Bearing Defect Classification Project Report

Uthappa Madettira[1], Betelheim Mengesha[2], and Joseph O'Leary[3]

[1]uthu@umd.edu, 120305085
[2]bmengesh@umd.edu, 120340765
[3]joleary1@umd.edu, 115995727

9 December 2024

Contents

1 Introduction

Roller element bearings are a critical component of many mechanical systems, allowing system components to freely rotate relative to one another. Roller element bearings are also subject to mechanical wear on their internal components, potentially leading to suck bearings which cannot perform their intended function. Failed bearings may require significant system downtime to perform appropriate corrective maintenance, and the health of bearings is critical to ensuring the continuing availability of many mechanical systems [1, 17]. To preserve the availability of mechanical systems involving bearings, an automated prognostics and health management system can provide useful information to critical decision makers regarding the need for maintenance, and about what type of repair is needed given that a failure has been detected. Such systems rely upon field data collected for individual bearings, and use machine learning methods to draw conclusions about the health state of such bearings [4].

The objective of the work presented in this report is to develop a machine learning classifier that is capable of performing health assessment for SKF32208 roller bearings. As with any machine learning solution, the models developed in this report are constrained by the available data, and the type of assessment that is desired. For this work, data was provided in the form of time series of accelerometer data. The data was measured at 50 kHz in the axial direction of a bearing operating at 800 RPM on a laboratory testing rig. To obtain samples with known defects, defects were intentionally introduced to the bearings in various combinations. The defects of interest include roller failures, inner race failures, and outer race failures. These defects, the lack of any defects, and the possible combinations of the defects each compose a single class which should be distinguished using the machine learning model, since each combination of defects corresponds to a different health state and potentially different maintenance requirements. A summary of the data is shown in table 1. The provided set of 2,560 time series is split into testing and training data sets, each of which contains an equal number of samples of each of the 8 possible class labels.

Table 1: Data Summary

Label	Training Time Series	Testing Time Series
Healthy	256	64
Roller Failure	256	64
Inner Race Failure	256	64
Outer Race Failure	256	64
Inner Race + Roller Failure	256	64
Inner + Outer Race Failure	256	64
Inner + Outer Race + Roller Failure	256	64
Outer Race + Roller Failure	256	64

Given the data provided, several machine learning classifiers were trained with the objective of accurately classifying all 512 testing time series given only the training data. The classifier models which could be used for this purpose must be able to handle 8 classes, and must be trained within a reasonable computation time. The computation time is not a significant constraint on the choice of models, but will be used later in the analysis of the performance of the various methods. The fact that the model must be able to make classifications among all 8 of the provided classes is a more significant constraint on which types of models can be used.

For this work, 3 types of machine learning models are trained, one of which has two subtypes. These are: self-organizing map minimum quantization error (SOM-MQE), support vector machines (specifically a one versus one (SVM-OvO) and a separate one versus rest (SVM-OvR) architecture), and an artificial neural network, specifically a multi-layer perceptron (ANN-MLP). These models are individually trained on only the training time series (the SOM-MQE model using only the healthy data while the other methods use all the labels) and their performances are assessed and compared based on their accuracy, confusion matrices, and approximate computational complexity. The remainder of this report is organized as follows. First, further discussion of the background of bearings in rotating machinery and previous studies which have developed similar machine learning classifiers for health assessment followed by a detailed discussion of the methodologies for each of the models used in this work. This is followed by an analysis of the performance of each model and a discussion of its benefits and drawbacks, followed by a conclusion describing the model which performs best on the given data.

2 Background

Bearing health assessment and diagnosis have been active areas of research for many years, as rolling element bearings are critical components in rotary machinery systems [17]. They play a vital role in ensuring smooth and efficient operations across various industries, from turbines to automotive systems. Bearing failures can lead to significant consequences, including costly repairs, replacements, and unplanned downtime [6].

To address these challenges, various techniques have been developed to monitor health and detect potential issues early. Common methods include vibration analysis [9, 10, 7, 20], acoustic emissions [11, 15], thermal monitoring [3, 16], and oil debris analysis [5, 21]. Among these, vibration analysis stands out due to its cost-effectiveness and simplicity. It has proven to be particularly effective in diagnosing bearing health by identifying anomalies in bearing movement [12].

The foundation of fault diagnosis in rotating machinery lies in analyzing vibration variations, which are highly sensitive to minor structural changes. Any fault that occurs in the rotor significantly influences its vibration behavior [9]. However, the random nature of vibration signals from machine elements makes them difficult to model using a single mathematical formula. This challenge can be addressed by extracting meaningful statistical features in the time domain, such as peak, mean, root mean square (RMS), maximum, minimum, standard deviation, skewness, and kurtosis [18, 14, 12, 13, 19].

In addition to time-domain analysis, frequency-domain techniques are widely used for bearing fault diagnosis [19, 2, 8]. These methods involve calculating characteristic fault frequencies, which depend on bearing parameters and extracting these frequency-domain features. The envelope method, in particular, helps identify peaks at characteristic fault frequencies, serving as key indicators for assessing the health of rolling element bearings.

In this study, as mentioned in the introduction, a test bed is used to collect vibration data from a SKF 32208 tapered roller bearing. To accurately analyze SKF 32208 tapered roller bearings, it is essential to understand their specifications and parameters. Table 2 outlines the key dimensions and operational details needed to calculate fault frequencies, including pitch diameter, roller diameter, number of rollers, contact angle, and the rotational speeds of the inner and outer races. Since all bearings used in the analysis are of the same model, these parameters and specifications remain consistent across all eight health conditions.

Table 2: Bearing dimensions and parameters

Parameters	Value
Pitch Diameter (D)	60 mm
Roller Diameter (d)	10 mm
Number of Rollers (n)	17
Contact Angle (Φ)	14.94
Rotating Speed of Inner Race (fi)	800 rpm
Rotating Speed of Outer Race (fo)	0 rpm

These parameters form the basis for computing critical fault frequencies such as Ball Passing Frequency Outer race (BPFO), Ball Passing Frequency Inner race (BPFI), Ball Fault Frequency (BFF), and Fundamental Train Frequency (FTF). BPFO and BPFI represent faults in the outer and inner races, respectively, caused by the passage of roller elements. BFF corresponds to faults in the roller elements, triggered by alternating contacts with the inner and outer races. Lastly, FTF is associated with faults in the cage or mechanical looseness within the bearing. In some studies like [1, 17], the use of Ball Spin Frequency (BSF) is also shown. BSF represents the rotational frequency of each ball and is associated with roller faults, specifically BFF. The relationship between them is defined such that BFF is twice the BSF.

The equations 1 to 4 are used to calculate fault frequencies, and their computed values, are presented in Table 3.

$$BPFO = \frac{nfi}{2}\left(1 - \frac{d}{D}\cos\theta\right) \tag{1}$$

$$BPFI = \frac{nfi}{2}\left(1 + \frac{d}{D}\cos\theta\right) \tag{2}$$

$$BFF = \frac{Dfi}{d}\left[\left(1 - \frac{d}{D}\cos\theta\right)\right]^2 \tag{3}$$

$$FTF = \frac{fi}{2}\left(1 - \frac{d}{D}\cos\theta\right) \tag{4}$$

Table 3: Calculated Fault Frequencies

Fault frequencies	BPFO	BPFI	BFF	FTF
Value (HZ)	131.8	94.5	77.8	7.75

3 Methodology

The methodology followed in this work is shown in the flow chart below.

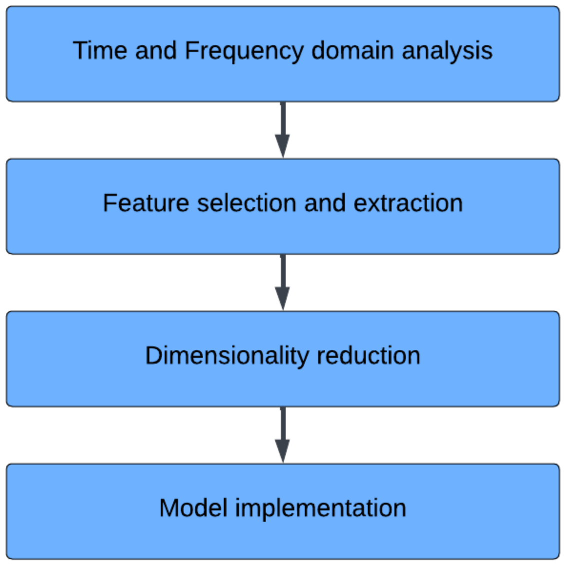

Time and Frequency domain analysis
Feature selection and extraction
Dimensionality reduction
Model implementation

Figure 1: Flowchart for the methodology followed

3.1 Time and Frequency domain analysis

The data and the data labels provided are imported and stored in numpy arrays. To visualize the patterns in the dataset and analyze the behavior of different labeled data, the acceleration is plotted against the data sample number for the first sample of each label. Eight plots are generated, each corresponding to a specific label in the dataset. These visualizations provide insight into the trends and variations in the acceleration data in different labels. The Fast Fourier Transform (FFT) was applied to the time-domain signals to analyze the frequency characteristics of the dataset. The Fourier transform is computed along the sample axis for both the training and test datasets, transforming the acceleration data into its frequency components. The frequency magnitude is plotted for the first sample of each label. The frequency axis is limited to 0–220 Hz for a better visualization of the significant frequency components. The plots are included in the appendices.

3.2 Feature selection and extraction

Both time domain and the frequency domain features are considered in this work. The frequency magnitude from 20-220 Hz, standard deviation and skewness are used as features. The frequency features were computed by applying a Fast Fourier Transform (FFT) to the signals, isolating the frequency range of interest (20 Hz to 220 Hz) and are visualized as shown in figure 2. The standard deviation is calculated to quantify the variability of the

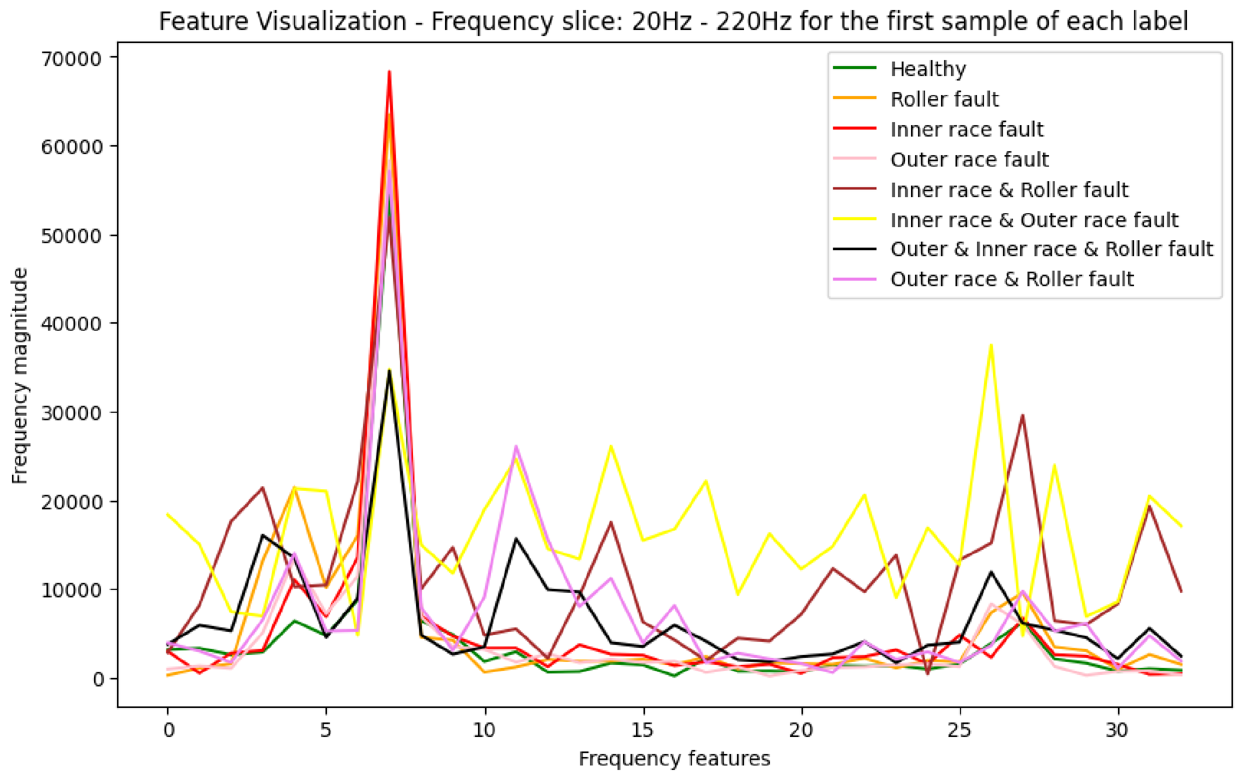


Figure 2: Feature visualization - Frequency

signal across samples, providing insights into the signal's dispersion, and is visualized as shown in Figure 3. Finally, skewness, a measure of the asymmetry of the data distribution, is used to highlight deviations from a normal distribution, which may indicate faults and is visualized as shown in figure 4. A total of 35 features are extracted, 33 frequency domain features and 2 time domain features. These features are visualized for different fault states of the data.

The calculated fault frequencies (BPFO, BFF, BPFI, FTF) were successfully extracted (see the figure in the appendix, Figures 14 to 17). However, they did not provide additional value for analysis, as they mostly duplicated features already captured in the 20–220 Hz range due to their values being below 132 Hz. Since these fault frequencies were already encompassed within that range, no additional information was gained by including them. Consequently, they were not included in the feature matrix.

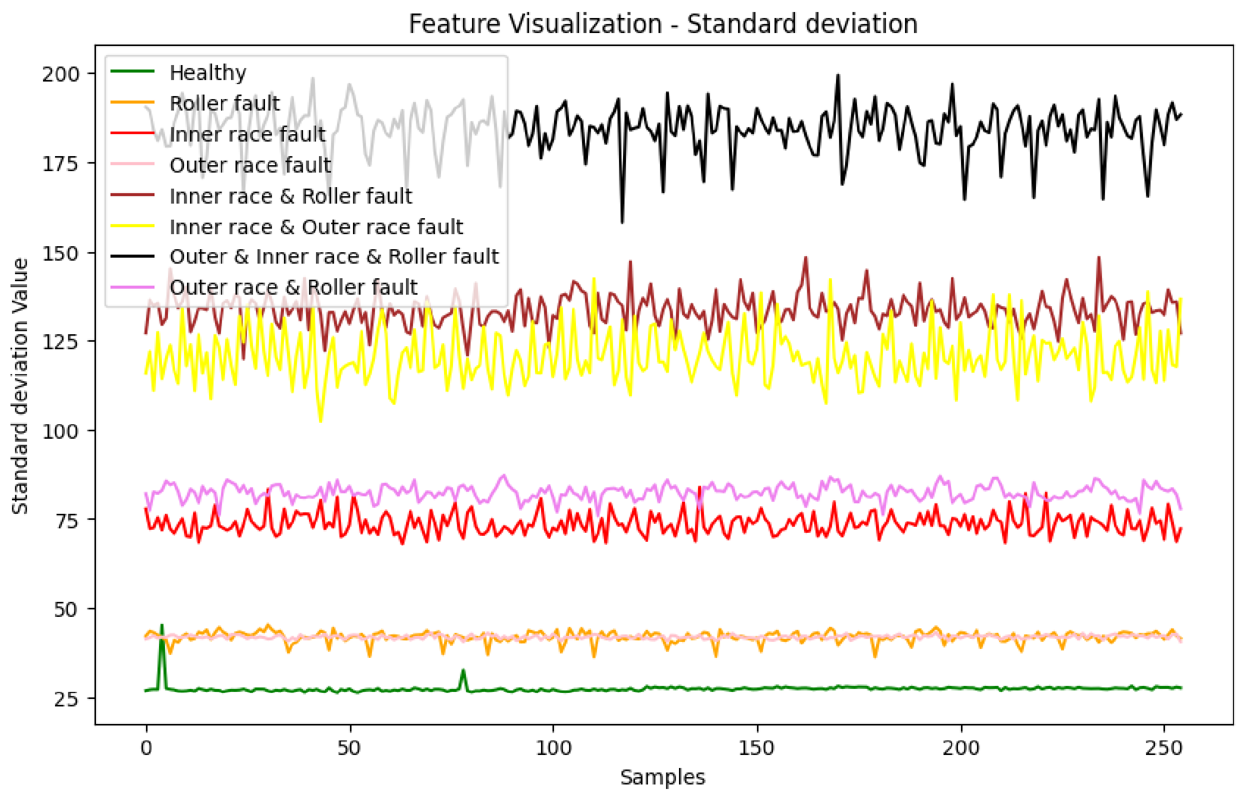


Figure 3: Feature visualization - Standard Deviation

3.3 Dimensionality Reduction

Principal Component Analysis (PCA) is applied to reduce the dimensionality of the feature space while retaining the most significant information. Initially, PCA is performed on both training and testing features to compute the explained variance ratio for each principal component. The cumulative variance is then analyzed to determine the number of components required to preserve 95% of the total variance. This threshold is chosen to balance dimensionality reduction with information retention. Based on this analysis, the training and testing features are transformed using the selected number of principal components, resulting in a reduced feature set of 26 features. This approach effectively minimizes computational complexity while maintaining the integrity of the data for subsequent analysis and classification tasks.

3.4 Model implementation

Four models - Self-Organizing Map Mean Quantization Error, Support Vector Machine - One vs Rest, Support Vector Machine - One vs One, and Artificial Neural Network are implemented in this work to diagnose and classify the bearing health.

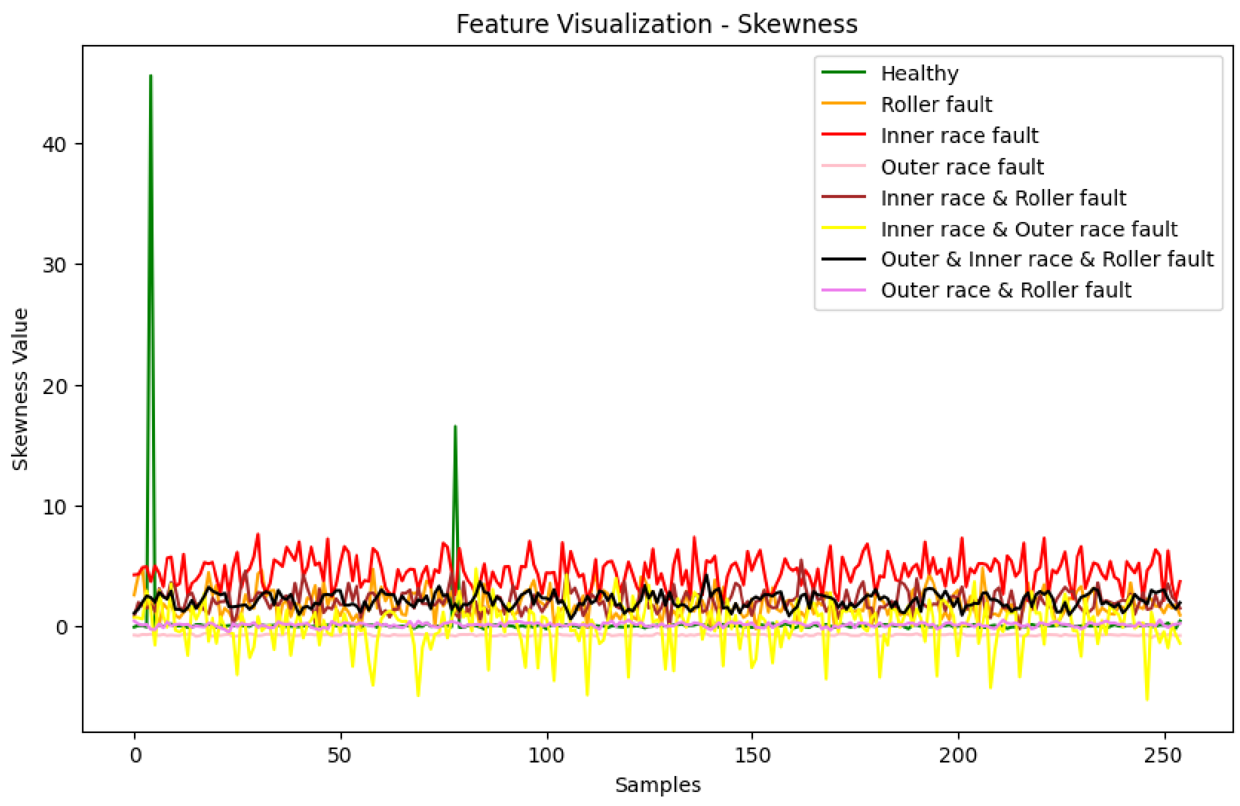


Figure 4: Feature visualization - Skewness

3.4.1 Self-Organizing Map: Mean Quantization Error

The Self-Organizing Map with Minimum Quantization Error (SOM-MQE) model is implemented to analyze the bearing degradation. Different values for the SOM grid are tried and the resulting output is analyzed. The best performance is observed for a 10x10 grid with 26-dimensional input space to map the features effectively. The model is trained on the healthy training data for 1,000 epochs, enabling it to learn the underlying patterns and relationships within the data. After training, the transformed outputs are computed for both the training and testing datasets. The mean Euclidean distances of the transformed values are calculated to assess the quantization error, which serves as an indicator of how well the data fits into the trained SOM structure.

3.4.2 Support Vector Machine: One vs Rest

A One-vs-Rest Support Vector Machine classifier with a radial basis function kernel is implemented to classify the data into multiple fault categories. The base SVM model is configured with a regularization parameter C=1 and automatic scaling of the kernel coefficient. The OvR approach trained separate binary classifiers for each label, allowing multi-class classification by predicting the most probable label. A confusion matrix is generated to analyze the classification performance. Additionally, class probabilities for both training and testing datasets are computed and visualized, illustrating the model's confidence levels across the eight fault labels.

3.4.3 Support Vector Machine: One vs One

A One-vs-One Support Vector Machine classifier with an RBF kernel is implemented to classify the data into multiple fault categories. The One-vs-One classifier trains binary classifiers for each pair of classes, making it suitable for multi-class classification by aggregating decisions from all pairwise classifiers. The model is trained on the training data, and predictions are made on the testing dataset. A confusion matrix is generated to evaluate the model's performance in distinguishing between different fault categories.

3.4.4 Artificial Neural Network: Multi-Layer Perceptron

An Artificial Neural Network model is implemented using the MLPClassifier from scikit-learn to classify the testing data into fault categories. A grid search approach is used to optimize hyperparameters such as the number of hidden layers, activation functions regularization parameter (alpha), learning rate, and solver. The best model configuration includes two hidden layers with 128 and 64 neurons, the ReLU activation function, and the Adam optimizer. A confusion matrix is used to visualize the model's classification performance.

4 Results and Discussion

4.1 Self-Organizing Map: Minimum Quantization Error

To assess the quality of the self-organizing map minimum quantization error model, a first-round visual inspection of the plot of the minimum quantization error versus sample number is performed. This shows that the model is reasonably capable of distinguishing 6 of the 8 classes from one another, but struggles to differentiate between classes 7 and 8, shown at the extreme right of figure 5, sample numbers 385-512 in the testing set. Depending on choice of bounding values between the classes, the model also shows some confusion between class 2 (samples 65-129) and classes 7 and 8, as the minimum quantization error for these classes is similar in magnitude. The model has similar performance on the training data, as shown in figure 12 in the appendices of this report.

The performance of the SOM-MQE model is rather poor for this application. Partly this is due to the complexity of the input features themselves, and partly a misapplication of SOM-MQE modelling to this problem. Generally, SOM-MQE models are good at showing the damage accumulation of a continuously degrading system, while the input bearing data in this work is a set of 8 distinct and disconnected classes. This is not to say that an SOM-MQE model cannot work, but given that the model is not able to clearly distinguish between two or three of the classes, it can reasonably be concluded that the SOM-MQE is not an optimal model for classifying bearing health using the provided data and previously discussed data pre-processing steps.

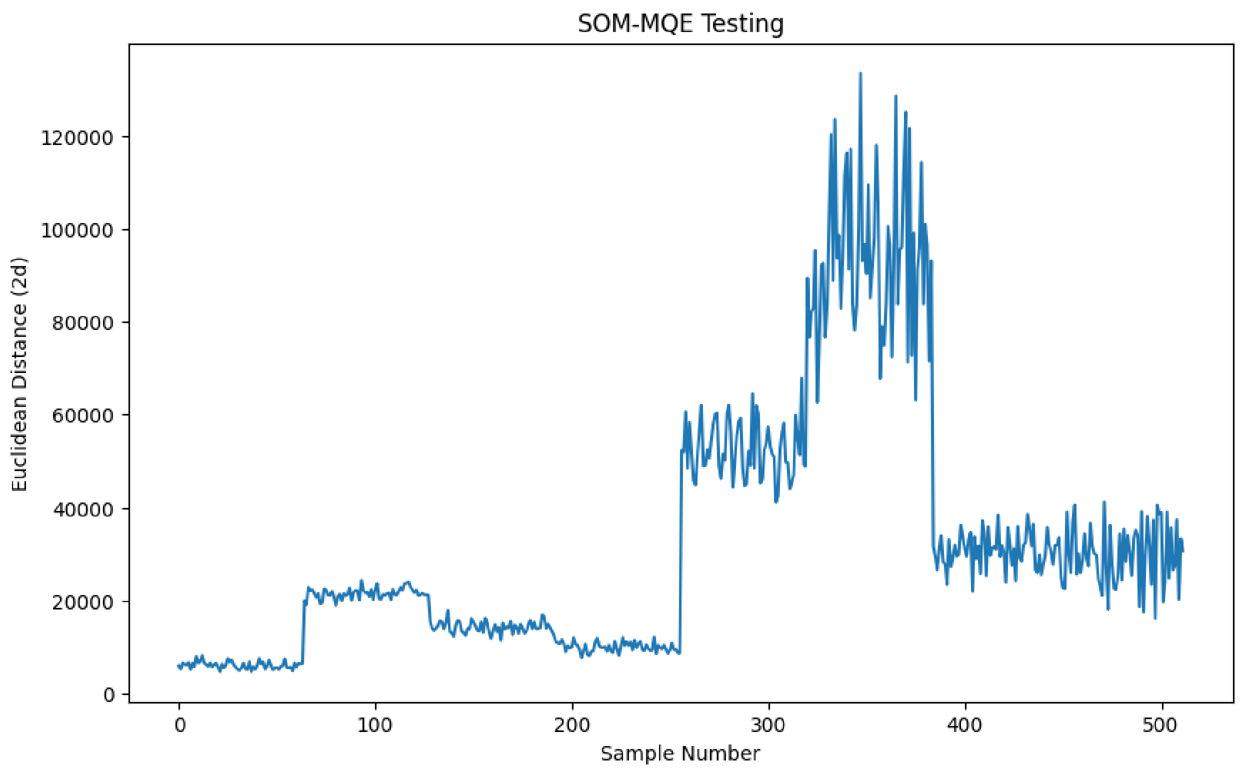
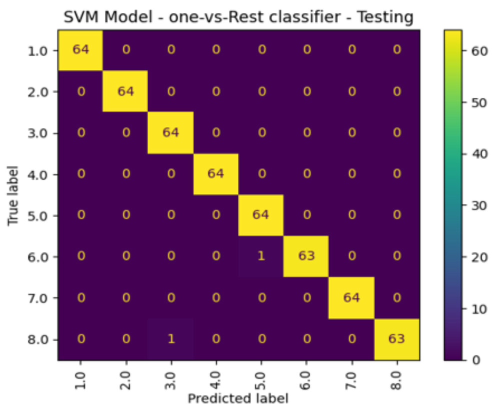


Figure 5: Testing Time Series' Minimum Quantization Error versus Sample Number

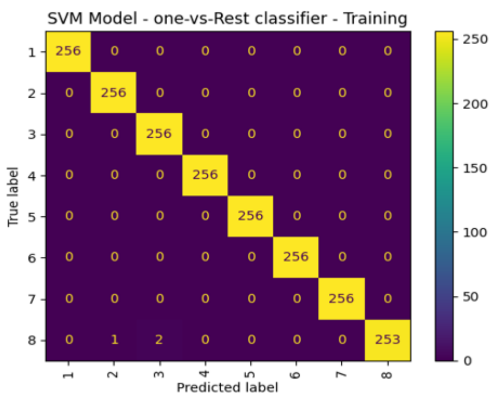
4.2 Support Vector Machine: One versus Rest

The performance of the SVM-OvR model is assessed using its confusion matrix and supported using a plot of its class probabilities as a function of the sample number. Further, an accuracy value of 99.61% for the testing data and 99.85% for the training data provide a less precise but still useful assessment of the quality of the model. The testing data confusion matrix of the SVM-OvR model is shown in figure 6.



True label \ Predicted label	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0
1.0	64	0	0	0	0	0	0	0
2.0	0	64	0	0	0	0	0	0
3.0	0	0	64	0	0	0	0	0
4.0	0	0	0	64	0	0	0	0
5.0	0	0	0	0	64	0	0	0
6.0	0	0	0	0	1	63	0	0
7.0	0	0	0	0	0	0	64	0
8.0	0	0	1	0	0	0	0	63

(a) Testing Confusion Matrix



True label \ Predicted label	1	2	3	4	5	6	7	8
1	256	0	0	0	0	0	0	0
2	0	256	0	0	0	0	0	0
3	0	0	256	0	0	0	0	0
4	0	0	0	256	0	0	0	0
5	0	0	0	0	256	0	0	0
6	0	0	0	0	0	256	0	0
7	0	0	0	0	0	0	256	0
8	0	1	2	0	0	0	0	253

(b) Training Confusion Matrix

Figure 6: One versus Rest Support Vector Machine Confusion Matrices

Primarily of interest, the results in the testing matrix show that the model is confused only by two samples, one which is class 8 (outer race + roller failure) and labeled as 3 (inner race failure) and one class 6 (inner race + outer race failure) mistakenly labeled as class 5 (inner race + roller failure). Within the training matrix, it can be seen that the model struggles with class 8 (outer race + roller failure), mislabeling a total of three points as class 2 (roller failure) or 3 (inner race failure). This confusion is between physically unrelated failures, since the model is not mislabeling outer race and roller failures as just outer race or just roller failures, for example. This indicates that the model is not confused because physically similar classes are similar in the training feature space, but that the model is confused likely due to random variation in the data. This could potentially be resolved by including additional pre-processing steps, but an accuracy above 99% is extremely good. Further refinement of the model may result in slightly better results, but due to the already high accuracy, such refinements may not be necessary and may lead to an over fit model.

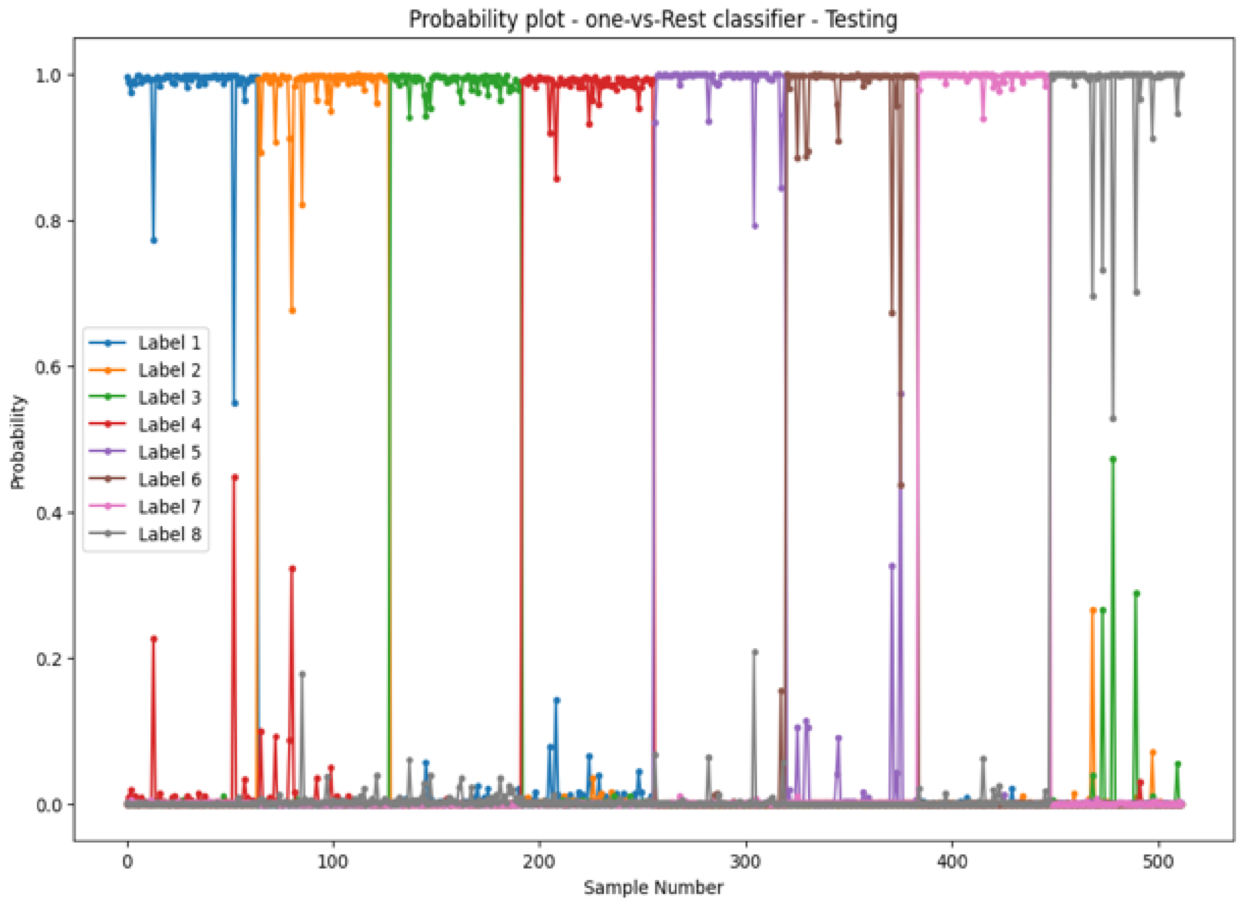
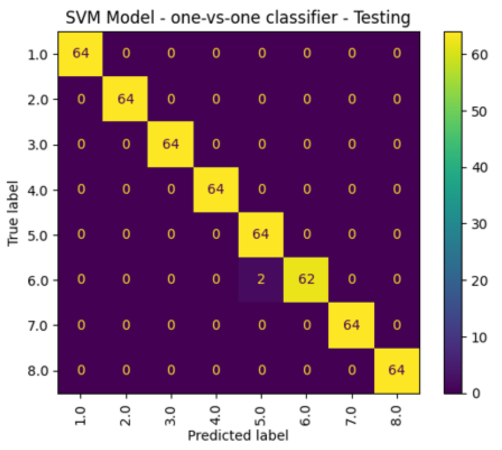


Figure 7: One versus Rest Support Vector Machine Testing Class Probabilities

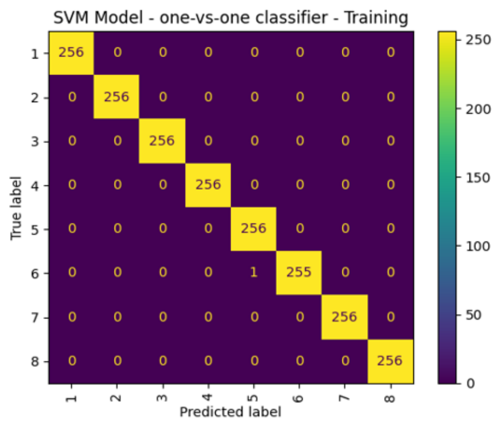
As can be seen in figure 7 for the testing data, the classification probabilities are fairly well differentiated, with not many results near the center of the probability space (probabilities closer to $\frac{1}{2}$ than 1 or 0). This supports the idea that the confusion in the model is likely due to random variability in the input data rather than an under fitting of the features. In the case of under fit, it would be expected that many of the class probabilities would be near to $\frac{1}{2}$, but given that this is not the case, it is more likely that the confusion is the result of random variability which could be improved through better pre-processing. Class probabilities for the training data are shown in a supplemental figure, figure 13 in the appendices.

4.3 Support Vector Machine: One versus One

The SVM-OvO model, in contrast to the SVM-OvR, does not make predictions based on a set of classifiers for each class individually. As a result there is no class probability prediciton using such a model. Despite this, the SVM-OvR model can still be compared to the other models using its accuracy of 99.61% on the testing data (99.95% on the training data), and using its confusion matrices. The confusion matrices are shown in figure 8.



(a) Testing Confusion Matrix



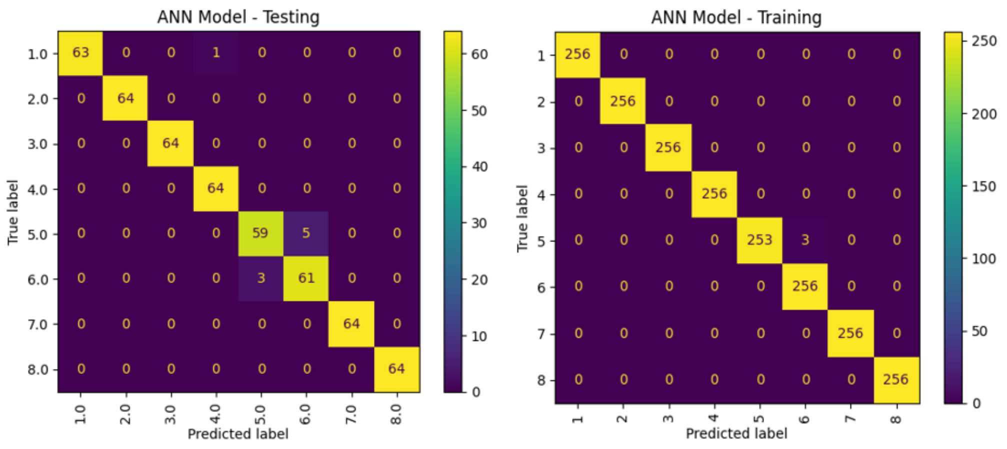
(b) Training Confusion Matrix

Figure 8: One versus Rest Support Vector Machine Confusion Matrices

Based on the confusion matrix for the testing data, it is clear that the SVM-OvO model does a good job at distinguishing all 8 classes of interest, failing to correctly classify only two input points, mistaking class 5 for class 6. This confusion corresponds to misidentifying inner race and roller failures as inner race and outer race failure. This confusion has physical similarity - both the correct and the incorrect class include failure on the inner race. Inspecting the confusion matrix for the training data, only one point is incorrectly identified, and it corresponds to the same confusion as is present in the testing data. This may indicate a slight over fit in the model, since the training accuracy is better than the testing accuracy, but the fact that the inaccuracies are occuring in the same manner for both training and testing supports the idea that the model is failing due to variability in the data, not due to over fit. Overall the SVM-OvO model does an excellent job at identifying the health state of bearings based on the input data provided.

4.4 Artificial Neural Network: Multi-Layer Perceptron

Artificial neural networks are generally known for their flexibility, being reasonably well adapted to a wide variety of application domains without significant required domain expertise for their construction. For bearing health assessment, using the same input data as the other models presented in this report, an ANN-MLP was trained and able to achieve 98.24% accuracy on the testing data (99.85% on the training data). This is lower than the support vector machine models but still better than the self organizing map. The confusion matrices for the ANN-MLP model are presented in figure 9.



(a) Testing Confusion Matrix

(b) Training Confusion Matrix

Figure 9: One versus Rest Support Vector Machine Confusion Matrices

Inspecting the testing confusion matrix, it is clear that the ANN-MLP is confused between classes 5 and 6, similar to the SVM-OvO model, although the ANN-MLP has misclassified a larger number of data points. When comparing the training and testing confusion matrices, there is good evidence of a small amount of overfit. Specifically, since the training confusion matrix (with an overall 99.85% accuracy) only has confusion in which class 5 is labeled as class 6, while the testing confusion matrix shows more confusion and confusion in which class 6 is labeled class 5 in addition to the same confusion type as in the training data, there is a reasonable chance that the model is slightly over fit. The accuracy difference between the training and testing results is not so large as to be certainly an indication of over fit, but it is a much bigger difference than that between training and testing for the support vector machines.

The primary advantage of the ANN-MLP model is that it does not require as much domain knowledge, since the model itself is more complex and capable of learning much more complex patterns in the input data. In this work, however, since all of the models were trained on the exact same input data, which was processed using the same amount of domain knowledge, the ANN-MLP model does not possess this advantage. In the results presented here, it is clear that while the ANN-MLP is a capable model, there are better options based on the analysis discussed previously.

5 Conclusion

In summary, the support vector machines and the artificial neural network classifiers are all capable of making good predictions about the health state of a bearing. The SOM-MQE model performs poorly, especially on classes 7 and 8. Based on the predictive power on the testing data, the SVM-OvO and SVM-OvR models are equivalently good, and capable of making reasonable predictions. The predictive accuracy of the models is summarized in table 4.

Table 4: Summary of Model Accuracies

Model	Testing Accuracy
SOM-MQE	$\approx$ 85%
SVM-OvR	99.61%
SVM-OvO	99.61%
ANN-MLP	98.24%

There is also the matter of computational cost to consider. Specifically, the SVM-OvO model trains a classifier for each pair of classes, while the SMV-OvR model trains one for each class. In making health assessments between 8 classes, this means that the SVM-OvO requires 28 support vectors, while the SVM-OvR requires only 8. This reduces the computational cost of the SVM-OvR relative to the SVM-OvO significantly. Both the SOM-MQE and the ANN-MLP models are neural network based, requiring a significantly larger number of computations simply by nature of their rather large number of individual nodes. Based on the better accuracy acheived by using the SVM-OvR versus the ANN-MLP and SOM-MQE, and the fact that the SVM-OvR has a lower computational cost than the SVM-OvO model, the SVM-OvR model is the recommended classifier for this application.

6 Contribution Statement

All members of the group agree that we have each contributed equally to the completion of this project.

References

[1] Andrew J Bayba et al. *Health Assessment and Fault Classification of Roller Element Bearings.* Army Research Laboratory, 2012.

[2] N Harish Chandra and AS Sekhar. "Fault detection in rotor bearing systems using time frequency techniques". In: *Mechanical Systems and Signal Processing* 72 (2016), pp. 105–133.

[3] Anurag Choudhary, Tauheed Mian, and Shahab Fatima. "Convolutional neural network based bearing fault diagnosis of rotating machine using thermal images". In: *Measurement* 176 (2021), p. 109196.

[4] Paula J Dempsey, Gary Kreider, and Thomas Fichter. *Tapered roller bearing damage detection using decision fusion analysis.* Tech. rep. NASA Glenn Research Center, 2006.

[5] IM Flanagan, JR Jordan, and HW Whittington. "Wear-debris detection and analysis techniques for lubricant-based condition monitoring". In: *Journal of Physics E: Scientific Instruments* 21.11 (1988), p. 1011.

[6] Aiwina Heng et al. "Rotating machinery prognostics: State of the art, challenges and opportunities". In: *Mechanical systems and signal processing* 23.3 (2009), pp. 724–739.

[7] Alaa Abdulhady Jaber. "Diagnosis of bearing faults using temporal vibration signals: a comparative study of machine learning models with feature selection techniques". In: *Journal of Failure Analysis and Prevention* 24.2 (2024), pp. 752–768.

[8] Saja M Jawad and Alaa A Jaber. "Bearings health monitoring based on frequency-domain vibration signals analysis". In: *Engineering and Technology Journal* 41.1 (2023), pp. 86–95.

[9] Jay Prakash Kumar, Premanand S Chauhan, and Prem Prakash Pandit. "Time domain vibration analysis techniques for condition monitoring of rolling element bearing: A review". In: *Materials Today: Proceedings* 62 (2022), pp. 6336–6340.

[10] Iulian Lupea and Mihaiela Lupea. "Machine learning techniques for multi-fault analysis and detection on a rotating test rig using vibration signal". In: *Symmetry* 15.1 (2022), p. 86.

[11] David Mba. "Acoustic emissions and monitoring bearing health". In: *Tribology Transactions* 46.3 (2003), pp. 447–451.

[12] BR Nayana and Paul Geethanjali. "Analysis of statistical time-domain features effectiveness in identification of bearing faults from vibration signal". In: *IEEE Sensors Journal* 17.17 (2017), pp. 5618–5625.

[13] Miguel Delgado Prieto et al. "Bearing fault detection by a novel condition-monitoring scheme based on statistical-time features and neural networks". In: *IEEE Transactions on Industrial Electronics* 60.8 (2012), pp. 3398–3407.

[14] Thomas W Rauber, Francisco de Assis Boldt, and Flavio Miguel Varejao. "Heterogeneous feature models and feature selection applied to bearing fault diagnosis". In: *IEEE Transactions on Industrial Electronics* 62.1 (2014), pp. 637–646.

[15] Bart Scheeren, Miroslaw Lech Kaminski, and Lotfollah Pahlavan. “Acoustic emission monitoring of naturally developed damage in large-scale low-speed roller bearings”. In: *Structural Health Monitoring* 23.1 (2024), pp. 360–382.

[16] Haidong Shao et al. “Intelligent fault diagnosis of rotor-bearing system under varying working conditions with modified transfer convolutional neural network and thermal images”. In: *IEEE Transactions on Industrial Informatics* 17.5 (2020), pp. 3488–3496.

[17] David Siegel et al. “Novel method for rolling element bearing health assessment—A tachometer-less synchronously averaged envelope feature extraction technique”. In: *Mechanical Systems and Signal Processing* 29 (2012), pp. 362–376.

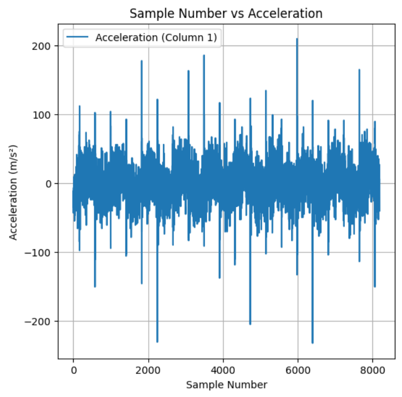
[18] V Sugumaran and KI Ramachandran. “Effect of number of features on classification of roller bearing faults using SVM and PSVM”. In: *Expert Systems with Applications* 38.4 (2011), pp. 4088–4096.

[19] Muhammad Masood Tahir et al. “Enhancing fault classification accuracy of ball bearing using central tendency based time domain features”. In: *IEEE Access* 5 (2016), pp. 72–83.

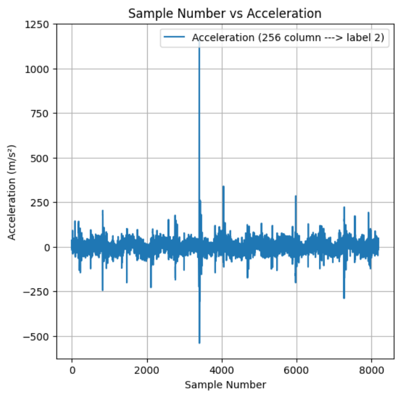
[20] Bayu Adhi Tama et al. “Recent advances in the application of deep learning for fault diagnosis of rotating machinery using vibration signals”. In: *Artificial Intelligence Review* 56.5 (2023), pp. 4667–4709.

[21] Xiaoliang Zhu, Chong Zhong, and Jiang Zhe. “Lubricating oil conditioning sensors for online machine health monitoring–A review”. In: *Tribology International* 109 (2017), pp. 473–484.

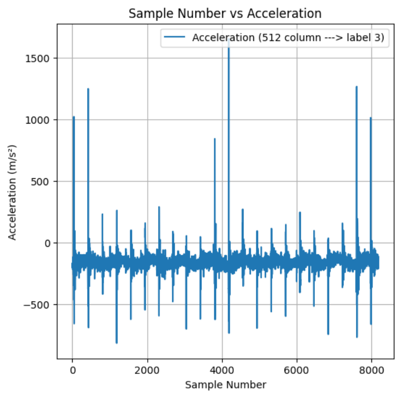
7 Appendix: Supplemental Figures



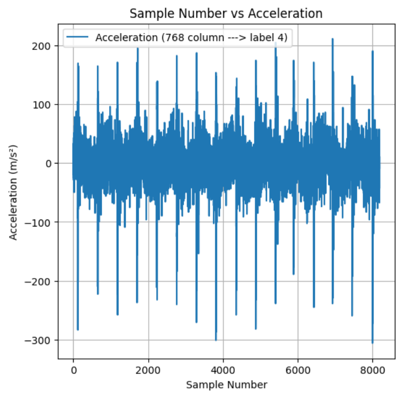
(a) Label 1



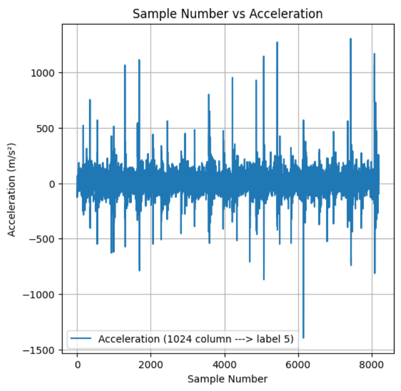
(b) Label 2



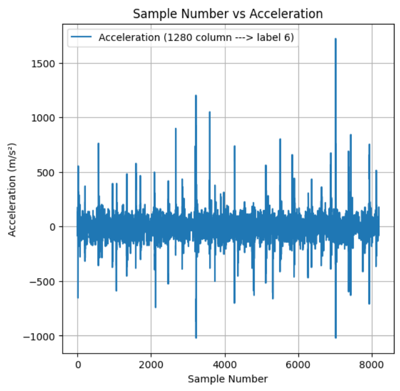
(c) Label 3



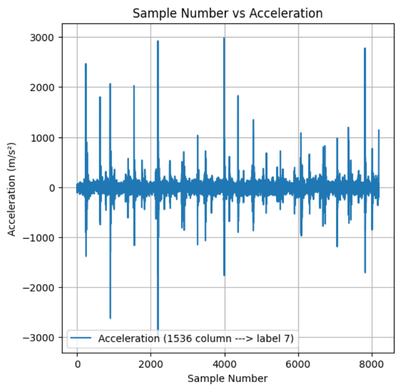
(d) Label 4



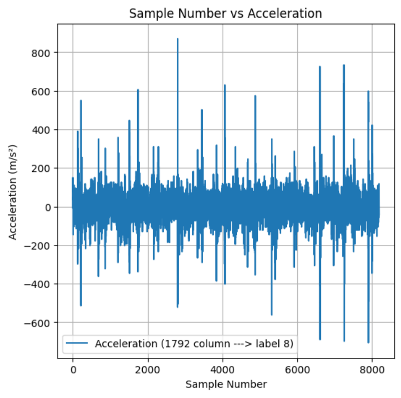
(e) Label 5



(f) Label 6

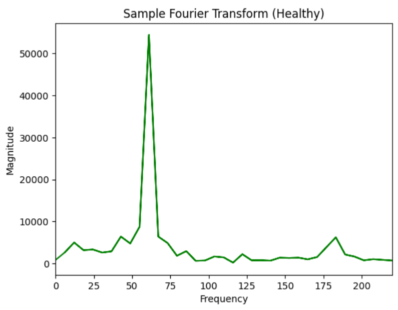


(g) Label 7

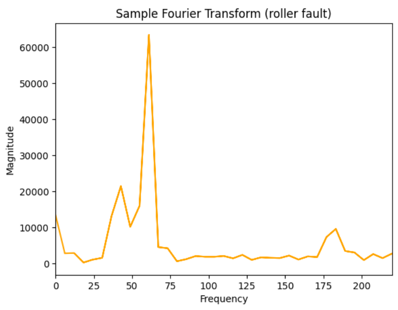


(h) Label 8

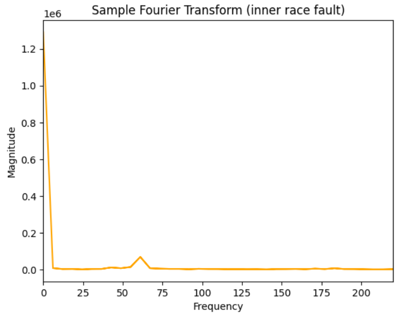
Figure 10: Time domain analysis plots



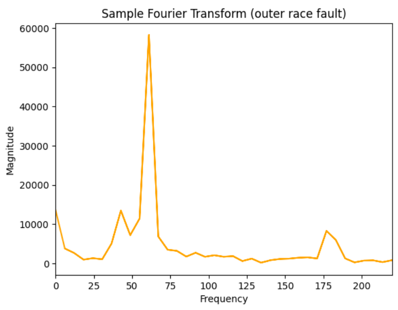
(a) Label 1



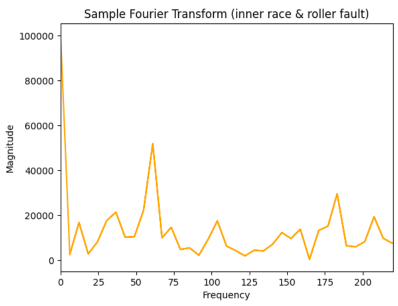
(b) Label 2



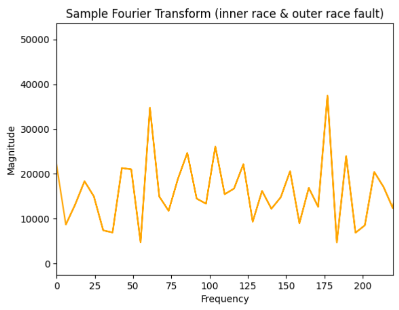
(c) Label 3



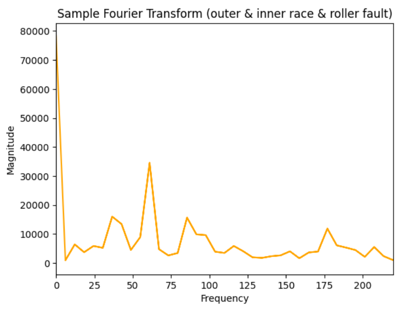
(d) Label 4



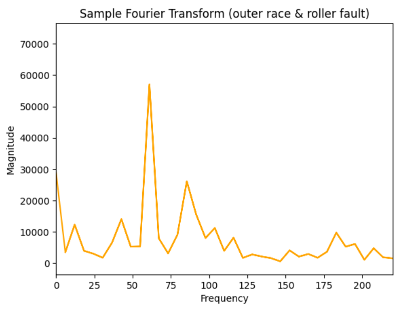
(e) Label 5



(f) Label 6



(g) Label 7



(h) Label 8

Figure 11: Frequency domain analysis plots

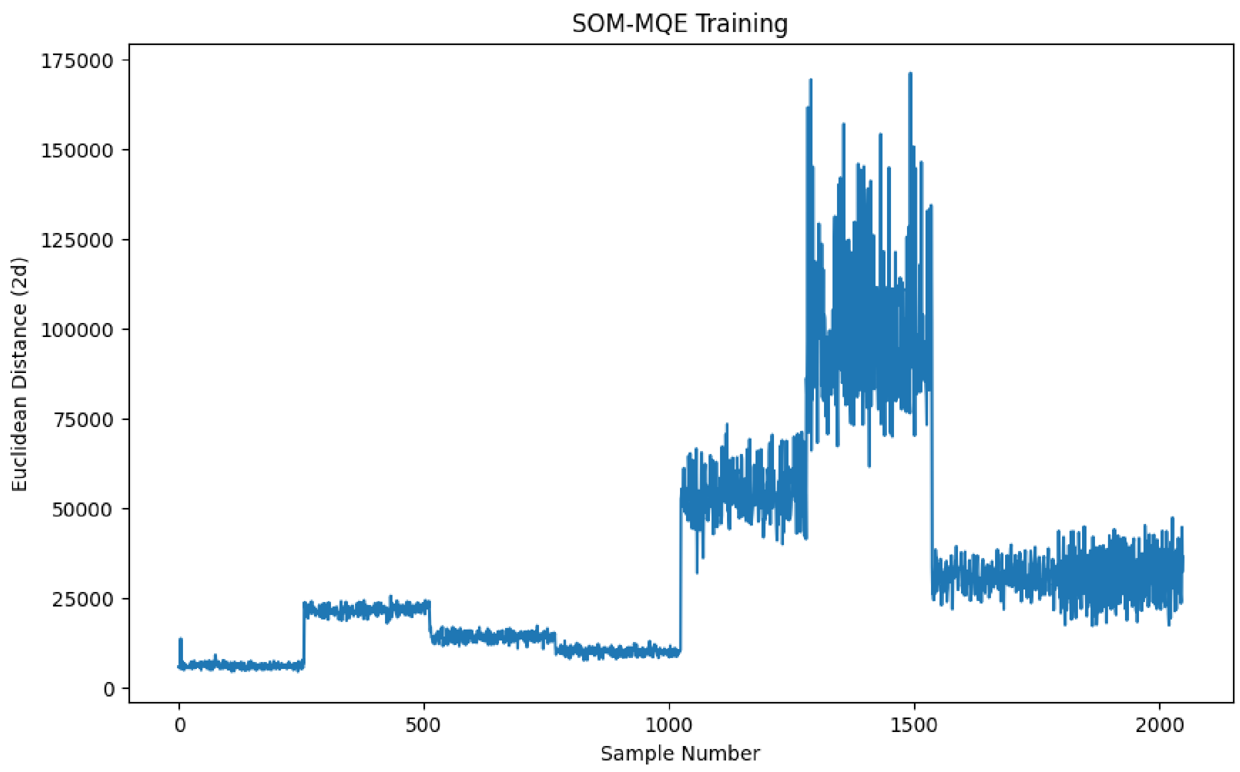


Figure 12: Training Time Series Minimum Quantizatioon Error versus Sample Number

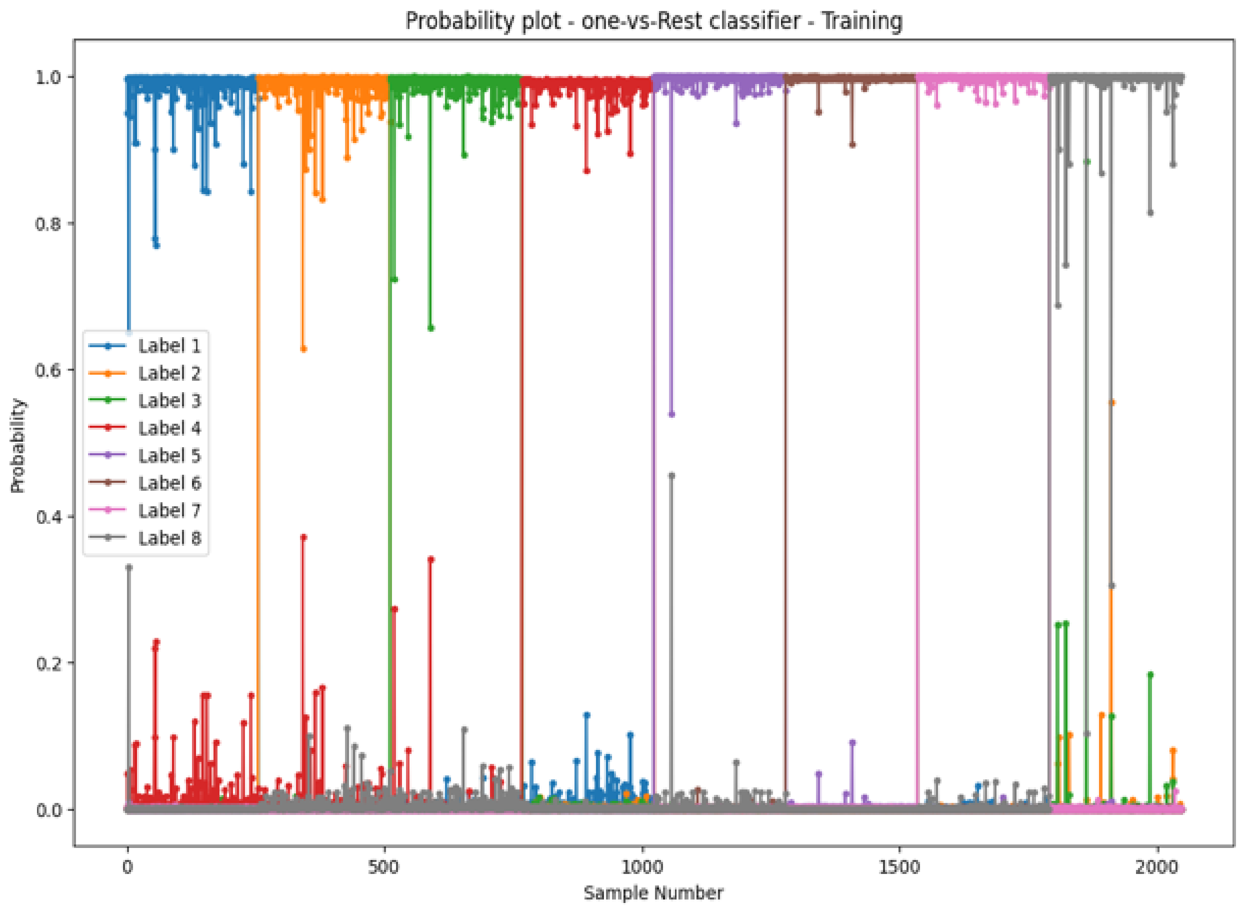


Figure 13: One versus Rest Support Vector Machine Training Class Probabilities

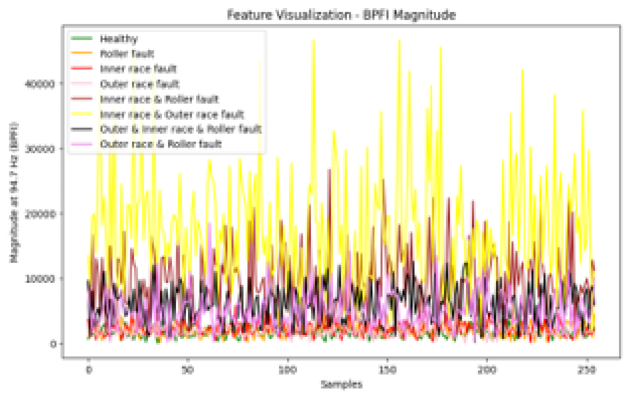


Figure 14: Ball Passing Frequency Inner Race

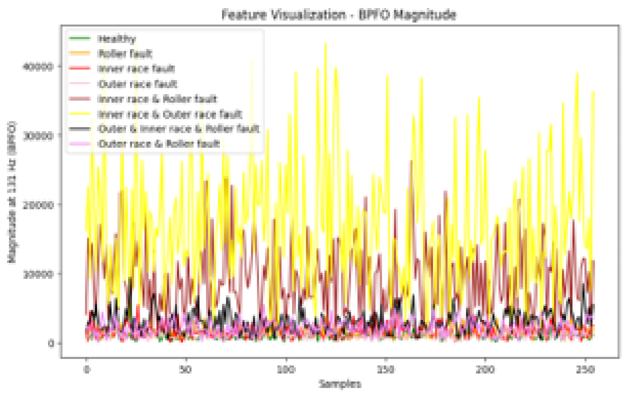


Figure 15: Ball Passing Frequency Outer Race

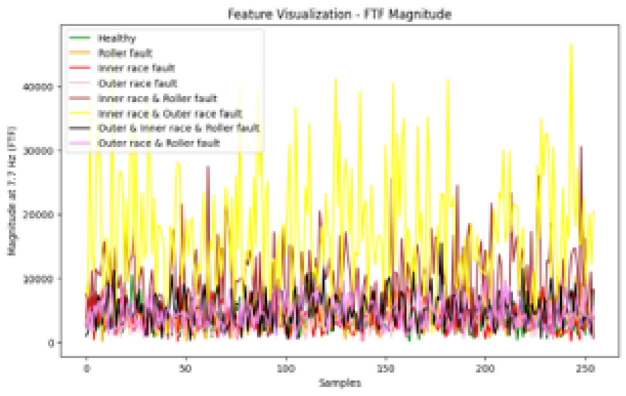


Figure 16: Fundamental Train Frequency

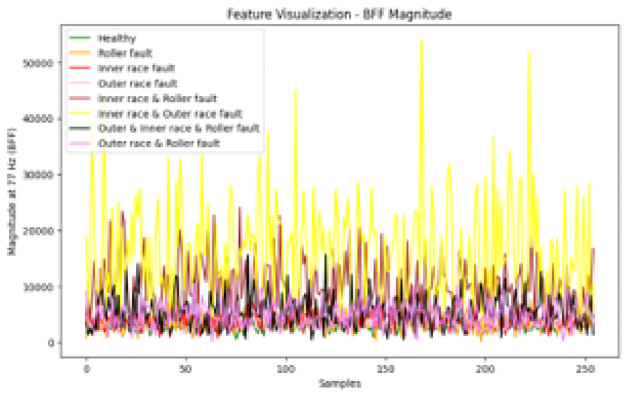


Figure 17: Ball Fault Frequency

8 Appendix: Python Source Code

```
# -*- coding: utf-8 -*-
"""final_project.ipynb

Automatically generated by Colab.

Original file is located at
   https://colab.research.google.com/drive/1wA5qOSV4oN3gtyEiXzrpqd65s16V9Ef_
"""

# Import the necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import scipy as scp
from scipy.io import loadmat
from scipy.stats import skew, kurtosis
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
!pip install sklearn_som
from sklearn_som.som import SOM
from sklearn.preprocessing import StandardScaler

# Mount the drive
from google.colab import drive
drive.mount('/content/drive/', force_remount=True)

# Commented out IPython magic to ensure Python compatibility.
# Set the path to directory
path_to_folder = "ENME691/final_project/Project 3/Dataset for Team"
# %cd /content/drive/My\ Drive/{path_to_folder}

"""Data preprocessing - Loading data from .mat and writing them into a .csv file

Load data from .csv file and store them in numpy array
"""

# Training data
file_path = '/content/drive/MyDrive/ENME691/final_project/Project 3/Dataset for
   Team/train_data.csv'
data_train_csv = pd.read_csv(file_path)
data_train_final = data_train_csv.to_numpy()
print("Shape of training data ",data_train_final.shape)

# Testing data
file_path = '/content/drive/MyDrive/ENME691/final_project/Project 3/Dataset for
   Team/data_test_final.csv'
```

```
data_test_csv = pd.read_csv(file_path, header=None)
data_test_final = data_test_csv.to_numpy()
print("Shape of testing data ", data_test_final.shape)

# Training data labels
file_path = '/content/drive/MyDrive/ENME691/final_project/Project 3/Dataset for
   Team/train_data_labels.csv'
data_trainlbs_csv = pd.read_csv(file_path)
data_trainlbs_final = data_trainlbs_csv.to_numpy()
print("Shape of training data labels ",data_trainlbs_final.shape)

"""Time Domain plots"""

# plotting for label 1
# Use sample indices as the time axis
time = np.arange(data_train_final.shape[0])

# Select a single test column to plot (e.g., the first column---> label 1)
acceleration = data_train_final[:, 0] # First column, adjust if needed
# Plotting
plt.figure(figsize=(10, 6))
plt.plot(time, acceleration, label='Acceleration (Column 1)')
plt.xlabel('Sample Number')
plt.ylabel('Acceleration (m/s)')
plt.title('Sample Number vs Acceleration')
plt.legend()
plt.grid(True)

# Plotting for label 2
# Select a single test column to plot (e.g., 256 column ---> label 2)
acceleration = data_train_final[:, 256] # First column, adjust if needed
# Plotting
plt.figure(figsize=(10, 6))
plt.plot(time, acceleration, label='Acceleration (256 column ---> label 2)')
plt.xlabel('Sample Number')
plt.ylabel('Acceleration (m/s)')
plt.title('Sample Number vs Acceleration')
plt.legend()
plt.grid(True)
plt.show()

# Plotting for label 3
# Select a single test column to plot (e.g., 512 column ---> lebel 3)
acceleration = data_train_final[:, 512] # First column, adjust if needed
# Plotting
plt.figure(figsize=(10, 6))
plt.plot(time, acceleration, label='Acceleration (512 column ---> label 3)')
plt.xlabel('Sample Number')
plt.ylabel('Acceleration (m/s)')
plt.title('Sample Number vs Acceleration')
plt.legend()
plt.grid(True)
plt.show()

# Plotting for label 4
# Select a single test column to plot (e.g., 768 column ---> lebel 4)
acceleration = data_train_final[:, 768] # First column, adjust if needed
# Plotting
plt.figure(figsize=(10, 6))
plt.plot(time, acceleration, label='Acceleration (768 column ---> label 4)')
plt.xlabel('Sample Number')
plt.ylabel('Acceleration (m/s)')
plt.title('Sample Number vs Acceleration')
plt.legend()
plt.grid(True)
plt.show()

# Plotting for label 5
# Select a single test column to plot (e.g., 1024 column ---> label 5)
acceleration = data_train_final[:, 1024] # First column, adjust if needed
# Plotting
plt.figure(figsize=(10, 6))
plt.plot(time, acceleration, label='Acceleration (1024 column ---> label 5)')
plt.xlabel('Sample Number')
plt.ylabel('Acceleration (m/s)')
plt.title('Sample Number vs Acceleration')
plt.legend()
plt.grid(True)
plt.show()

# Plotting for label 6
# Select a single test column to plot (e.g., 1280 column ---> label 6)
acceleration = data_train_final[:, 1280] # First column, adjust if needed
# Plotting
plt.figure(figsize=(10, 6))
plt.plot(time, acceleration, label='Acceleration (1280 column ---> label 6)')
plt.xlabel('Sample Number')
plt.ylabel('Acceleration (m/s)')
plt.title('Sample Number vs Acceleration')
plt.legend()
plt.grid(True)
plt.show()

# Plotting for label 7
# Select a single test column to plot (e.g., 1536 column ---> lebel 7)
acceleration = data_train_final[:, 1563] # First column, adjust if needed
# Plotting
plt.figure(figsize=(10, 6))
plt.plot(time, acceleration, label='Acceleration (1536 column ---> label 7)')
plt.xlabel('Sample Number')
plt.ylabel('Acceleration (m/s)')
plt.title('Sample Number vs Acceleration')
plt.legend()
plt.grid(True)
plt.show()

# Plotting for label 8
# Select a single test column to plot (e.g., 1792 column ---> lebel 8)
acceleration = data_train_final[:, 1792] # First column, adjust if needed
# Plotting
plt.figure(figsize=(10, 6))
plt.plot(time, acceleration, label='Acceleration (1792 column ---> label 8)')
plt.xlabel('Sample Number')
plt.ylabel('Acceleration (m/s)')
plt.title('Sample Number vs Acceleration')
plt.legend()
plt.grid(True)
plt.show()

"""Fourier transformation and plots"""

sampling_frequency = 50000
data_train_frequency = scp.fft.fft(data_train_final, axis=0)
data_test_frequency = scp.fft.fft(data_test_final, axis=0)
frequency_axis = scp.fft.fftfreq(len(data_train_final), 1/sampling_frequency)
plt.plot(np.abs(frequency_axis), np.abs(data_train_frequency[:,0]), color='green
    ')
plt.title('Sample Fourier Transform (Healthy)')
plt.xlim([0,220])
plt.xlabel('Frequency')
plt.ylabel('Magnitude')
plt.show()
plt.plot(np.abs(frequency_axis), np.abs(data_train_frequency[:,256]), color='
    orange')
plt.title('Sample Fourier Transform (roller fault)')
plt.xlim([0,220])
plt.xlabel('Frequency')
plt.ylabel('Magnitude')
plt.show()
plt.plot(np.abs(frequency_axis), np.abs(data_train_frequency[:,512]), color='
    orange')
```

```
plt.title('Sample Fourier Transform (inner race fault)')
plt.xlim([0,220])
plt.xlabel('Frequency')
plt.ylabel('Magnitude')
plt.show()
plt.plot(np.abs(frequency_axis), np.abs(data_train_frequency[:,768]), color='
    orange')
plt.title('Sample Fourier Transform (outer race fault)')
plt.xlim([0,220])
plt.xlabel('Frequency')
plt.ylabel('Magnitude')
plt.show()
plt.plot(np.abs(frequency_axis), np.abs(data_train_frequency[:,1024]), color='
    orange')
plt.title('Sample Fourier Transform (inner race & roller fault)')
plt.xlim([0,220])
plt.xlabel('Frequency')
plt.ylabel('Magnitude')
plt.show()
plt.plot(np.abs(frequency_axis), np.abs(data_train_frequency[:,1280]), color='
    orange')
plt.title('Sample Fourier Transform (inner race & outer race fault)')
plt.xlim([0,220])
plt.xlabel('Frequency')
plt.ylabel('Magnitude')
plt.show()
plt.plot(np.abs(frequency_axis), np.abs(data_train_frequency[:,1536]), color='
    orange')
plt.title('Sample Fourier Transform (outer & inner race & roller fault)')
plt.xlim([0,220])
plt.xlabel('Frequency')
plt.ylabel('Magnitude')
plt.show()
plt.plot(np.abs(frequency_axis), np.abs(data_train_frequency[:,1792]), color='
    orange')
plt.title('Sample Fourier Transform (outer race & roller fault)')
plt.xlim([0,220])
plt.xlabel('Frequency')
plt.ylabel('Magnitude')
plt.show()

"""Feature selection and extraction"""

# Frequency

min_freq = np.argmin(np.abs(frequency_axis - 20))
max_freq = np.argmin(np.abs(frequency_axis - 220))
```

```
data_train_freq = np.abs(data_train_frequency[min_freq:max_freq,:])
data_test_freq = np.abs(data_test_frequency[min_freq:max_freq,:])
print(data_train_freq.shape)
print(data_test_freq.shape)

plt.figure(figsize=(10, 6))
labels = ['Healthy', 'Roller fault', 'Inner race fault', 'Outer race fault', '
   Inner race & Roller fault', 'Inner race & Outer race fault', 'Outer & Inner
   race & Roller fault', 'Outer race & Roller fault']
colors = ['green', 'orange', 'red', 'pink', 'brown', 'yellow', 'black', 'violet']
for i in range(0,8,1):
  plt.plot(np.arange(33), data_train_freq[:,i*256], label=labels[i], color=colors
     [i])
plt.title("Feature Visualization - Frequency slice: 20Hz - 220Hz for the first
   sample of each label")
plt.xlabel("Frequency features")
plt.ylabel("Frequency magnitude")
plt.legend()
plt.show()

# Mean

# Calculate the mean for the training set
data_train_mean = np.mean(data_train_final, axis=0).reshape(1, -1)
mean_plot = data_train_mean.flatten()

# Calculate the skewness for the testing set
data_test_mean = np.mean(data_test_final, axis=0).reshape(1, -1)

plt.figure(figsize=(10, 6))
labels = ['Healthy', 'Roller fault', 'Inner race fault', 'Outer race fault', '
   Inner race & Roller fault', 'Inner race & Outer race fault', 'Outer & Inner
   race & Roller fault', 'Outer race & Roller fault']
colors = ['green', 'orange', 'red', 'pink', 'brown', 'yellow', 'black', 'violet']
for i in range(0,8,1):
  plt.plot(np.arange(255), mean_plot[i*256:256*(i+1)-1], label=labels[i], color=
     colors[i])
plt.title("Feature Visualization - mean")
plt.xlabel("Samples")
plt.ylabel("Mean Value")
plt.legend()
plt.show()

# Standard deviation

# Calculate the skewness for the training set
```

```
data_train_std = np.std(data_train_final, axis=0).reshape(1, -1)
std_plot = data_train_std.flatten()

# Calculate the skewness for the testing set
data_test_std = np.std(data_test_final, axis=0).reshape(1, -1)

plt.figure(figsize=(10, 6))
labels = ['Healthy', 'Roller fault', 'Inner race fault', 'Outer race fault', '
    Inner race & Roller fault', 'Inner race & Outer race fault', 'Outer & Inner
    race & Roller fault', 'Outer race & Roller fault']
colors = ['green', 'orange', 'red', 'pink', 'brown', 'yellow', 'black', 'violet']
for i in range(0,8,1):
  plt.plot(np.arange(255), std_plot[i*256:256*(i+1)-1], label=labels[i], color=
      colors[i])
plt.title("Feature Visualization - Standard deviation")
plt.xlabel("Samples")
plt.ylabel("Standard deviation Value")
plt.legend()
plt.show()

# Skewness

# Calculate the skewness for the training set
data_train_skew = skew(data_train_final, axis=0).reshape(1, -1)
skew_plot = data_train_skew.flatten()

# Calculate the skewness for the testing set
data_test_skew = skew(data_test_final, axis=0).reshape(1, -1)

plt.figure(figsize=(10, 6))
labels = ['Healthy', 'Roller fault', 'Inner race fault', 'Outer race fault', '
    Inner race & Roller fault', 'Inner race & Outer race fault', 'Outer & Inner
    race & Roller fault', 'Outer race & Roller fault']
colors = ['green', 'orange', 'red', 'pink', 'brown', 'yellow', 'black', 'violet']
for i in range(0,8,1):
  plt.plot(np.arange(255), skew_plot[i*256:256*(i+1)-1], label=labels[i], color=
      colors[i])
plt.title("Feature Visualization - Skewness")
plt.xlabel("Samples")
plt.ylabel("Skewness Value")
plt.legend()
plt.show()

# Kurtosis

# Calculate the kurtosis for the training set
data_train_kurtosis = kurtosis(data_train_final, axis=0).reshape(1, -1)
```

```
kurtosis_plot = data_train_kurtosis.flatten()
print(data_train_kurtosis.shape)
# Calculate the kurtosis for the testing set
data_test_kurtosis = kurtosis(data_test_final, axis=0).reshape(1, -1)

plt.figure(figsize=(10,6))
labels = ['Healthy', 'Roller fault', 'Inner race fault', 'Outer race fault', '
   Inner race & Roller fault', 'Inner race & Outer race fault', 'Outer & Inner
   race & Roller fault', 'Outer race & Roller fault']
colors = ['green', 'orange', 'red', 'pink', 'brown', 'yellow', 'black', 'violet']
for i in range(0,8,1):
  plt.plot(np.arange(255), kurtosis_plot[i*256:256*(i+1)-1], label=labels[i],
     color=colors[i])
plt.title("Feature Visualization - Kurtosis")
plt.xlabel("Samples")
plt.ylabel("Kurtosis Value")
plt.legend()
plt.show()

# Feature stack

training_features = np.concatenate((data_train_std.T, data_train_skew.T,
   data_train_freq.T), axis=1)
testing_features = np.concatenate((data_test_std.T, data_test_skew.T,
   data_test_freq.T), axis=1)
print(training_features.shape)
print(testing_features.shape)

# PCA using xplained variance to understand how many components to keep

# Apply PCA for training features
pca = PCA()
pca.fit(training_features)
data_train_pca = pca.transform(training_features)

# Apply PCA for test features
pca_test = PCA()
pca_test.fit(testing_features)
data_test_pca = pca_test.transform(testing_features)

# Explained variance to understand how many components to keep
explained_variance = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance)

# Explained variance to understand how many components to keep
explained_variance_test = pca_test.explained_variance_ratio_
cumulative_variance_test = np.cumsum(explained_variance_test)
```

```
# Print out explained variance for each component
print("Explained variance per component for training features:",
    explained_variance)
print("Cumulative explained variance for training features:", cumulative_variance
    )

# Print out explained variance for each component
print("Explained variance per component for test features:",
    explained_variance_test)
print("Cumulative explained variance for test features:",
    cumulative_variance_test)

# Select number of components based on cumulative variance (e.g., 95% of total
    variance)
n_components = np.argmax(cumulative_variance >= 0.95) + 1 # Number of components
    to retain 95% variance
pca_train = PCA(n_components=n_components)
data_train_reduced = pca_train.fit_transform(training_features)

# Select number of components based on cumulative variance (e.g., 95% of total
    variance)
n_components = np.argmax(cumulative_variance >= 0.95) + 1 # Number of components
    to retain 95% variance
pca_test = PCA(n_components=n_components)
data_test_reduced = pca_train.transform(testing_features)

# Output the reduced data
print(f"Reduced data shape: {data_train_reduced.shape}")
print(data_train_reduced)

# Output the reduced data
print(f"Reduced data shape: {data_test_reduced.shape}")
print(data_test_reduced)

# SOM-MQE model

somMQEModel = SOM(m=10, n=10, dim=26)
somMQEModel.fit(data_train_reduced[0:255,:], epochs=1000)
transformedValues = somMQEModel.transform(data_train_reduced)
transformedTesting = somMQEModel.transform(data_test_reduced)
distances = [np.mean(transformedValues[i,:]) for i in range(np.shape(
    transformedValues)[0])]
distancesTest = [np.mean(transformedTesting[i,:]) for i in range(np.shape(
    transformedTesting)[0])]
print('Model = ',i+1)
plt.figure(figsize=(10, 6))
plt.plot(distancesTest)
plt.title('SOM-MQE Testing')
plt.xlabel('Sample Number')
plt.ylabel('Euclidean Distance (2d)')
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(distances)
plt.title('SOM-MQE Training')
plt.xlabel('Sample Number')
plt.ylabel('Euclidean Distance (2d)')
plt.show()

from sklearn.multiclass import OneVsRestClassifier
from sklearn import svm
from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix

data_trainlbs_final = data_trainlbs_final.reshape(-1)
data_testlbs_final = np.concatenate((np.ones(shape=(1,64)),2*np.ones(shape=(1,64)
    ),3*np.ones(shape=(1,64)),4*np.ones(shape=(1,64)),5*np.ones(shape=(1,64)),6*np
    .ones(shape=(1,64)),7*np.ones(shape=(1,64)),8*np.ones(shape=(1,64))), axis=1).
    reshape(-1)

# Create the base SVM model
base_svm = svm.SVC(probability=True, kernel='rbf', C=1.0, gamma='scale')

# Create the One-vs-Rest classifier
ovr_svm = OneVsRestClassifier(base_svm)
ovr_svm.fit(data_train_reduced, data_trainlbs_final)
ovr_predictions = ovr_svm.predict(data_train_reduced)

# Evaluate the model
ovr_accuracy = accuracy_score(data_trainlbs_final, ovr_predictions)
print(f"One-vs-Rest Accuracy: {ovr_accuracy * 100:.2f}%")

# Print classification report
print("\nOne-vs-Rest Classification Report:")
print(classification_report(data_trainlbs_final, ovr_predictions))

# Compute confusion matrix
conf_matrix = confusion_matrix(data_trainlbs_final, ovr_predictions)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=np.
    unique(data_trainlbs_final))
disp.plot(cmap='viridis', xticks_rotation='vertical')
```

```
plt.title("SVM Model - one-vs-Rest classifier - Training")
plt.show()

# Plot the probability
train_probability = ovr_svm.predict_proba(data_train_reduced)
test_probability = ovr_svm.predict_proba(data_test_reduced)

plt.figure(figsize=(12, 8))
plt.plot(train_probability, marker='.')
plt.title('Probability plot - one-vs-Rest classifier - Training')
plt.legend(['Label 1', 'Label 2', 'Label 3', 'Label 4', 'Label 5', 'Label 6', '
   Label 7', 'Label 8'])
plt.xlabel('Sample Number')
plt.ylabel('Probability')
plt.show()

plt.figure(figsize=(12, 8))
plt.plot(test_probability, marker='.')
plt.title('Probability plot - one-vs-Rest classifier - Testing')
plt.legend(['Label 1', 'Label 2', 'Label 3', 'Label 4', 'Label 5', 'Label 6', '
   Label 7', 'Label 8'])
plt.xlabel('Sample Number')
plt.ylabel('Probability')
plt.show()

from sklearn.multiclass import OneVsOneClassifier
# Create the One-vs-One classifier
ovo_svm = OneVsOneClassifier(base_svm)

# Train the model
ovo_svm.fit(data_train_reduced, data_trainlbs_final)

# Predict on the test set
ovo_predictions = ovo_svm.predict(data_train_reduced)

# Evaluate the model
ovo_accuracy = accuracy_score(data_trainlbs_final, ovo_predictions)
print(f"One-vs-One Accuracy: {ovo_accuracy * 100:.2f}%")

# Print classification report
print("\nOne-vs-One Classification Report:")
print(classification_report(data_trainlbs_final, ovo_predictions))

# Compute confusion matrix
conf_matrix = confusion_matrix(data_trainlbs_final, ovo_predictions)

# Display the confusion matrix
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=np.
   unique(data_trainlbs_final))
disp.plot(cmap='viridis', xticks_rotation='vertical')
plt.title("SVM Model - one-vs-one classifier - Training")
plt.show()

# ANN MODEL
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, accuracy_score
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix
from sklearn.model_selection import GridSearchCV

param_grid = {
     'hidden_layer_sizes': [(128, 64)],
    'activation': ['relu', 'tanh'],
    'alpha': [ 0.01],
    'learning_rate_init': [ 0.01 ],
    'solver': ['adam']
}

grid_search = GridSearchCV(
    estimator=MLPClassifier(max_iter=500, random_state=42),
    param_grid=param_grid,
    cv=5,
    scoring='accuracy'
)

grid_search.fit(data_train_reduced, data_trainlbs_final)
best_model = grid_search.best_estimator_
print("Best Parameters:", grid_search.best_params_)

# Make predictions
ann_predictions = best_model.predict(data_train_reduced)

# Evaluate the model
ann_accuracy = accuracy_score(data_trainlbs_final, ann_predictions)
print(f"ANN Accuracy : {ann_accuracy * 100:.2f}%")

# Print classification report
print("Classification Report:")
print(classification_report(data_trainlbs_final, ann_predictions))

# Compute confusion matrix
conf_matrix = confusion_matrix(data_trainlbs_final, ann_predictions)
```

```
# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix, display_labels=np.
    unique(data_trainlbs_final))
disp.plot(cmap='viridis', xticks_rotation='vertical')
plt.title("ANN Model - Training")
plt.show()

# Plot the probability
train_probability = best_model.predict_proba(data_train_reduced)
test_probability = best_model.predict_proba(data_test_reduced)

plt.figure(figsize=(12, 8))
plt.plot(train_probability, marker='.')
plt.title('Probability - Training for ANN model')
plt.legend(['Label 1', 'Label 2', 'Label 3', 'Label 4', 'Label 5', 'Label 6', '
    Label 7', 'Label 8'])
plt.xlabel('Sample Number')
plt.ylabel('Probability')
plt.show()

plt.figure(figsize=(12, 8))
plt.plot(test_probability, marker='.')
plt.title('Probability - Testing for for ANN model')
plt.legend(['Label 1', 'Label 2', 'Label 3', 'Label 4', 'Label 5', 'Label 6', '
    Label 7', 'Label 8'])
plt.xlabel('Sample Number')
plt.ylabel('Probability')
plt.show()

# # Load the training data from the .mat file and print to check the structure
# training_data_mat = loadmat('data_train.mat')
# # Print the keys
# print(training_data_mat)
# # Write the training data into a csv file for better visualization and handling
# data_train = training_data_mat['data_train']
# data_train_flatten = []
# # Loop through each sub-array in data_train and flatten it
# for i, array in enumerate(data_train.flat):
#     if isinstance(array, np.ndarray):
#         flattened_row = array.flatten()
#         data_train_flatten.append(flattened_row)
# # Store the data in a DataFrame
# training_data_df = pd.DataFrame(data_train_flatten)
# training_data_df = training_data_df.T
# directory = '/content/drive/MyDrive/ENME691/final_project/Project 3/Dataset for
     Team'
# file_path = f'{directory}/train_data.csv'
# training_data_df.to_csv(file_path, index=False)
# Check for the shape of the training data

# # Load the testing data from the .mat file and print to check the structure
# testing_data_mat = loadmat('data_test.mat')
# # Print the keys
# # print(testing_data_mat)
# # Write the testing data into a csv file for better visualization and handling
# data_test = testing_data_mat['data_test']
# data_test_flatten = []
# # Loop through each sub-array in data_test and flatten it
# for i, array in enumerate(data_test.flat):
#     if isinstance(array, np.ndarray):
#         flattened_row = array.flatten()
#         data_test_flatten.append(flattened_row)
# # Store the data in a DataFrame
# testing_data_df = pd.DataFrame(data_test_flatten)
# testing_data_df = testing_data_df.T
# directory = '/content/drive/MyDrive/ENME691/final_project/Project 3/Dataset for
    Team'
# file_path = f'{directory}/test_data.csv'
# testing_data_df.to_csv(file_path, index=False)
# Check for the shape of the testing data
# file_path = '/content/drive/MyDrive/ENME691/final_project/Project 3/Dataset for
    Team/test_data_modified.csv'
# data_test_csv = pd.read_csv(file_path)
# num_rows, num_columns = data_test_csv.shape
# print("Number of rows",num_rows)
# print("Number of columns",num_columns)
# Store the testing data in a numpy array
# data_test_mod = data_test_csv.to_numpy()
# print(data_test_mod.shape)

# data_test_1 = np.random.rand(8192, 1)
# data_test_2 = np.random.rand(8192, 1)
# data_test_3 = np.random.rand(8192, 1)
# data_test_4 = np.random.rand(8192, 1)
# data_test_5 = np.random.rand(8192, 1)
# data_test_6 = np.random.rand(8192, 1)
# data_test_7 = np.random.rand(8192, 1)
# data_test_8 = np.random.rand(8192, 1)
# for i in range(512): # Loop through all columns
#     if data_test_mod[0, i] == 1:
#         data_test_1 = np.concatenate((data_test_1, data_test_mod[1:, i].reshape
    (-1, 1)), axis=1)
#     elif data_test_mod[0, i] == 2:
```

```
#        data_test_2 = np.concatenate((data_test_2, data_test_mod[1:, i].reshape
    (-1, 1)), axis=1)
#    elif data_test_mod[0, i] == 3:
#        data_test_3 = np.concatenate((data_test_3, data_test_mod[1:, i].reshape
    (-1, 1)), axis=1)
#    elif data_test_mod[0, i] == 4:
#        data_test_4 = np.concatenate((data_test_4, data_test_mod[1:, i].reshape
    (-1, 1)), axis=1)
#    elif data_test_mod[0, i] == 5:
#        data_test_5 = np.concatenate((data_test_5, data_test_mod[1:, i].reshape
    (-1, 1)), axis=1)
#    elif data_test_mod[0, i] == 6:
#        data_test_6 = np.concatenate((data_test_6, data_test_mod[1:, i].reshape
    (-1, 1)), axis=1)
#    elif data_test_mod[0, i] == 7:
#        data_test_7 = np.concatenate((data_test_7, data_test_mod[1:, i].reshape
    (-1, 1)), axis=1)
#    elif data_test_mod[0, i] == 8:
#        data_test_8 = np.concatenate((data_test_8, data_test_mod[1:, i].reshape
    (-1, 1)), axis=1)

# # Final concatenation
# data_test_final = np.concatenate((data_test_1[:, 1:], data_test_2[:, 1:],
    data_test_3[:, 1:],data_test_4[:, 1:], data_test_5[:, 1:], data_test_6[:, 1:],
    data_test_7[:, 1:], data_test_8[:, 1:]),axis=1)

# print(data_test_final[:,64])

# output_file = "data_test_final.csv"
# np.savetxt(output_file, data_test_final, delimiter=",", fmt="%.6f")


# # Load the training data labels from the .mat file and print to check the
    structure
# training_datalbs_mat = loadmat('data_train_labels.mat')
# # Print the keys
# print(training_datalbs_mat)
# # Write the training data labels into a csv file for better visualization and
    handling
# data_train_labels = training_datalbs_mat['data_train_labels']
# # Store the data in a DataFrame
# training_datalbs_df = pd.DataFrame(data_train_labels)
# directory = '/content/drive/MyDrive/ENME691/final_project/Project 3/Dataset for
     Team'
# file_path = f'{directory}/train_data_labels.csv'
# training_datalbs_df.to_csv(file_path, index=False)
# Check for the shape of the training data labels
# # Load the testing data labels from the .mat file and print to check the
    structure
# testing_datalbs_mat = loadmat('data_test_labels.mat')

# # Write the testing data labels into a csv file for better visualization and
    handling
# data_test_labels = testing_datalbs_mat['data_test_labels']
# # Store the data in a DataFrame
# testing_datalbs_df = pd.DataFrame(data_test_labels)
# directory = '/content/drive/MyDrive/ENME691/final_project/Project 3/Dataset for
     Team'
# file_path = f'{directory}/test_data_labels.csv'
# testing_datalbs_df.to_csv(file_path, index=False)
# Check for the shape of the training data labels


# import math
# D = 60 # mm
# d = 10 # mm
# n= 17
# angle = 14.94 * math.pi
# f_i = 800 / 60 # Hz

# BPFO_freq = n * f_i / 2 * (1- (d/D *math.cos(angle)))
# BPFI_freq = n * f_i / 2 * (1+ (d/D *math.cos(angle)))
# BFF_freq = f_i * (D / d) * (1 - (d / D * math.cos(angle))**2)
# FTF_freq = f_i / 2 * (1- (d/D *math.cos(angle)))
# print("BPFO:", BPFO_freq)
# print("BPFI:", BPFI_freq)
# print("BFF:", BFF_freq)
# print("FTF:", FTF_freq)

# # Ball Pass Frequency Inner (BPFI) of 95 Hz
# # Calculate the index corresponding to the frequency closest to 95 Hz
# BPFI_index = np.argmin(np.abs(frequency_axis - BPFI_freq))
# # Extract the magnitude at the BPFI for training and test data
# BPFI_train_magnitude = np.abs(data_train_frequency[BPFI_index, :])
# BPFI_test_magnitude = np.abs(data_test_frequency[BPFI_index, :])
# # Print or use the BPFI magnitudes for classification
# print("BPFI magnitude in training data:", BPFI_train_magnitude)
# print("Shape of BPFI", BPFI_train_magnitude.shape)
# plt.figure(figsize=(10, 6))
# labels = ['Healthy', 'Roller fault', 'Inner race fault', 'Outer race fault', '
    Inner race & Roller fault', 'Inner race & Outer race fault', 'Outer & Inner
    race & Roller fault', 'Outer race & Roller fault']
```

```
# colors = ['green', 'orange', 'red', 'pink', 'brown', 'yellow', 'black', 'violet
    ']
# for i in range(0,8,1):
#   plt.plot(np.arange(255), BPFI_train_magnitude[i*256:256*(i+1)-1], label=labels
    [i], color=colors[i])
# plt.title("Feature Visualization - BPFI Magnitude")
# plt.xlabel("Samples")
# plt.ylabel("Magnitude at 99 Hz (BPFI)")
# plt.legend()
# plt.show()

# #  Ball pass frequency of the outer race (BPFO) of 131 Hz
# # Calculate the index corresponding to the frequency closest to 131 Hz
# BPFO_index = np.argmin(np.abs(frequency_axis - BPFO_freq))

# # Extract the magnitude at the BPFO for training and test data
# BPFO_train_magnitude = np.abs(data_train_frequency[BPFO_index, :])
# BPFO_test_magnitude = np.abs(data_test_frequency[BPFO_index, :])

# # Print or use the BPFO magnitudes for classification
# print("BPFO magnitude in training data:", BPFO_train_magnitude)
# print("Shape of BPFO", BPFO_train_magnitude.shape)


# plt.figure(figsize=(10, 6))
# labels = ['Healthy', 'Roller fault', 'Inner race fault', 'Outer race fault', '
    Inner race & Roller fault', 'Inner race & Outer race fault', 'Outer & Inner
    race & Roller fault', 'Outer race & Roller fault']
# colors = ['green', 'orange', 'red', 'pink', 'brown', 'yellow', 'black', 'violet
    ']
# for i in range(0,8,1):
#   plt.plot(np.arange(255), BPFO_train_magnitude[i*256:256*(i+1)-1], label=labels
    [i], color=colors[i])
# plt.title("Feature Visualization - BPFO Magnitude")
# plt.xlabel("Samples")
# plt.ylabel("Magnitude at 131 Hz (BPFO)")
# plt.legend()
# plt.show()

# #  Fundamental train frequency (FTF) of 7.7 Hz
# # Calculate the index corresponding to the frequency closest to 7.7 Hz
# FTF_index = np.argmin(np.abs(frequency_axis - FTF_freq))

# # Extract the magnitude at the FTF for training and test data
# FTF_train_magnitude = np.abs(data_train_frequency[FTF_index, :])
# FTF_test_magnitude = np.abs(data_test_frequency[FTF_index, :])
# # Print or use the FTF magnitudes for classification
# print("FTF magnitude in training data:", FTF_train_magnitude)
# print("Shape of FTF", FTF_train_magnitude.shape)

# plt.figure(figsize=(10, 6))
# labels = ['Healthy', 'Roller fault', 'Inner race fault', 'Outer race fault', '
    Inner race & Roller fault', 'Inner race & Outer race fault', 'Outer & Inner
    race & Roller fault', 'Outer race & Roller fault']
# colors = ['green', 'orange', 'red', 'pink', 'brown', 'yellow', 'black', 'violet
    ']
# for i in range(0,8,1):
#   plt.plot(np.arange(255), FTF_train_magnitude[i*256:256*(i+1)-1], label=labels[
    i], color=colors[i])
# plt.title("Feature Visualization - FTF Magnitude")
# plt.xlabel("Samples")
# plt.ylabel("Magnitude at 7.7 Hz (FTF)")
# plt.legend()
# plt.show()

# #  Ball Fault Frequency (BFF) of 77 Hz
# # Calculate the index corresponding to the frequency closest to 77 Hz
# BFF_index = np.argmin(np.abs(frequency_axis - BFF_freq))

# # Extract the magnitude at the BFF for training and test data
# BFF_train_magnitude = np.abs(data_train_frequency[BFF_index, :])
# BFF_test_magnitude = np.abs(data_test_frequency[BFF_index, :])

# # Print or use the BFF magnitudes for classification
# print("BFF magnitude in training data:", BFF_train_magnitude)
# print("Shape of BFF", BFF_train_magnitude.shape)

# plt.figure(figsize=(10, 6))
# labels = ['Healthy', 'Roller fault', 'Inner race fault', 'Outer race fault', '
    Inner race & Roller fault', 'Inner race & Outer race fault', 'Outer & Inner
    race & Roller fault', 'Outer race & Roller fault']
# colors = ['green', 'orange', 'red', 'pink', 'brown', 'yellow', 'black', 'violet
    ']
# for i in range(0,8,1):
#   plt.plot(np.arange(255), BFF_train_magnitude[i*256:256*(i+1)-1], label=labels[
    i], color=colors[i])
# plt.title("Feature Visualization - BFF Magnitude")
# plt.xlabel("Samples")
# plt.ylabel("Magnitude at 77 Hz (BFF)")
# plt.legend()
# plt.show()
# from sklearn import svm
# from sklearn.metrics import confusion_matrix

# AllAtOnceSVM =svm.SVC()
# data_trainlbs_final = data_trainlbs_final.reshape(-1)
# # data_testlbs_final = data_testlbs_final.reshape(-1)
# data_testlbs_final = np.concatenate((np.ones(shape=(1,64)),2*np.ones(shape
   =(1,64)),3*np.ones(shape=(1,64)),4*np.ones(shape=(1,64)),5*np.ones(shape
   =(1,64)),6*np.ones(shape=(1,64)),7*np.ones(shape=(1,64)),8*np.ones(shape
   =(1,64))), axis=1).reshape(-1)

# AllAtOnceSVM.fit(data_train_reduced, data_trainlbs_final)
# # confMTrain = confusion_matrix(data_trainlbs_final, AllAtOnceSVM.predict(
   data_train_reduced))
# # print(confMTrain)
# # confMTest = confusion_matrix(data_testlbs_final, AllAtOnceSVM.predict(
   data_test_reduced))
# # print(confMTest)
# print(AllAtOnceSVM.predict(data_train_reduced))
# print(AllAtOnceSVM.predict(data_test_reduced))
# print(AllAtOnceSVM.score(data_train_reduced, data_trainlbs_final))
# print(AllAtOnceSVM.score(data_test_reduced, data_testlbs_final))
```