0 & 30 - limits

    x_MMSE(k) = num_int / den_int;
    k = k+1;
    plot (x_MMSE)
    xlabel ('MMSE')
    ylabel ('Measurement')
    title ('MMSE wrt Measurement')
end
%----------------------------------Part B--------------------------------%
var_x=4; % Variance of 4
mean_x=15; % mean value of 15
i=1;
x_MMSE_Gaus = zeros(1,2); %Pre-allocation for speed
for T=[1: 100] % Case 1 = [1:100]  & Case 2 = [1:5]
    x_mean=mean(encoder(1:T))*T;
    x_MMSE_Gaus(i)=(x_mean/var_noise + mean_x/var_x)/(T/var_noise +
1/var_x);   %Equation (13)
    i=i+1;
    plot (x_MMSE_Gaus)
    xlabel ('MMSE')
    ylabel ('Measurement')
    title ('MMSE wrt Measurement - Gaussian Dist')
end
```

NOTE: T = [1:5] to consider the second scenario where only first five elements of the measurement were available.

Relevant Plots / Data

Part A – Normally Distributed

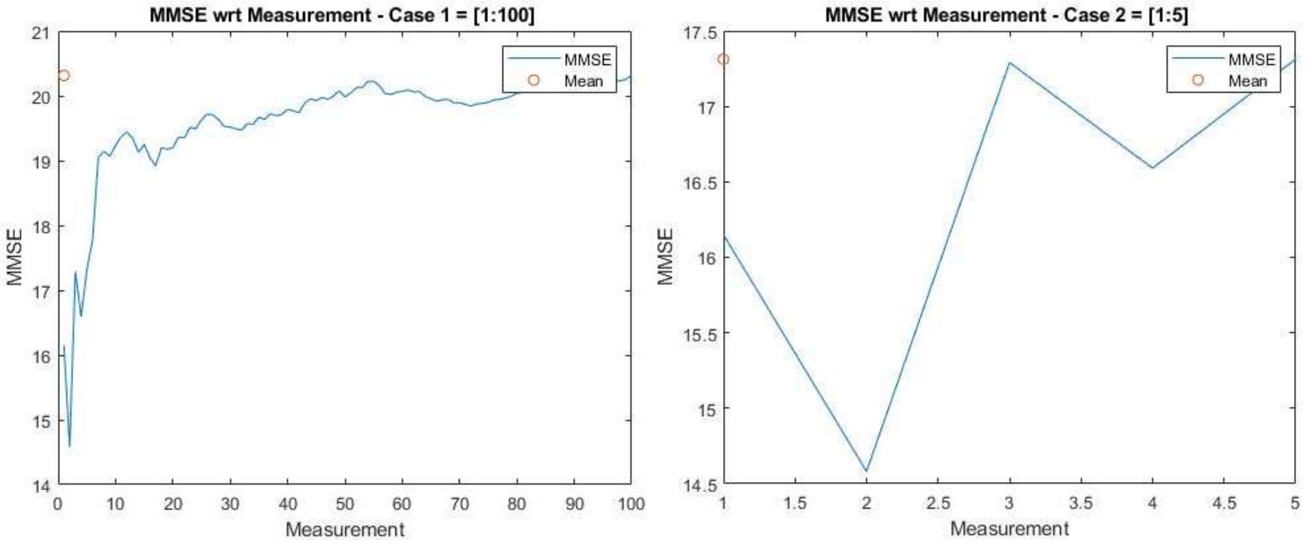


Figure 5 - Tracking MMSE estimator for both scenarios: Case 1 - Entire measurement & Case 2 - Only five first elements

Normally Distributed	
Scenario	$MMSE\ Estimator, x^*$
Case 1 – Entire measurement @T=100	20.3131
Case 2 – Only first five elements @T = 5	17.3125

Table 4 - x^* that minimizes the MSE in both scenarios | Normally Distributed.

Part B – Gaussian Distributed

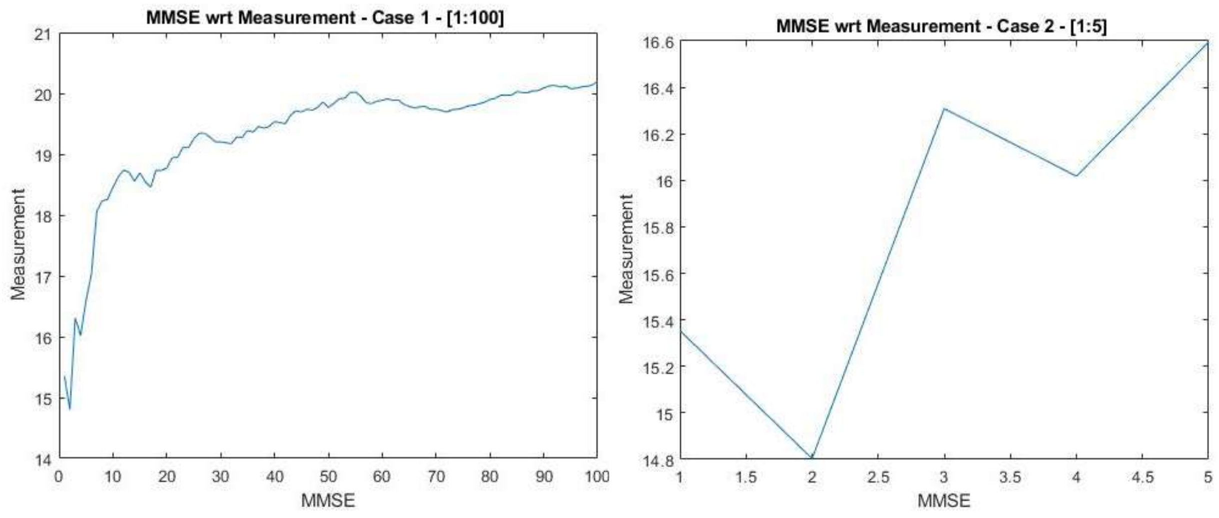


Figure 6 - Tracking MMSE estimator with the prior knowledge being Gaussian distributed - mean value of 15 & variance of 4.

Gaussian Distributed	
Scenario	$MMSE\ Estimator, x^*$
Case 1 – Entire measurement @T=100	20.1962
Case 2 – Only first five elements @T = 5	16.5948

Table 5 - that minimizes MSE in both scenarios | Gaussian Distributed.

Critical Analysis / Comparison

Figure 5 shows two plots, the entire measurement vector of the *encoder.mat* was available for the plot on the *left* while only first five measurements were available for the plot on the *right.* Table 4 shows the final MMSE estimator value for both scenarios by eliminating sensor noise which is normally distributed, $v \sim N(0,9)$. It was observed during the mathematical modelling/process that increasing the number of measurements, T, shifts the MMSE estimator towards the measurement and away from the prior knowledge. In other words, as the measurements increases, the MMSE estimator begins to rely less on the prior knowledge provided and more on the data. In the first scenario with the entire measurement available, the estimator relies more on the data and moves toward non-Bayesian case as $\frac{\sigma^2}{T}$ goes to zero. While, on the other hand in the second case with only the first five measurements available, the estimator relies heavily on the prior knowledge being normally distributed, $v \sim (0,9)$ but slightly on the five measurements available. Hence, measurements provide a balance between the weight of the prior knowledge and the evidence arising from the data. It was also observed that with no measurements, estimator, $x^* = \mathrm{E}(x) = 0$ is optimal Bayesian, no interface by the data.

In *Part B,* Gaussian distributed prior knowledge was introduced to the system with a mean value of 15 and variance of four. The plot on the *left* of Figure 6 is the result of the entire measurements available and considering the MMSE estimator mathematically illustrated in Equation 13, it was deduced that having more measurements enabled ${\sigma_x}^2$ to dominate $\frac{\sigma^2}{T}$ which allowed the MMSE estimator to be close to the mean value from the measurements, y_t, $\mathrm{E}(x|Y_{1:T}) \approx \frac{1}{T}\sum_{t=1}^{T} y_t$.Whereas on the other hand, the second scenario with only the first five elements, plot on the *right* of Figure 6, this time $\frac{\sigma^2}{T}$ dominates ${\sigma_x}^2$ which meant that MMSE estimator is now closely aligned with the mean value of prior knowledge, μ_x rather than of the measurements. Ideal scenario will be a balance between the prior knowledge and the amount of measurement data available as was the case in *Part A*.

$$\sigma^2{}_{x|Y_{1:T}} = \left(\frac{T}{\sigma^2} + \frac{1}{\sigma^2{}_x}\right)^{-1} = \frac{\sigma^2{}_x\sigma^2}{T\sigma^2{}_x + \sigma^2} = \frac{\sigma^2}{T\sigma^2{}_x + \sigma^2}\sigma^2{}_x = \frac{\sigma^2{}_x}{\sigma^2{}_x + \frac{\sigma^2}{T}}\frac{\sigma^2}{T} \qquad Eq.(15)$$

The MMSE estimator shown in Equation 13 was further derived to determine Equation 15 which indicates that combining prior information with the measurement data resulted in reduced variance consequently reducing uncertainty hence leading to a better MMSE estimator.

Two-sided CUSUM Test

A two-sided CUSUM test was designed to detect faults in the sensor measurement, ‘*straingauge.mat’* with the normal operation of the system being $3\,000\ KNm$, a variance of one and a threshold of ± 20, possible drift, γ was ignored.

$$S_t = \left\{\frac{y_t - \theta_0}{\sigma_0}\right\} \sim \mathrm{N}(\,0, 1\,) \qquad Eq.(16)$$

$$g_t = g_{t-1} + \left(\frac{y_t - \theta_0}{\sigma_0}\right) \qquad Eq.(17)$$

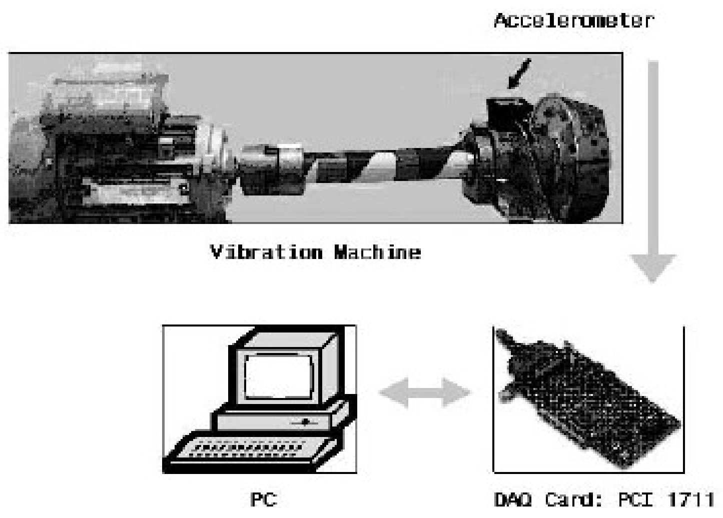
Equation 16 is commonly used to indicate if the system is normal and was used to determine the appropriate threshold for the decision rule however, in this case, the threshold was assumed to be ± 20 hence, it was not needed to derive Equation 16 using $g_T = \sum_{t=1}^{T} S_t$, a classical CUSUM approach (Hardcastle, 1996). The MATLAB code shown below was executed to detect if the measurement exceeds the set threshold.

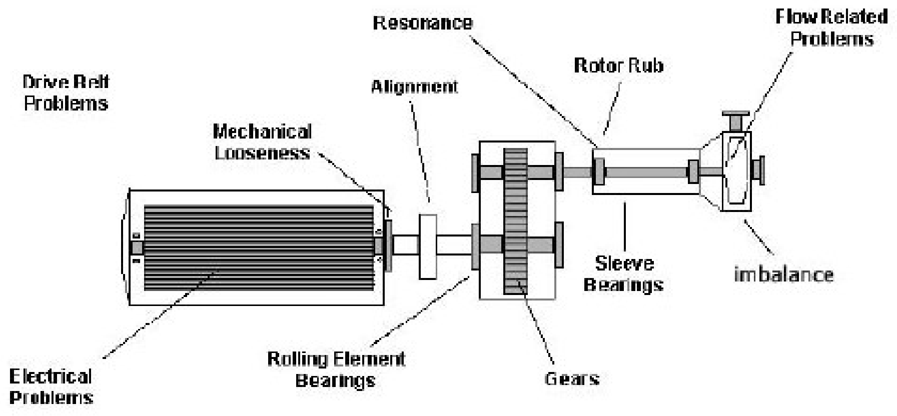
```
%---------------------------Part C: Two-Sided CUSUM Test ------------------%
load straingauge.mat;
theta0 = 3000;
sigma0 = 1;
threshold=20;
straingauge = [0,straingauge];
T = length(straingauge);

g(1)=0; % Initialization / Starts from zero value
%NOTE: Possible drift(gamma) is ignored
gamma = 0; % Therefore gamma = 0
g(1) = 0;
for j=1:T-1
    g(j+1)=g(j)+(straingauge(j+1)-theta0)/sigma0 + gamma;  % Equation 17
    if abs(g(j+1)) > threshold % if threshold crossed
        fault_on_set_time = j+1; %Fault occurs / Note the fault on set time
        disp(['Fault occured at ',num2str(fault_on_set_time)])
       % break;
    end
end
```

The MATLAB code shown above was executed which indicated two faults (exceeded the limit), first at index 3 ($T = 2$) and the second at index 9 ($T = 8$). The code was purposefully not set to stop after detecting the first change/fault via *break* code since this allowed to detect all the measurements exceeding the limit. Although, when implementing it in an industrial zone, it may be ideal to *break* the code and stop the system when the threshold exceeded.

Appendix

Appendix A: Machinery Vibration Monitoring System / Fault Types





Appendix B: MATLAB Code – Lab A.I

Appendix B.1: Vibration Signal - Time-Domain & Frequency Domain Analysis (PSD Plot)

```
% Author: Awabullah M Syed
% Date: 12 March 2021
% Description: Multisensor and Decision Systems of Test Rig: Bearing Defect
%   Gear Mesh, Resonance, Imbalance and Misalignment and extracting
%   informative features for health monitoring purpose

% NOTE: Please run the code section-wise

%-----------------------Lab A.I-------------------------------------------%

%-----------------------Task 1: Vibration Signal Analysis----------------%
%Load Files (ALWAYS RUN THIS SECTION)
load bearing.mat %Vibration MEAS of bearing-defect (Ts = 50sec,fs = 1 000)
load gearmesh.mat %Vibration MEAS of gearmesh rig (Ts = 50sec, fs = 1 000)
load misalignment.mat % MEAS of misalignment rig (Ts = 50sec, fs = 1 000)
load imbalance.mat % MEAS of imbalance rig (Ts = 50sec, fs = 1 000)
load resonance.mat % MEAS of resonance rig (Ts = 50sec, fs = 1 000)

Ts = 50;
Fs = 1000;
T = 1 / Fs;
N = 50000; %Measurement
t = (0:N-1); %time
%----------------------Time-Domain Plots---------------------------------%
featurename = {'bearing','gearmesh','misalignment','imbalance','resonance'};
feature = [bearing,gearmesh,misalignment,imbalance,resonance];
% Plot the figures of each fault in time domain
    for i=1:5
        subplot(5,1,i);
        plot(t,feature(:,i));
        xlabel('time'),ylabel(featurename{i});
        title(['Figure of ',featurename{i},' in time domain']);
    end

figure (6)  %Full-screen the plot for better viewing
subplot(2,3,1)
plot (t,bearing)  %plot for bearing measurement
title ('Time-Domain of bearing-defect rig')
xlabel('Time sec')
ylabel ('Sampled Measurement')
subplot(2,3,2)
plot (t,gearmesh) % plot for gearmesh meassurements
title ('Time-Domain of gearmesh rig')
xlabel('Time sec')
ylabel ('Sampled Measurement')
```

```
subplot(2,3,3)
plot (t,misalignment) % plot for misalignment rig meassurements
title ('Time-Domain of misalignment rig')
xlabel('Time sec')
ylabel ('Sampled Measurement')
subplot(2,3,4)
plot (t,imbalance) % plot for imbalance rig meassurements
title ('Time-Domain of imbalance rig')
xlabel('Time sec')
ylabel ('Sampled Measurement')
subplot(2,3,5)
plot (t,resonance) % plot for resonance rig meassurements
title ('Time-Domain of resonance rig')
xlabel('Time sec')
ylabel ('Sampled Measurement')
%----------------------------Frequency-Domain Plot----------------%
[P1,~] = pwelch(bearing,[],[],[],1000); %samping frequency of 1 kHz
[P2,~] = pwelch(gearmesh,[],[],[],1000);
[P3,~] = pwelch(misalignment,[],[],[],1000);
[P4,~] = pwelch(imbalance,[],[],[],1000);
[P5,f] = pwelch(resonance,[],[],[],1000);
%Since all the frequency component of pwelch are same, therefore ~ is used
%       to speed the processing time
P = [P1,P2,P3,P4,P5]; % PSD of all 5
til = ["Bearing freq","Gearmesh freq","Misalignment freq","Imbalance
freq","Resonance freq"];
% Frequency plot
i = 7;
k = 1;
while i > 6 && i <12
figure (i)
plot(f,P(:,k))
xlabel ('Frequency (Hz)')
ylabel ('Power Spectral Density Estimate')
title ({til(1,k)})
i = i +1;
k = k +1;
end
```

Appendix B.2: Feature Extraction

```
%-----------------------------Lab A.I - Task II--------------------------%
% Task II : Feature Extraction
%--------------------Reshaping Matrices---------------------------------%
reshape_bearing = reshape(bearing,1000,50);
reshape_gearmesh = reshape(gearmesh,1000,50);
reshape_imbalance = reshape(imbalance,1000,50);
reshape_misalignment = reshape(misalignment,1000,50);
reshape_resonance = reshape(resonance,1000,50);
%-------------------------Pre-allocating for speed-----------------------%
x_normalb = zeros(1000,50); x_normalg = zeros(1000,50);
x_normali = zeros(1000,50); x_normalm = zeros(1000,50);
x_normalr = zeros(1000,50);

%---------------------------Normalization-------------------------------%
for j = 1:50
    xmean_b = repmat(mean(reshape_bearing(:,j)),1000,1);
    xmean_g = repmat(mean(reshape_gearmesh(:,j)),1000,1);
    xmean_i = repmat(mean(reshape_imbalance(:,j)),1000,1);
    xmean_m = repmat(mean(reshape_misalignment(:,j)),1000,1);
    xmean_r = repmat(mean(reshape_resonance(:,j)),1000,1);

    x_normalb(:,j) = reshape_bearing(:,j) - xmean_b; %Bearing
    x_normalg(:,j) = reshape_gearmesh(:,j) - xmean_g; %GearMesh
    x_normali(:,j) = reshape_imbalance(:,j) - xmean_i; % Imbalance
    x_normalm(:,j) = reshape_misalignment(:,j) - xmean_m;% Misalignment
    x_normalr(:,j) = reshape_resonance(:,j) - xmean_r; % Resonance

end
% saved so that it can be used in assignment problem I(a) [Appendix B.4]
save normalized_bearing x_normalb
save normalized_gearmesh x_normalg
save normalized_imbalance x_normali
save normalized_misalignment x_normalm
save normalized_resonance x_normalr
%-----------------------Feature f1--------------------------------------%
for k = 1:50
    [PSD_b(:,k),f1] = pwelch(x_normalb(:,k),[],[],[],1000);
    [PSD_g(:,k),f1] = pwelch(x_normalg(:,k),[],[],[],1000);
    [PSD_i(:,k),f1] = pwelch(x_normali(:,k),[],[],[],1000);
    [PSD_m(:,k),f1] = pwelch(x_normalm(:,k),[],[],[],1000);
    [PSD_r(:,k),f1] = pwelch(x_normalr(:,k),[],[],[],1000);
end

for k1 = 1:50
    f1_b(:,k1) = (norm(PSD_b(:,k1))) / sqrt(max(size(PSD_b))); % f1 of
bearing
```

```
     f1_g(:,k1) = (norm(PSD_g(:,k1))) / sqrt(max(size(PSD_g))); % f1 of
gearmesh
     f1_i(:,k1) = (norm(PSD_i(:,k1))) / sqrt(max(size(PSD_i)));
     f1_m(:,k1) = (norm(PSD_m(:,k1))) / sqrt(max(size(PSD_m)));
     f1_r(:,k1) = (norm(PSD_r(:,k1))) / sqrt(max(size(PSD_r)));
 end
 %------------------(Butterworth) Feature f2------------------------------%

 %NOTE: filter_extract is a function that is designed to apply appropriate
 % filter and determine PSD, which is then used to extract features;
 % f2,f3,f4. The function is shown at the end of the page.

 [B,A] = butter(11,0.1);  %11th order low pass Butterworth Digital Filter
 f2_b = filter_extract(B,A,x_normalb,Fs); %feature, f2 of bearing
 f2_g = filter_extract(B,A,x_normalg,Fs); %feature, f2 of gearmesh
 f2_i = filter_extract(B,A,x_normali,Fs);
 f2_m = filter_extract(B,A,x_normalm,Fs);
 f2_r = filter_extract(B,A,x_normalr,Fs);
 %--------------------------Band pass Filter f3 (50 - 200 Hz)-------------%
 [B,A] = butter(13,[0.1 0.4]); % 13th Order

 f3_b = filter_extract(B,A,x_normalb,Fs);
 f3_g = filter_extract(B,A,x_normalg,Fs);
 f3_i = filter_extract(B,A,x_normali,Fs);
 f3_m = filter_extract(B,A,x_normalm,Fs);
 f3_r = filter_extract(B,A,x_normalr,Fs);

 %---------------------------High pass Filter f4 (200Hz)-------------------%
 [B,A] = butter(18,0.4,'high'); % 18th order

 f4_b = filter_extract(B,A,x_normalb,Fs);
 f4_g = filter_extract(B,A,x_normalg,Fs);
 f4_i = filter_extract(B,A,x_normali,Fs);
 f4_m = filter_extract(B,A,x_normalm,Fs);
 f4_r = filter_extract(B,A,x_normalr,Fs);

 function f_1 = filter_extract(B,A,xn,f)
 for i = 1:50
     y(:,i) = filter(B,A,xn(:,i));
     psd = pwelch(y(:,i),[],[],[],f);
     f_1(:,i) = norm(psd)/sqrt(max(size(psd)));
 end
 end
```

Appendix B.3: Data Visualization

```
% Task III
%------------------------Task III: Data Visualization-------------------%
%Desciption: Dimension reduction since there is four feature - Principal
    %component analysis (PCA) is used to map two dimensions obtained
%Transposing since "corrcef" only works with columns
f1_b1 = transpose(f1_b); f2_b1 = transpose(f2_b);
f3_b1 = transpose(f3_b); f4_b1 = transpose(f4_b);
f1_g1 = transpose(f1_g); f2_g1 = transpose(f2_g);
f3_g1 = transpose(f3_g); f4_g1 = transpose(f4_g);
f1_i1 = transpose(f1_i); f2_i1 = transpose(f2_i);
f3_i1 = transpose(f3_i); f4_i1 = transpose(f4_i);
f1_m1 = transpose(f1_m); f2_m1 = transpose(f2_m);
f3_m1 = transpose(f3_m); f4_m1 = transpose(f4_m);
f1_r1 = transpose(f1_r); f2_r1 = transpose(f2_r);
f3_r1 = transpose(f3_r); f4_r1 = transpose(f4_r);

%Features Matrices
f_b = [f1_b1,f2_b1,f3_b1,f4_b1]; %Bearing fault features
f_g = [f1_g1,f2_g1,f3_g1,f4_g1]; %Gearmesh fault features
f_i = [f1_i1,f2_i1,f3_i1,f4_i1]; %Imbalance fault features
f_m = [f1_m1,f2_m1,f3_m1,f4_m1]; %Misalignment fault features
f_r = [f1_r1,f2_r1,f3_r1,f4_r1]; %Resonance fault features

save bearing_features.mat f_b
save gearmesh_features.mat f_g
save imbalance_features.mat f_i
save misalignment_features.mat f_m
save resonance_features.mat f_r

% Task III: Data Visualization
load bearing_features.mat
load gearmesh_features.mat
load imbalance_features.mat
load misalignment_features.mat
load resonance_features.mat
```

```
% Principal Component Analysis PCA

G = [f_b ; f_g ; f_i ; f_m ; f_r]; %Combining fault cases

c = corrcoef(G); %Correlation coefficent matrix c of G
[v,d] = eig(c); % v - EigenVector & d - EigenValues of G
T = [v(:,end)' ; v(:,end-1)']; %Transformation matrix T from the first 2 com

z = T*G'; %Creates a 2-dimensional feature vector z
% Scatter plot of the 2-dimensional features
figure (13)
plot(z(1,1:50), z(2,1:50),'ko') ; hold on
plot(z(1,51:100), z(2,51:100),'bo'); hold on
plot(z(1,101:150), z(2,101:150),'ro'); hold on
plot(z(1,151:200), z(2,151:200),'go'); hold on
plot(z(1,201:250), z(2,201:250),'co'); hold off
xlabel ('z1'); ylabel('z2');
legend({'Fault 1','Fault 2','Fault 3','Fault 4','Fault 5'},'Location',...
    'southwest','NumColumns',2)
title(‘PCA Feature Signal (4 Energy Levels)')

Fault_1 = z(:,1:50)';
Fault_2 = z(:,51:100)';
Fault_3 = z(:,101:150)';
Fault_4 = z(:,151:200)';
Fault_5 = z(:,201:250)';

% Saving - To be used for part (b) of the assignment & also for Lab A. II
save Fault_1 Fault_1
save Fault_2 Fault_2
save Fault_3 Fault_3
save Fault_4 Fault_4
save Fault_5 Fault_5
```

Appendix C: Problem I.A – 6 Energy Levels

```
%------------------------------Problem I(Assignment)----------------------%
%------------------------Problem I (Part A)-------------------------------%
% Data
clear; clc;
load bearing.mat %Vibration MEAS of bearing-defect (Ts = 50sec,fs = 1 000)
load gearmesh.mat %Vibration MEAS of gearmesh rig (Ts = 50sec, fs = 1 000)
load misalignment.mat % MEAS of misalignment rig (Ts = 50sec, fs = 1 000)
load imbalance.mat % MEAS of imbalance rig (Ts = 50sec, fs = 1 000)
load resonance.mat % MEAS of resonance rig (Ts = 50sec, fs = 1 000)
% Normalized
load normalized_bearing
load normalized_gearmesh
load normalized_imbalance
load normalized_misalignment
load normalized_resonance

%---------------------------Part A---------------------------------------%
%Pre allocation for speed
load preallocation_assignment.mat
f = 1000;

%--------------------------Feature f1----------------------------------%
%Feature, f1 - Low-pass filter (25Hz - 25Hz / fs/2 - Wn = 0.05) / 7th order
[B,A] = butter(7,0.05,'low');  %7th order low pass Butterworth Digital
Filter
for k = 1:50 %Applying low pass 7th order filter
    y1_b(:,k) = filter(B,A,x_normalb(:,k));
    y1_g(:,k) = filter(B,A,x_normalg(:,k));
    y1_i(:,k)= filter(B,A,x_normali(:,k));
    y1_m(:,k) = filter(B,A,x_normalm(:,k));
    y1_r(:,k) = filter(B,A,x_normalr(:,k));
end
for k2 = 1:50 %Calculating Power Spectral Density (PSD) using Welchs method
    PSD_low_b(:,k2) = pwelch(y1_b(:,k2),[],[],[],1000);
    PSD_low_g(:,k2) = pwelch(y1_g(:,k2),[],[],[],1000);
    PSD_low_i(:,k2) = pwelch(y1_i(:,k2),[],[],[],1000);
    PSD_low_m(:,k2) = pwelch(y1_m(:,k2),[],[],[],1000);
    PSD_low_r(:,k2) = pwelch(y1_r(:,k2),[],[],[],1000);
end
for k3 = 1:50 % Feature, f1
f1_b(:,k3) = (norm(PSD_low_b(:,k3))) / sqrt(max(size(PSD_low_b)));
f1_g(:,k3) = (norm(PSD_low_g(:,k3))) / sqrt(max(size(PSD_low_g)));
f1_i(:,k3) = (norm(PSD_low_i(:,k3))) / sqrt(max(size(PSD_low_i)));
f1_m(:,k3) = (norm(PSD_low_m(:,k3))) / sqrt(max(size(PSD_low_m)));
f1_r(:,k3) = (norm(PSD_low_r(:,k3))) / sqrt(max(size(PSD_low_r)));
end
%---------------------------Feature f2----------------------------------%
```

```
% Feature, f2 - Band-Pass filter (25 -50) [0.05 0.1]
[B,A] = butter(6,[0.05 0.1]); %6th Order, [25Hz - 50Hz]
f2_b = filter_extract(B,A,x_normalb,f);
f2_g = filter_extract(B,A,x_normalg,f);
f2_i = filter_extract(B,A,x_normali,f);
f2_m = filter_extract(B,A,x_normalm,f);
f2_r = filter_extract(B,A,x_normalr,f);

%------------------------Feature f3--------------------------------------%
% Feature, f3 - Band-Pass filter (50 - 100Hz) [0.1 0.2] / 9th Order
[B,A] = butter(9,[0.1 0.2]); %9th Order, [25Hz - 50Hz]

f3_b = filter_extract(B,A,x_normalb,f);
f3_g = filter_extract(B,A,x_normalg,f);
f3_i = filter_extract(B,A,x_normali,f);
f3_m = filter_extract(B,A,x_normalm,f);
f3_r = filter_extract(B,A,x_normalr,f);

%-------------------------Feature, f4-------------------------------------%
[B,A] = butter(8,[0.2 0.4]); %8th Order, [100 - 200 Hz]

f4_b = filter_extract(B,A,x_normalb,f);
f4_g = filter_extract(B,A,x_normalg,f);
f4_i = filter_extract(B,A,x_normali,f);
f4_m = filter_extract(B,A,x_normalm,f);
f4_r = filter_extract(B,A,x_normalr,f);

%--------------------------Feature f5-------------------------------------%
% Feature, f5
[B,A] = butter(9,[0.4 0.7]); %9th Order, [200 - 350 Hz]
f5_b = filter_extract(B,A,x_normalb,f);
f5_g = filter_extract(B,A,x_normalg,f);
f5_i = filter_extract(B,A,x_normali,f);
f5_m = filter_extract(B,A,x_normalm,f);
f5_r = filter_extract(B,A,x_normalr,f);

%---------------------------Feature f6-----------------------------------%
% Feature, f6
[B,A] = butter(16,0.7,'high'); %16th Order, [200 - 350 Hz]
f6_b = filter_extract(B,A,x_normalb,f);
f6_g = filter_extract(B,A,x_normalg,f);
f6_i = filter_extract(B,A,x_normali,f);
f6_m = filter_extract(B,A,x_normalm,f);
f6_r = filter_extract(B,A,x_normalr,f);
```

```
%--------------PCA Analysis (Dimension Reduction--------------------------%
% Principle Component Analysis (PCA)
%Transposing since "corrcef" only works with columns
f1_b1 = transpose(f1_b); f2_b1 = transpose(f2_b);
f3_b1 = transpose(f3_b); f4_b1 = transpose(f4_b);
f5_b1 = transpose(f5_b); f6_b1 = transpose(f6_b);

f1_g1 = transpose(f1_g); f2_g1 = transpose(f2_g);
f3_g1 = transpose(f3_g); f4_g1 = transpose(f4_g);
f5_g1 = transpose(f5_g); f6_g1 = transpose(f6_g);

f1_i1 = transpose(f1_i); f2_i1 = transpose(f2_i);
f3_i1 = transpose(f3_i); f4_i1 = transpose(f4_i);
f5_i1 = transpose(f5_i); f6_i1 = transpose(f6_i);

f1_m1 = transpose(f1_m); f2_m1 = transpose(f2_m);
f3_m1 = transpose(f3_m); f4_m1 = transpose(f4_m);
f5_m1 = transpose(f5_m); f6_m1 = transpose(f6_m);

f1_r1 = transpose(f1_r); f2_r1 = transpose(f2_r);
f3_r1 = transpose(f3_r); f4_r1 = transpose(f4_r);
f5_r1 = transpose(f5_r); f6_r1 = transpose(f6_r);

%Features Matrices
f_b = [f1_b1,f2_b1,f3_b1,f4_b1,f5_b1,f6_b1]; %Bearing fault features
f_g = [f1_g1,f2_g1,f3_g1,f4_g1,f5_g1,f6_g1]; %Gearmesh fault features
f_i = [f1_i1,f2_i1,f3_i1,f4_i1,f5_i1,f6_i1]; %Imbalance fault features
f_m = [f1_m1,f2_m1,f3_m1,f4_m1,f5_m1,f6_m1]; %Misalignment fault features
f_r = [f1_r1,f2_r1,f3_r1,f4_r1,f5_r1,f6_r1]; %Resonance fault features

G = [f_b ; f_g ; f_i ; f_m ; f_r]; %Combining fault cases

c = corrcoef(G); %Correlation coefficent matrix c of G
[v,d] = eig(c); % v - EigenVector & d - EigenValues of G
T = [v(:,end)' ; v(:,end-1)']; %Transformation matrix T from the first 2 com

z = T*G'; %Creates a 2-dimensional feature vector z
% Scatter plot of the 2-dimensional features

%----------------------------Plot-------------------------------------%
figure (1)
%plot(z(1,:),z(2,:),'o')
plot(z(1,1:50), z(2,1:50),'ko') ; hold on
plot(z(1,51:100), z(2,51:100),'bo'); hold on
plot(z(1,101:150), z(2,101:150),'ro'); hold on
plot(z(1,151:200), z(2,151:200),'go'); hold on
plot(z(1,201:250), z(2,201:250),'co'); hold off
xlabel ('z1'); ylabel('z2');
legend({'Fault 1','Fault 2','Fault 3','Fault 4','Fault 5'},'Location',...
    'northeast','NumColumns',2)
title('PCA Feature Signal (6 Energy Levels)')
%------------------------Filter extract function----------------------%
% NOTE: Filter extract fuction was used to calcuted the PSD and each
%   features
function f_1 = filter_extract(B,A,xn,f)
for i = 1:50
    y(:,i) = filter(B,A,xn(:,i));
    psd = pwelch(y(:,i),[],[],[],f);
    f_1(:,i) = norm(psd)/sqrt(max(size(psd)));
end
end
```

Appendix D: MATLAB Code – Nearest Neighbor Algorithm

Appendix D.1: Distance Measure – Euclidean Distance

```
function distance = euc(a,b)
% Euclidean Distance
%   calculates the Euclidean distance between two cases with an equal
%   number of features.

% Author: Awabullah Syed
% Date: 10th April 2021

if nargin ~= 2
    error('Two input arguments required.');
    return;
end

if ~all(size(a) == size(b))
    error('Dimensions of inputs are not equalt.');
    return;
end
if min(size(a)) ~= 1
    error('Input is not a vector');
    return;
end

%Calculate the Euclidean Distance using the MATLAB's norm function
distance = norm (a-b);
end
```

Appendix D.2: 1-Nearest Neighbor Algorithm

```
%------------------------------Pattern Classification--------------------%
%------------------------------Lab A.II----------------------------------%
%1-Nearest Neighbor Algorithm
% Load Faults
load Fault_1 %Bearing Fault
load Fault_2 % Gearmesh Fault
load Fault_3 % Imbalance Fault
load Fault_4 % Misalignment Fault
load Fault_5 % Resonance Fault


%Training Data - used in construction of the classifier
%Test Data - used to evaluate the goodness or performance of the classifier
NoOfTrainingCases = 35;  %Training cases
NoOfTestingCases = length(Fault_1) - NoOfTrainingCases; % 15 vectors
% Trainging Data (First 35 data)
trainingSet = [Fault_1(1:NoOfTrainingCases,:);
    Fault_2(1:NoOfTrainingCases,:);
    Fault_3(1:NoOfTrainingCases,:);
    Fault_4(1:NoOfTrainingCases,:);
    Fault_5(1:NoOfTrainingCases,:)]; %Training Data
% Testing set by last 15 vectors
testingSet = [Fault_1(NoOfTrainingCases+1:end,:);
    Fault_2(NoOfTrainingCases+1:end,:);
    Fault_3(NoOfTrainingCases+1:end,:);
    Fault_4(NoOfTrainingCases+1:end,:);
    Fault_5(NoOfTrainingCases+1:end,:)];  %Testing Data
%----------Initial Varaiables for 1-nearest neighbour search------------%
% Label sets
trainingTarget = [ones(1,NoOfTrainingCases),...
    ones(1,NoOfTrainingCases)*2,...
    ones(1,NoOfTrainingCases)*3,...
    ones(1,NoOfTrainingCases)*4,...
    ones(1,NoOfTrainingCases)*5];

testingTarget = [ones(1,NoOfTestingCases),...
    ones(1,NoOfTestingCases)*2,...
    ones(1,NoOfTestingCases)*3,...
    ones(1,NoOfTestingCases)*4,...
    ones(1,NoOfTestingCases)*5];
%---------------------------1-nearest neighbour search-----------------%
% Total number of cases
totalNoOfTestingCases = NoOfTestingCases * 5;
totalNoOfTrainingCases = NoOfTrainingCases * 5;


inferredlabels = zeros(1,totalNoOfTestingCases);
```

```
%This loop cycles through each unlabelled item:
for unlabelledCaseIdx = 1:totalNoOfTestingCases
    unlabelledCase = testingSet(unlabelledCaseIdx, :);

    %As any distance is shorter than infinity
    shortestDistance = inf;
    shortestDistanceLabel = 0; %Assign a temporary label

    %This loop cycles through each labelled item:
    for labelledCaseIdx = 1:totalNoOfTrainingCases
        labelledCase = trainingSet(labelledCaseIdx, :);

        %Calculate the Euclidean distance:
        currentDist = euc(unlabelledCase,labelledCase);

        %Check the distance
        if currentDist < shortestDistance
            shortestDistance = currentDist;
            shortestDistanceLabel = trainingTarget(labelledCaseIdx);
        end
    end %Closes the inner for loop
    %Assign the ofund label to the vector of inferred labels:
    inferredlabels(unlabelledCaseIdx) = shortestDistanceLabel;
end %Outer For loop ends

% No. of correctly classified samples
Nc = length(find(inferredlabels == testingTarget));
% No. of all samples
Na = length(testingTarget);

%Accuracy of Classification ACC
Acc = 100*(Nc/Na); % Accuracy of classification (ACC)
disp(Acc)
% Previously used to cacluate accuracy
% Takes more execution time
% Changed with the 'Acc' code above

% Na=totalNoOfTestingCases;  % Requires more execution time, NOT included
%     Nc=0;
%     for i=1:Na
%         if(inferredlabels(i)==testingTarget(i))
%             Nc=Nc+1;
%         end
%     end
%     Acc(i) = 100*Nc/Na; disp(Acc)
%-------------------------------End of Script (1-Nearest Neighbor)--------%
```

Appendix D.3: K-Nearest Neighbor Algorithm

```
%------------------------------(Problem I (Part B)-----------------------%
%Extend the 1-nearest neighbours algorithm developed in Lab A.II to create
%k-nearest neighbours soltuions
    % K = 1, 3 and 5
    % Performed with only four FEATURES obtained during Lab A.I

% Load Faults from Lab A.I (After Applying PCA to 4 Features)
load Fault_1 %Bearing Fault
load Fault_2 % Gearmesh Fault
load Fault_3 % Imbalance Fault
load Fault_4 % Misalignment Fault
load Fault_5 % Resonance Fault
%Training Data - used in construction of the classifier
%Test Data - used to evaluate the goodness or performance of the classifier
NoOfTrainingCases = 35;
NoOfTestingCases = length(Fault_1) - NoOfTrainingCases; % 15 vectors
% Trainging Data
trainingSet = [Fault_1(1:NoOfTrainingCases,:);
    Fault_2(1:NoOfTrainingCases,:);
    Fault_3(1:NoOfTrainingCases,:);
    Fault_4(1:NoOfTrainingCases,:);
    Fault_5(1:NoOfTrainingCases,:)]; %Training Data
% Testing set by last 15 vectors
testingSet = [Fault_1(NoOfTrainingCases+1:end,:);
    Fault_2(NoOfTrainingCases+1:end,:);
    Fault_3(NoOfTrainingCases+1:end,:);
    Fault_4(NoOfTrainingCases+1:end,:);
    Fault_5(NoOfTrainingCases+1:end,:)];  %Testing Data
%----------Initial Varaiables for k-nearest neighbour search------------%
% Label sets
trainingTarget = [ones(1,NoOfTrainingCases),...
    ones(1,NoOfTrainingCases)*2,...
    ones(1,NoOfTrainingCases)*3,...
    ones(1,NoOfTrainingCases)*4,...
    ones(1,NoOfTrainingCases)*5];
testingTarget = [ones(1,NoOfTestingCases),...
    ones(1,NoOfTestingCases)*2,...
    ones(1,NoOfTestingCases)*3,...
    ones(1,NoOfTestingCases)*4,...
    ones(1,NoOfTestingCases)*5];
%---------------------------k-nearest neighbour search-----------------%
% Total number of cases
totalNoOfTestingCases = NoOfTestingCases * 5;
totalNoOfTrainingCases = NoOfTrainingCases * 5;

inferredlabels = zeros(1,totalNoOfTestingCases);
```

```
% This loop cycles through each unlabelled item:
for unlabelledCaseIdx = 1:totalNoOfTestingCases
    unlabelledCase = testingSet(unlabelledCaseIdx,:);

    % As any distance is shorter than infinity
    shortestDistance = inf;
    % Assign a temporary label
    shortestDistanceLabel = 0;

    currentDist = 0;

    % This loop cycles through each labelled item:
    for labelledCaseIdx = 1:totalNoOfTrainingCases
        labelledCase = trainingSet(labelledCaseIdx, :);
        % Calculate the Euclidean distance:
        currentDist(labelledCaseIdx) = euc(unlabelledCase, labelledCase);
    end
    % find the 3 shortest distances
    [shortestDistance,I] = mink(currentDist,3); % 3 for k =3
    % match the 3 shortest distances with the corresponding labels
    shortestDistanceLabel_temp = trainingTarget(I);

    % Most frequent label of k (k = 3 in this case) shortest distances
    % Mode used for most frequent values
    [shortestDistanceLabel,F] = mode(shortestDistanceLabel_temp);

    % if all 3 shortest distances fall into different fault types,
    % label the testing data as the type with shortest distance
    if F == 1
        shortestDistanceLabel = shortestDistanceLabel_temp(1);
    end

% Assign the found label to the vector of inferred labels:
inferredLabels(unlabelledCaseIdx) = shortestDistanceLabel;
end

% No. of correctly classified samples
Nc = length(find(inferredLabels == testingTarget));
% No. of all samples
Na = length(testingTarget);

%Accuracy of Classification ACC
Acc = 100*(Nc/Na); % Accuracy of classification (ACC)
disp(Acc)

%--------------End of Script (K-Nearest Neighbor)--------------------------%
```

Appendix D.4: Accuracy of K-Nearest Neighbor Algorithm

Part A

```
Na=totalNoOfTestingCases;
    Nc=0;
    for i=1:Na
        if(inferredlabels(i)==testingTarget(i))
            Nc=Nc+1;
        end
    end
    Acc(i) = 100*Nc/Na;
    disp(Acc)
```

Part B

```
%Accuracy of Classification ACC
Acc = 100*(Nc/Na); % Accuracy of classification (ACC)
disp(Acc)
```

Bibliography

A. Noori-Khajavi, R. K. (1995). Frequency and time domain analyses of sensor signals in drilling - I. Correlation with drill wear . *Internaltional Journal of Machine Tools and Manufacture* , Volume 35, Issue 6, June 1995, Pages 775-793.

Adnan Hamad, D. Y. (2010). Radial basis function neural network in fault detection of automotive engines . *International Journal of Engineering Science and Technology* , Vol. 2, No. 10, 2020, pp 1-8.

Ahmadi, H. (2012). Comparison of Two Classifiers; K-Nearest Neighbor and Artificial Neural Network, for Fault Diagnosis on a Main Engine Journal - Bearing . Volume 20 (Article ID 360236).

Alzaqebah, M. (2020). Neighborhood search methods with moth optimization algorithm as a wrapper method for feature selection problems . *International Journal of Electrical and Computer Engineering (IJECE)*, VOl. 10, No 4, Auguest 2020, pp. 3672~3684.

Devi, M. N. (n.d.). *Pattern Recognition* . pp 48 - 95.

Hardcastle, R. (1996). CUSUM: a credible method for the determination of authorship. *Science & Justice* , 129 - 138.

Imdad, U. (n.d.). *Classification of Students Results Using KNN and ANN* . Faisalabad : Department of Computer Science, National Textile University, Faisalabad .

Jaitley, U. (2018, October 8). *Why Data Normalization is necessary for Machine Learning models* . Retrieved from Medium : https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029

Jollifee, A. (n.d.). Principal Component Analysis. Vol. 45, Iss 3.

Mustafa, M. (n.d.). Comparison between KNN and ANN Classification in Brain Balancing Application via Spectrogram Image . *Jounal of Computer Science and Computational Mathematics* , Volume 2, Issue 4, April 2012 .

Svante Wold, K. E. (1987). Principal Component Analysis . *Chemometrics and Intelligent Laboratory Systems* , Pages 37 - 52 .