Universidad Carlos III de Madrid

Bachelor in Telecommunication Technologies Engineering

ML Classifier for Fault Diagnosis in Rotary Machines

Alberto del Río Jara……100429823

INDEX

ML Classifier for Fault Diagnosis in Rotary Machines

Abstract

Rotary machines are a vital part of industrial machinery and are used across many different industries. However, these machines are naturally subject to faulty parts due to age. These faults must currently be checked through scheduled maintenance, but to better combat these issues, engineers need a way to collect data and better predict faulty parts [4]. Machine Learning (ML) has become an increasingly popular solution to the problem of collecting data and classifying machine faults. We want to design a machine learning program that will be able to use different algorithms to improve the classification and storage of fault diagnosis data.

There are many different types of classifiers that are useful for different tasks. Our goal is to use these classifiers, put them in one program, and allow engineers to easily collect and analyze data from these classifiers to diagnose machine faults. Machine learning can change the way people collect and analyze data; it can decrease the amount of time and resources dedicated to completing a specific task — this is the case for fault diagnosis. The integration of machine learning models can be complex and difficult, but the ability to predict and identify machine faults quickly and efficiently is essential to the development of machine fault diagnosis.

1.0 Problem

Rotating machines are widely used in many industries, such as manufacturing and energy. However, these machines are prone to developing faults over time, which can lead to unexpected breakdowns. This poses a serious problem because, without proper fault detection and monitoring, companies can face costly downtime, expensive repairs, and even safety risks. These machines often operate continuously, making any sudden failure a significant issue that disrupts production and leads to financial losses. Therefore, the need to monitor machine health and detect faults before they escalate is critical.

Current maintenance approaches, like scheduled maintenance, replace parts based on a predetermined schedule rather than their actual condition. This method can result in two undesirable outcomes: either the machine fails before the maintenance occurs, leading to unplanned downtime, or the parts are replaced prematurely, wasting resources and money. Both scenarios highlight the inefficiency of not having a real-time, condition-based monitoring system. This lack of a solution means that industries remain vulnerable to operational disruptions and increased maintenance costs.

The absence of effective fault detection can also impact the safety of workers. Sudden machine failures can cause accidents, making it important to have a system that detects and predicts these issues in advance. Furthermore, as machines become more complex, identifying and diagnosing faults becomes more challenging. This emphasizes the importance of developing an automated system capable of continuously monitoring machines and predicting potential failures, ensuring not only efficiency but also workplace safety.

2.0 Objective

The objective of this project is to develop a machine learning-based tool for real-time fault diagnosis in rotary machines, which are critical components in many industries. The system aims to leverage various machine learning classifiers to analyze sensor data from these machines and identify potential faults before they lead to major failures, minimizing downtime and improving operational efficiency [7].

The tool will collect and process real-time vibration data from accelerometers attached to rotary machines. This data will then be used to train and apply multiple machine learning classifiers, enabling the system to detect and classify faults such as bearing failures, misalignments, and other issues [7]. By utilizing a range of classifiers, the system will ensure robust and accurate fault detection that is adaptable to different machine conditions and fault types.

Additionally, the project aims to provide a user-friendly interface that allows engineers to interact with the system, select classifiers, modify feature extraction methods, and train the tool using historical data. This flexibility will make it easier for users to fine-tune the system for their specific needs, ensuring a tailored approach to fault diagnosis.

The overall goal is to create a platform that offers real-time, accurate, and scalable fault detection, reducing maintenance costs, improving machine reliability, and enhancing the ability of industries to monitor the health of their equipment. The system's integration of multiple machine learning classifiers will push the boundaries of current predictive maintenance technologies, providing a cutting-edge solution for rotary machine health monitoring.

3.0 Background

3.1 Key Concepts

The Machine Fault Simulators was created by SpectraQuest, Inc. and its main purpose is to gain an understanding of different vibration signatures for industrial machine health diagnosis [1]. The simulation machine can mimic the conditions of real-world machine faults without affecting real-world production [1]. Machine diagnosticians can use the machine to gain expertise and experience in diagnosing these different faults and gain a deeper understanding of real-world vibration spectra [1]. The Machine Fault Simulator can also collect data on bearings that cause faults and those that don’t. For our Machine Classifiers, we can collect data from the Machine Fault Simulator from multiple scenarios of single and multi-fault rotary machines. We can use this data to train and test Machine Learning algorithms for rotary machine fault diagnosis. Data is collected from the fault simulator by switching out nominal components on the fault simulator with one or more faulty components [5]. The machine is then turned on and set to a specific rpm value [5]. Data is then collected for 10 seconds by the sensors once the target speed has been reached [5]. The process is repeated for each faulty component that is being tested [5].

Machine Learning is a field of computer science that uses data and statistical analysis to draw inferences from patterns in data [2]. Machine learning is a vast field with multiple different algorithms that can be used to learn from and classify data. Machine learning models are divided into three different methods of training data: Supervised, Unsupervised, and

Semi-supervised learning [3]. Supervised learning trains algorithms on labeled datasets to predict outcomes by changing the weights of the model to produce the most accurate outcome [3]. Unsupervised learning trains algorithms on unlabeled datasets to analyze and cluster data to find new patterns and information from the datasets [3]. Semi-supervised learning is a combination of supervised and unsupervised learning that "uses smaller labeled datasets to guide classification and feature extraction from a larger, unlabeled dataset" [3]. For each of these methods, programmers can use different Machine Learning algorithms to help with classification. Some of the most popular algorithms include Decision Trees, Neural Networks, Naïve Bayes, Clustering, K-means, SVM, K-nearest neighbor, Logistic Regression, and Linear Regression. These different machine learning algorithms can be used for specific tasks that best analyze the data for classification and prediction. For example, neural networks are great at recognizing patterns and are used in fields such as natural language translation and speech recognition [3]. Once the data has been trained and tested on machine learning algorithms, programmers must assess their model's performance. Different assessments can be used to determine the performance of a machine learning model, including the confusion matrix, accuracy, precision, recall, and F1 scores.

3.2 Related Work

Machine learning in machine health diagnostics has become a growing research area. Machine learning algorithms would provide a better maintenance strategy for industrial machinery and improve machine lifetime and plant profits by continuously checking for machine faults rather than checking manually. A research paper by Saha et al. investigated using machine learning to diagnose bearing faults using an SVM algorithm. The paper investigated four common bearing faults: ball fault, outer race fault, inner race fault, and cage fault [4]. The methodology includes data generation, signal conditioning, feature extraction, model development, and model prediction [4]. The data generation stage generated the vibration data using an accelerometer sensor mounted on the bearing housing [4]. The signal conditioning stage changed the data into a .csv or .txt data for the software to read [4]. The feature extraction stage decided which features from the dataset would be used to train and test the model [4]. These include time-domain features like kurtosis: the tailedness of a distribution of variables, crest factor: the difference between the peak and average of a signal in decibels, and form factors: the specification of components in relation to their design [4]. The model development stage decided the approach for fault detection using a machine learning algorithm [4]. Finally, the model prediction stage analyzed the results from the classification model using accuracy, precision, recall, and F1 score [4]. The developers used the support vector machine as their machine learning approach by finding "the optimal hyperplane that differentiates the training data into two binary classes by maximizing the margin" [4]. The developers also used Particle Swarm Optimization to determine a variable that "optimizes the parameters based on function f(x), where f(x) is a fitness function or objective function" [4]. After training and testing the SVM and PSO-SVM algorithms, a confusion matrix was used to calculate accuracy, precision, recall, and F1 score [4]. Overall, the SVM performed well with 92% accuracy, with an improvement of almost 2% using the PSO-SVM model [4].

Currently, this problem has been tackled before by the Mechanical Engineering department at the University of Arkansas. The datasets used for this experiment tested 38 different fault scenarios at three different RPMs with 25 datasets collected for each test [6]. The features were then extracted from the datasets and split into 80% training and 20% testing data [6]. There were five different machine learning algorithms used on the testing data including

K-nearest neighbor, Gaussian Naïve Bayes classifier, Support Vector Machine, Decision Tree, and Random Forest [6]. The results were measured based on how accurate the classifications for each algorithm were on the test set [6]. The accuracy was measured based on partially correct outcomes as well as correct outcomes [6]. This weighs a classification that might be partially correct higher than one that is completely wrong, so if the model classifies the correct location but the wrong fault it will give partial accuracy to the model [6]. The KNN algorithm had an accuracy of 77.1794%, the SVM machine had an accuracy of 82.0512%, the Gaussian Naïve Bayes algorithm had an accuracy of 93.0769%, The Decision Tree algorithm had an accuracy of 98.4515%, and finally, the Random Forest algorithm had an accuracy of 99.979% [6]. The Random Forest resulted in the highest accuracy score and was used to train and test the datasets of three different frequency levels for each fault [6]. They were initially tested individually on 25Hz, 50Hz, and 75Hz with high accuracy scores. They were then trained on training data of 25 Hz and tested on 50 Hz which did not perform well, with an accuracy of 22.0513% [6]. The training data was then split evenly between 50 Hz and 25 Hz datasets as well as split into thirds for 25 Hz, 50 Hz, and 75 Hz. When trained on different datasets the accuracy improved to above 99% for each frequency level [6].

Our implementation will be similar to other machine learning algorithm implementations used to determine machine faults. The biggest difference in our implementation is the datasets that need to be processed. Other projects mainly focus on one type of mechanical fault. Saha et al. focused on bearing faults specifically, but there are other types of faults it does not take into consideration or look at such as gear pitting faults [4]. Our project focuses on both multiple different single-fault and multi-fault cases. This allows the machine learning algorithm to determine more faults as well as combinations of faults with more accuracy than a machine learning algorithm that is only applicable to one specific type of fault. To improve upon the original design of this project, we would like to train and test regression models to investigate classifying data that the model has and hasn't been trained on. This would give us a higher level of accuracy over a wider range of fault simulations, therefore developing machine learning algorithms that would better adapt to determining different kinds of faults with better accuracy and quickness.

4.0 Design

4.1 Use Cases

The users of our program are Dr. Jensen and students of the MEEG department with access to Dr. Jensen’s lab at the ENRC. They are able to do the following with our design:

- Run the program with little to no knowledge of ML/classifiers
- Import old and new data into the program from a CSV file/s
- Perform feature extraction on the data in the uploaded CSV file/s using statistical analysis so that the data can be turned into meaningful features which then can be classified
- Choose which model/s (SVM, Decision Tree, Linear Regression, Logistic Regression, or Neural Network) the user would like to use to classify the data
- Perform analysis through accuracy, precision, recall, F1-score, and a confusion matrix to determine the success of each ML model used
- Export the data they used along with the results from the classifiers and analysis of the classifiers to a CSV file

Our program is user friendly to faculty and students from the MEEG department and can be used alongside the Machine Fault Simulator. Our program uses prompts and friendly user

interfaces to make it easily accessible for users without prior knowledge of machine learning or programming. So, the user can run the program and easily download data, perform feature extraction, run machine learning models, and print performance metrics quickly and efficiently without a background in machine learning.

4.2 Requirements

4.2.1 Functional Requirements

Our program is designed to analyze sensor data to detect and categorize faults in rotary machines. The process begins with data ingestion and preprocessing, where raw data from sensors measuring vibrations and rotational speed is collected. These measurements are stored in structured files, capturing frequency-domain and time-domain data. The frequency-domain data represents patterns in vibration signals over time, while the time-domain data records direct signal fluctuations. This combination provides a comprehensive view of the machine's condition. Before further analysis, the system ensures that all data is cleaned, formatted correctly, and free of inconsistencies. Noise filtering and missing data handling improve the accuracy of subsequent steps, resulting in a well-structured dataset ready for processing.

Next, the system applies feature extraction to transform raw sensor data into meaningful indicators. Time-domain analysis helps identify sudden changes in vibrations by extracting values such as peak amplitude, mean, standard deviation, and variance. On the other hand, frequency-domain analysis converts time-based signals into spectral components, revealing periodic faults that may not be obvious in direct readings. By combining both approaches, the system ensures that even subtle faults can be detected. These extracted features serve as input for machine learning models, allowing them to differentiate between normal and faulty operating conditions with greater accuracy.

After extracting the most relevant features, the system performs feature selection to refine the dataset further. By prioritizing essential attributes, the system reduces data complexity while maintaining classification accuracy. This eliminates unnecessary information, ensuring that machine learning models focus only on the most significant indicators. The final dataset consists of the most critical features necessary for detecting and categorizing faults, enhancing efficiency and reliability. With the refined dataset, the system proceeds to data splitting, where the information is divided into training and testing subsets. Typically, 80% of the data is allocated for training, while 20% is reserved for testing. This division ensures that the machine learning models have sufficient examples to learn from while also allowing for the evaluation of unseen data. Proper data splitting is crucial for preventing overfitting and ensuring that the trained models can generalize well to new machine conditions.

Once the data is prepared, the system enters the model training phase. Various machine learning models are trained using the structured dataset, allowing them to recognize patterns in vibration and rotational speed readings. The models analyze historical data to learn how different faults manifest in sensor measurements. By fine-tuning parameters and optimizing classification techniques, the system ensures that the models achieve high accuracy in distinguishing normal operations from fault conditions. Following training, the models are applied to fault detection, determining whether a machine is operating normally or exhibiting a fault. The output is a simple binary classification: either a fault is detected, or the machine is functioning as expected. This step plays a crucial role in preventive maintenance, enabling

early identification of issues before they escalate into serious failures. By continuously monitoring machine conditions, the system helps minimize downtime and optimize maintenance schedules.

Beyond basic fault detection, the system also performs fault categorization, assigning detected faults to specific types. This classification step provides deeper insights into the root causes of issues, indicating whether the fault originates from the bearings, shaft, or other machine components. By labeling faults into predefined categories, maintenance teams can take precise corrective actions tailored to the specific issue, improving overall efficiency in machine diagnostics and repairs. To validate the accuracy and reliability of the system, a performance evaluation is conducted. The effectiveness of each classification model is measured using key performance metrics, including accuracy, precision, recall, and F1-score. These metrics provide insight into how well the system differentiates between faulty and non-faulty conditions. Additionally, confusion matrices are generated to highlight areas where classification errors occur, helping to refine and improve the models over time. For models handling continuous predictions, additional evaluation metrics such as mean squared error and R^2 score are applied. Regular performance assessments ensure that the system consistently meets accuracy standards and adapts to new machine conditions effectively.

To assist users in interpreting results, the system includes data visualization capabilities. Graphical representations such as confusion matrices make it easier to understand fault patterns. These visual tools allow users to track machine health over time, detect potential failure trends, and make informed decisions about maintenance actions. The intuitive interface ensures that users with little to no background in machine learning can interact with the system effectively.

By integrating advanced machine learning techniques with structured data analysis, our system provides an effective and user-friendly solution for fault detection and categorization. Through real-time monitoring, accurate classification, and detailed reporting, it enhances machine reliability, reduces maintenance costs, and contributes to more efficient industrial operations.

4.2.2 Interface Requirements

The interface of this project involves a comprehensive approach to integrating hardware and software components to enable accurate and efficient fault detection in rotary machines. The architecture is divided into four main components: data acquisition, feature extraction, machine learning model implementation, and user interaction through a menu-driven interface.

The hardware setup involves connecting accelerometer sensors to a rotary fault simulator to capture vibration data under various operational conditions. These sensors interface with a data acquisition system that converts analog signals into digital format and timestamps the data with RPM measurements. The data is stored in CSV files, ensuring compatibility with subsequent software processes.

The software component for feature extraction is implemented using Python, leveraging powerful data manipulation and analysis libraries such as Pandas, NumPy, and SciPy. The feature extraction process is designed to handle large datasets efficiently, processing data from 25, 50, and 75 RPM trials independently to avoid memory overload. The Frequency Domain feature extraction utilizes the Fast Fourier Transform (FFT) to convert time-series

data into the frequency domain, allowing the calculation of metrics such as maximum frequency, mean frequency, variance, standard deviation, signal power, skewness, and kurtosis. This is achieved through a custom Python script that reads CSV files from the rotary machine fault simulator, applies FFT to each sensor's data, computes the statistical metrics, and saves the extracted features into new CSV files. In the Time Domain feature extraction, the script calculates statistical metrics directly from the raw sensor data, including mean, variance, standard deviation, root mean square (RMS), peak-to-peak values, skewness, and kurtosis. Both feature extraction methods are designed to systematically process all input files, outputting structured datasets ready for machine learning models.

The machine learning component is implemented using the scikit-learn library, incorporating a variety of classifiers such as Decision Tree, Support Vector Machine (SVM), Logistic Regression, Linear Regression, and Neural Networks. These models are utilized for both fault detection (binary classification of fault vs. no-fault) and fault categorization (multiclass classification of specific fault types). The machine learning pipeline involves splitting the data into training and testing sets, standardizing the data (where applicable), and training each model on both Frequency Domain and Time Domain features. Performance metrics, including accuracy, precision, recall, F1-score, confusion matrices, mean squared error, and R2 score, are automatically generated for each model and displayed to the user. The system is designed to run all classifiers in parallel, providing a comprehensive overview of model performance without requiring manual selection or input from the user. This approach ensures that the output is both thorough and easy to interpret, facilitating the identification of the most effective machine learning model for different fault detection scenarios.

In addition to the core functionality, the system includes a user-friendly menu interface that allows the user to select various options for dataset processing and model evaluation. Upon launching the program, the user is prompted to choose between different RPM datasets (25, 50, or 75 RPM), and the type of feature extraction they wish to perform (Frequency Domain or Time Domain). The menu further allows the user to select which machine learning model they would like to apply, with options for Linear Regression, Logistic Regression, Decision Tree, SVM, and Neural Networks. The user can also specify the range of data files they want to process, making it easier to handle large datasets. This menu-driven interface ensures that users can navigate through the entire fault detection pipeline effortlessly, from data loading to model evaluation, providing flexibility and ease of use.

The system interfaces seamlessly between hardware and software. Data collected from the hardware is formatted for software processing, and the machine learning models interact directly with the feature extraction output. The modular design of the software allows for easy updates and maintenance, with distinct modules handling data acquisition, preprocessing, model training, evaluation, and user interaction. The integration of SMOTE for handling imbalanced data and bandpass filtering for noise reduction further enhances the robustness and accuracy of the system, ensuring that the results are reliable and suitable for real-world applications.

4.2.3 Performance Requirements

For feature extraction, our program is handling three different folders, testing 25, 50, and 75 RPM machine fault trial data, with 38 different fault cases each having 25 trials per fault. This results in 950 CSV files per folder. Therefore it is important that our feature extraction

program can handle this amount of data without crashing. Handling all three folders simultaneously resulted in too much memory use and crashed our program. To minimize memory usage, we decided to run one DataFrame at a time, output the csv file, and then empty the data frame so the memory would not be overloaded. Our plan currently is to create a while loop as a part of our menu function to handle one folder at a time, perform feature extraction on the folder, output the CSV file, and then empty the DataFrame to be used for another folder.

The large amounts of data can also cause a long runtime for our machine learning programs. Previously, we planned on running our feature extraction in the same program as our machine learning models, however, this resulted in a run time of up to 20 minutes, which was too long and unrealistic for our real-time scenarios. Preferably, our program would run in less than seven to ten minutes. Our solution to this problem is to run our feature extraction and models in separate programs. Our feature extraction is performed and then the data is placed into a CSV file that can be uploaded into the program for training and testing our machine learning models. This saves time by only running the feature extraction once, which takes seven to ten minutes per folder. So, as long as there are no updates to the folder, the feature extraction CSV files will remain the same and won't have to be run multiple times. This saves minutes of runtime on our program as the program for our machine learning models only takes about one minute to compute and print its results. As long as there is no need to update the feature extraction CSV files, this saves almost six minutes on run time, which is important for detecting faults in real-time scenarios.

To measure the results of our machine learning algorithms, we are looking at four performance metrics: accuracy, precision, recall, and F1-score. Additionally, we are printing the confusion matrix for each of our models, except for the linear regression model because it is a regression model rather than a classification one. Each model type has twelve different models to run, all made to test different data. We have one for each RPM speed (three total) and two for each feature extraction set. This results in 6 different tests for our model. Additionally, we are looking at the results for both the fault category and fault detection. Fault category refers to the specific fault type that the models will classify such as "Shaft Fault" or "Outer Bearing Fault" while fault detection refers to if there is any fault in the machine, but does not specify the type. This results in 12 performance metric outputs for each machine learning model (60 in total). Each algorithm's performance metrics are compared to determine which machine learning algorithms are best suited for classifying different machine faults. Additionally, we are comparing which feature extraction values are best for determining the fault types and if there is a difference in the results for determining specific faults vs. detecting faults in the machine, but not specifying what kind. This will be used as a research tool, so while high-performance scores would be nice and useful, there are no specific performance requirements, rather it is important to test multiple different scenarios for fault detection to research which models are the most accurate, useful and can be applied in real-world scenarios. In short, the process of testing multiple different machine learning models as a way to research which methods are best for fault detection is the main goal of our project. Therefore high performance metrics are not as important as gaining information on what methods are best for fault detection, which methods do or don't work, and why.

Our final results were written in our statistical analysis paper, which documented the performance of each machine learning model as well as the feature extraction methods. We found through the performance metrics that overall, time-domain feature extraction was more successful than frequency-domain feature extraction. Time domain features had a range of 95-100% for fault detection with the most successful models being Decision Tree and Linear

Regression with 100% accuracy and R2 score of 1.0. These models were able to detect the non-fault cases which other models had trouble predicting because the non-fault class had much smaller case numbers. Time domain features had a range of 8-100% for fault categorization with the most successful models being Decision Tree and Linear Regression with 96-98% for Decision Tree and a R2 score of 1.0 for the Linear Regression. The Frequency Features were more successful for the fault detection case with 96% for the SVM, Neural Network, and Logistic Regression models. The fault categorization performance was far lower with 1-8% accuracy with the best model being the Linear Regression model with an R2-score of 0.08. Detection of faults was much more successful than classification of faults in all cases. This is due to the smaller number of classes that must be sorted. For detection, there are only two classes, “faulty” and “not faulty” whereas classification has 38 classes, which is much harder for models to sort. The main issue with the detection code is that it has a small number of non-faulty cases when compared to faulty cases, but this can be remedied by adding more test cases. However, there were still models that were able to detect the faults with 100% accuracy even with this restriction. Overall, fault detection models that seemed to perform better consisted of Neural Networks, Decision Trees, Logistic Regression, and Support Vector Machines with consistently high scores. For fault categorization, models such as Logistic Regression, Decision Trees, and Neural Networks seemed to perform better while the Logistic Regression and SVM models failed to predict fault cases in many scenarios.

4.3 High-Level Architecture

The design of this project involves a comprehensive approach to integrating hardware and software components to enable accurate and efficient fault detection in rotary machines. The architecture is divided into three main components: data acquisition, feature extraction, and machine learning model implementation.

The hardware setup involves connecting accelerometer sensors to a rotary fault simulator to capture vibration data under various operational conditions. These sensors interface with a data acquisition system that converts analog signals into digital format and timestamps the data with RPM measurements. The data is stored in CSV files, ensuring compatibility with subsequent software processes.

The software component for feature extraction is implemented using Python, leveraging powerful data manipulation and analysis libraries such as Pandas, NumPy, and SciPy. The feature extraction process is designed to handle large datasets efficiently, processing data from 25, 50, and 75 RPM trials independently to avoid memory overload. The Frequency Domain feature extraction utilizes the Fast Fourier Transform (FFT) to convert time-series data into the frequency domain, allowing the calculation of metrics such as maximum frequency, mean frequency, variance, standard deviation, signal power, skewness, and kurtosis. This is achieved through a custom Python script that reads CSV files from the rotary machine fault simulator, applies FFT to each sensor's data, computes the statistical metrics, and saves the extracted features into new CSV files. In the Time Domain feature extraction, the script calculates statistical metrics directly from the raw sensor data, including mean, variance, standard deviation, root mean square (RMS), peak-to-peak values, skewness, and kurtosis. Both feature extraction methods are designed to systematically process all input files, outputting structured datasets ready for machine learning models.

The machine learning component is implemented using the scikit-learn library, incorporating a variety of classifiers such as Decision Tree, Support Vector Machine (SVM), Logistic Regression, Linear Regression, and Neural Networks. These models are utilized for both fault detection (binary classification of fault vs. no-fault) and fault categorization (multiclass classification of specific fault types). The machine learning pipeline involves splitting the data into training and testing sets, standardizing the data (where applicable), and training each model on both Frequency Domain and Time Domain features. Performance metrics, including accuracy, precision, recall, F1-score, confusion matrices, mean squared error, and R2 score, are automatically generated for each model and displayed to the user.

A key enhancement to the system is the **interactive terminal-based menu**, which significantly improves usability and efficiency. This menu allows users to navigate the entire process through simple numbered prompts. First, users select the dataset they wish to analyze (25, 50, or 75 RPM), then choose the type of feature extraction (Frequency Domain or Time Domain), and finally select a machine learning model from a list of five options. Once selections are made, the system automatically runs the corresponding processes and displays detailed results on-screen, including classification metrics and confusion matrices. The menu loop also enables users to quickly iterate through multiple configurations without rerunning or modifying any code manually. This interactive design makes the tool accessible for users of all experience levels and supports extensive experimentation with minimal effort.

The system interfaces seamlessly between hardware and software. Data collected from the hardware is formatted for software processing, and the machine learning models interact directly with the feature extraction output. The modular design of the software allows for easy updates and maintenance, with distinct modules handling data acquisition, preprocessing, model training, and evaluation.

4.4 Detailed Architecture

Our program starts by importing all of the machine learning libraries, including Pandas, Scikit Learn, and NumPy, that are used throughout our project to easily manipulate databases, create machine learning models, and create performance metrics. Pandas is a Python Library that is used to manipulate and analyze data. We used this library for feature extraction along with NumPy which is used for mathematical functions and manipulating multidimensional arrays. Scikit Learn is a machine learning and statistical modeling library we used to create our machine learning models. Additionally, it has multiple performance metric functions that measure the accuracy, precision, recall, F1-score, and more for multiple machine learning models. These libraries have been installed and are used with our feature extraction and model development programs.

Next, we downloaded the datasets into our program to perform feature extraction. Feature extraction is the process of transforming raw data into a set of useful features that can be used by machine learning models to classify data. The data we are performing feature extraction on consists of three different folders composed of multiple CSV files. Each folder holds multiple CSV files of Machine Fault Simulator trial data. The folders are separated by RPM speed including 25, 50, and 75 RPM. Each folder has 950 CSV files consisting of different machine fault trials. To download each of these files from the folder, we used the folders path and the glob function and put each file into a DataFrame. We then added each of these DataFrames to an array for each folder. Each DataFrame then went through the feature

extraction process and was simplified to a single row consisting of the feature-extracted data and placed into a final DataFrame that was converted into a CSV file used for training and testing our machine learning models.

Next, we began working on the feature-extraction. The first thing we noticed about our CSV files was, there were multiple repeated rows of time data. We only needed one column of time data, since it was the same for every column. So, we deleted eight of the repeated columns from each DataFrame, halving the number of rows in our DataFrame. After deleting any unnecessary columns, we created our feature extraction method. Before creating our feature extraction method, we first created an array of all the column names that would be created from our method. Then we created a DataFrame called "df_extracted" and set this to null with the column names set to the column names array we created earlier. We have two separate programs for feature extraction: a Time-Domain feature extraction program and a Frequency-Domain feature extraction program. The previous steps for each method are the same, but the data manipulation is different for each program. For the Frequency-Domain feature extraction, we took in the DataFrame of the data we are manipulating, the DataFrame that will store the feature extracted data, the category of fault as an integer, and whether there was a fault detected or not as an integer as parameters. We went through each column in the DataFrame, transformed it into a Fast Fourier Transform, and found the magnitude of the data. We then found the maximum, mean, variance, standard deviation, signal power, skew, and kurtosis of each magnitude value. We added these values into an array called df_freq_vals which holds all the feature-extracted data for a single CSV file. We then appended the fault_detected and fault_category values to the array and appended the array to the DataFrame of feature-extracted data. We repeated this process for every DataFrame in the array and then returned the feature-extracted DataFrame. The Time-Domain feature extraction function is similar to the Frequency-Domain function. They take in the same values and each goes through each column in the DataFrame, but rather than using the Fast Fourier Transform it uses the values of the columns, dropping any null values and converting them to numeric values, and finds the max, mean, variance, standard deviation, kurtosis, skew, precision time protocol and root square mean of each column. These feature-extracted values are appended to the df_time_vals array along with the fault_detection and fault_category values and then added to the DataFrame of feature-extracted values. To get the category and detection values we had to manually add these in since they were not added to the CSV files. The detection values were equal to 1 in all cases, meaning there was a fault detected, except for the no-fault trial CSV files. For the category value, there were 25 trials per fault, so the category was set to 1 and then incremented after every 25th CSV file was put through the feature extraction method. These DataFrames were then converted into CSV files that can be used for our machine learning models.

Once our data had been transformed, we worked on programming, training, and testing our machine-learning models. The machine learning models we have chosen to program are the SVM, Decision Tree, Linear Regression, Logistic Regression, and Neural Network. SVM and decision trees have already been studied for this project, but we are looking at them again with the frequency-domain features to see if there is any change in the model's accuracy. Additionally, we are studying three new models: Linear Regression, Logistic Regression, and Neural Networks to see if there is any improvement in the accuracy of fault prediction using these models compared to the previous models. The SVM model classifies data by finding the hyperplane that maximizes the distance between each class in an N-dimensional space [9]. We are using the SVM models to determine between the fault classes and non-fault classes, so we are finding the hyperplane that separates these classes using our data. The Decision Tree model uses data to create a tree-like structure that uses statistics to split data

based on features used for classification [10]. We are using the data from our DataFrames to find the attributes that best determine how to classify our data and use them to build our tree. Linear regression determines the connection between dependent and independent variables of a dataset, and logistic regression predicts the outcome of a data point based on independent variables by applying logistic functions [10]. These two functions are looking at the independent and dependent functions of the datasets and then finding a linear and logistic function that can be used to predict the class of the data point. Neural Networks use a series of algorithms to recognize patterns and relationships in data [11]. There is an input layer, output layer, and hidden layers used to process the data [11]. The Neural Network model takes in the datasets as input and uses them to adjust the weights and biases of the hidden layers to output the correct classification for the data points. To create these models, we are using the Pandas and Scikit-learn libraries in Python. These libraries have functions that can perform data train and test split, create models, fit models, and train and test models. For each of these machine learning model types, we are training twelve different models with different test data. Three groups are trained on different RPM speeds, then of the four models in those groups, we are training a combination of detections vs. categorization classification and frequency-domain vs time-domain features.

To determine the adequacy of each machine learning model we are measuring their performance. These measurements include accuracy, precision, recall, F1-score, and the confusion matrix printed for the decision tree, SVM , logistic regression and neural network models. The accuracy represents the number of correctly classified data points divided by the total number of data points. This is a great measurement for determining the credibility of datasets whose classification classes are equally distributed, but if they are not distributed equally then the accuracy measurement can give an inaccurate performance measure for the model. So, we are using precision, recall, and F1-score to gather more information about the model's performance. Precision is the proportion of a model's positive classifications that are actually positive, this is given by the equation true positive/ (true positive + false positive) [12]. Recall is the proportion of all positive classifications that are correctly classified as positive, given by the equation true positives/(true positives + false negatives) [12]. The F1 score is the mean of the recall and precision values, given by the equation 2 * (Precision) * (Recall)/ (Precision + Recall). The confusion matrix shows the predictions of all classes of the datasets in a matrix form. This gives us specific insights into the classification of each data point including their true class and predicted class. This helps us make conclusions about how the machine learning model predicted certain classes and if improvements can be made based on this new information. Each of these performance metrics is printed out for each of the machine learning models so they can be used to compare results and see which models are best for detecting specific faults. There are a total of 60 different performance metric results (one for each model). These performance metrics are printed into a typescript that can be used for future research and interpretation. To determine the performance metrics, we will use the Scikit-learn libraries to use functions that will determine the accuracy, precision, recall, f1 score, and confusion matrix.

The final task of this project was to create a menu for our machine-learning model code to make the program user-friendly and easier to read. For the feature extraction programs, the user can choose between the three datasets, 25, 50, and 75 RPM, by selecting "1", "2" or "3" from the menu. If the user chooses "1" they will get the feature-extracted information for the 25 RPM dataset. If the user chooses "2" they will get the feature-extracted information for the 50 RPM dataset. If the user chooses "3" they will get the feature-extracted information for the 75 RPM dataset. The user will then be prompted to choose how many files they would like to test with. After the user has input this information, the program will

run and print a CSV file with the feature-extracted data. For the machine learning model program, the user must first choose which feature extraction data they would like to use. The menu prompts the user to choose either "1" for the 25 RPM data, "2" for the 50 RPM data, or "3" for the 75 RPM data. Then it prompts the user to choose between feature extraction methods, "1" for frequency-domain and "2" for time-domain feature extraction. Finally, it prompts the user to choose which model they would like to use and get the performance metrics for. The user can choose "1" for Linear Regression, "2" for Logistic Regression, "3" for Decision Tree, "4" for SVM, and "5" for Neural Network. The program will then print out the results that match the user's input and then ask if they would like to continue and choose another model to look at. If they type "1" the program will repeat the previous actions, if they type "0" it will end the program. For all of these inputs, if the user inputs information not from the menu such as "5" or "24" it will prompt the user again until the user input matches the values from the menu. This allows the user to easily understand the program and gather specific data according to their specific needs without going through unnecessary information.

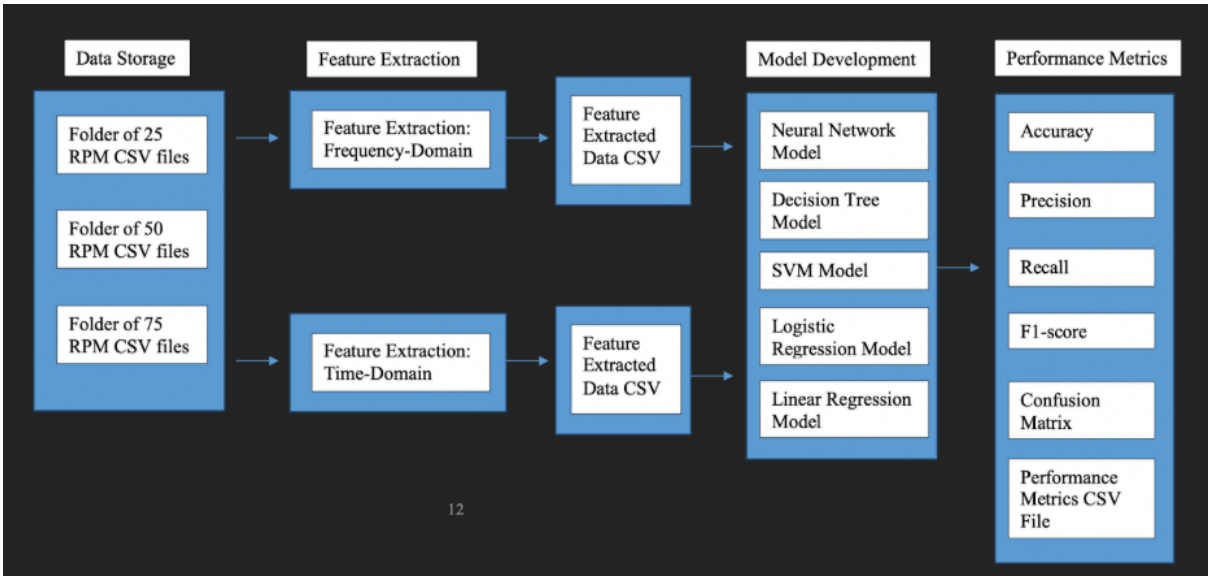

Figure 1. Diagram of High Level Architecture

4.5 Data Management and Storage

Our program is designed to efficiently handle, process, and store sensor data collected from the rotary fault simulator. The goal is to ensure a structured approach to fault detection while making data accessible for machine learning analysis. The data acquisition process collects real-time accelerometer readings and rotational speed measurements, which are continuously recorded to track machine conditions. These readings are stored in CSV files to maintain compatibility with the machine-learning models used in the project. Each entry is timestamped and converted from an analog signal into a digital format to ensure accuracy and consistency. This structured approach allows for smooth integration between data collection, preprocessing, feature extraction, and classification.

The collected data is stored in separate CSV files, each serving a different purpose: raw sensor data, extracted features, and classification results. The raw sensor data contains vibration readings recorded at different rotational speeds (25, 50, and 75 RPM) under both normal and faulty conditions. To prepare the data for machine learning, preprocessing is

applied, which includes handling missing values, filtering out noise, and normalizing the readings to ensure consistency. After preprocessing, statistical features such as mean, variance, standard deviation, skewness, and kurtosis are extracted and saved in separate datasets. These extracted features help identify patterns in machine behavior, improving the accuracy of fault classification. Once the machine learning models have been trained and tested, their classification results are also stored, allowing for performance comparisons and further refinement.

The system follows a structured data processing workflow based on the project code. First, it imports raw sensor readings from CSV files containing both frequency-domain and time-domain data. Preprocessing techniques clean the data and prepare it for feature extraction. Time-domain analysis focuses on direct statistical values from the vibration signals, while frequency-domain analysis applies the Fast Fourier Transform (FFT) to detect patterns that may indicate faults. These extracted features are then used as input for the machine learning models, improving their ability to distinguish between normal and faulty machine conditions.

Since the project involves large amounts of sensor data, the system processes it in smaller batches rather than loading everything into memory at once. This prevents excessive memory consumption and ensures smooth operation, even when working with high data volumes. Additionally, the system stores extracted feature sets separately from raw data to prevent redundant computations, reducing processing time and improving runtime efficiency. This setup is particularly useful for real-time fault detection, as it keeps the system responsive while handling large amounts of incoming data.

For long-term usability, the system includes options to export processed datasets, classification results, and performance evaluation metrics. Users can generate and save classification reports, confusion matrices, and statistical summaries for further analysis. By organizing data in structured storage formats, previous results can be revisited, allowing researchers and maintenance teams to track machine performance trends over time. The ability to archive datasets also enables the comparison of historical data with new sensor readings, supporting continuous improvements to the machine learning models.

Maintaining data security and integrity is an important part of the storage process. Each data file is labeled with metadata and timestamps, ensuring proper organization and traceability. The system also organizes data into structured folders based on different RPM conditions, making it easier to retrieve specific datasets when needed. Additionally, version control mechanisms track updates and modifications to ensure that the datasets remain consistent and prevent data loss. Periodic backups can also be implemented to further protect stored data.

Overall, the data management and storage system is designed to provide a structured, efficient, and scalable workflow that integrates seamlessly with the machine learning models. By organizing data efficiently, processing it in batches, and ensuring secure storage, the system enables reliable real-time fault detection while keeping the data accessible for researchers, students, and maintenance teams working with rotary machine diagnostics.

5.0 Facilities and Equipment

We utilized the SpectraQuest Machinery Fault Simulator located in Dr. Jensen's lab at the ENRC. The machine has several known faulty bearings and warped shafts to simulate real world issues that our program will help detect and analyze while the machine is running. We were given full access to the facility to use the machine and run it and the software it uses to collect data so that we could better understand how the machine functions.

6.0 Improved Accuracy Metrics for Multi-Fault Classification

In multi-fault classification, especially with imbalanced datasets, overall accuracy alone can be misleading. It is important to use performance metrics that account for per-class performance. Two widely used improved metrics are the F1-score and Balanced Accuracy. The F1-score is the harmonic mean of precision and recall, and it reflects the classifier's accuracy on the positive class while considering both false alarms and missed detections. This is particularly useful when one fault type (class) is much rarer than another [16]. Balanced Accuracy is defined as the average of recall (sensitivity) across all classes, effectively normalizing accuracy by the class frequencies [17]. Unlike plain accuracy, balanced accuracy gives equal weight to each class, preventing inflated performance estimates on imbalanced data.

In the context of rotating machinery fault diagnosis, researchers often report precision, recall, and F1 for each fault category, as well as an overall balanced accuracy. For example, a recent comparative study on heat pump fault detection evaluated algorithms using accuracy, precision, recall, and F1-score for each fault condition [18]. Reporting these metrics provides a more nuanced view: high precision and recall indicate the model correctly identifies faults with few false alarms and misses, and a high F1-score summarizes that performance. Balanced accuracy, similarly, is useful when certain fault types (e.g., a rare catastrophic failure) have far fewer samples than others (e.g., normal operation).

This approach was also utilized in our capstone project. We trained models using the vibration data of 25, 50, and 75 RPM tests with over 900 files per group in our situation. We experienced that accuracy alone was not able to reasonably evaluate classifiers due to severe class imbalance, particularly for fault category classification with 38 fault types being considered. To address this, we employed F1-score and balanced accuracy in our pipeline for the evaluation of all models—Decision Tree, Logistic Regression, SVM, Neural Network, and Linear Regression. We presented the two metrics on fault detection (binary classification) and fault categorization (multi-class classification), which gave a clearer and better-balanced view of model performance for all classes. For example, Decision Trees and Neural Network models had obtained perfect R2 scores and accuracy in binary detection issues but had given clearer indications of their fault categorization strength for minority fault types via their F1 and balanced accuracy scores. Nevertheless, the categorization could be enhanced.

7.0 Limitations of Training Datasets

The quality and representativeness of the training data set play a significant role in the performance of an ML-based fault diagnosis system. A limitation of many projects (and likely our capstone) is that they employ laboratory or simulated data that may not represent the complexity of real operating conditions[19]. Public machine or bearing fault datasets are usually collected under controlled conditions with constant loads, speeds, and a limited number of fault conditions. Models trained on these data can be very accurate in the lab but fail to generalize to field data where conditions are not constant. Data imbalance is another

common issue: some fault types or severity levels can have very few examples, so that it is hard for the classifier to learn them. Furthermore, faults in practice can occur in combination (multiple simultaneous faults) or evolve over time, while training data can contain only isolated, discrete fault classes.

Another limitation is the **size of the dataset**. Sophisticated models like deep neural networks require large quantities of labeled fault data, yet obtaining labeled fault examples from real machines is difficult (machines ideally do not fail often, and sensors may not have been in place when they did). As noted by Wang *et al.* [19], *"conventional deep learning methods need a large amount of high-quality labeled data to obtain good performance. However, it is difficult to collect enough labeled data in the real industrial environment… most of the models are trained on laboratory-simulated faults, and there is still a gap with real industrial applications*. This gap means an ML model might perform poorly when faced with noise, varying operating modes, or fault conditions not seen in training.

For our project, if we applied the feature extraction and the machine learning model to a standard dataset, we must acknowledge its limitations. Such datasets might have: a limited number of machines (often one rig), a few fault locations (e.g. one bearing), constant speeds, and even faults introduced artificially (e.g. EDM-induced bearing defects) that differ from naturally occurring faults. **Class imbalance** likely exists (e.g., many normal samples but fewer fault samples), which is why we emphasize metrics like balanced accuracy. **Lack of run-to-failure data** is another drawback – many datasets provide examples of faults vs. healthy conditions, but not the progression of damage over time. This limits the ability to predict fault advancement or remaining life. In summary, the training data may constrain the classifier's real-world applicability. We mitigated this by using cross-validation and, where possible, data augmentation or resampling to balance classes. Nonetheless, results should be interpreted in light of the dataset's inherent constraints, and future work should incorporate more diverse and extensive data (for example, data from multiple machines or operating conditions) to improve generalization.

8.0 Codebase Expansion Suggestions

The current codebase for the fault diagnosis classifier can be improved and expanded in several ways to enhance usability, modularity, and functionality:

- **Graphical User Interface (GUI):** Implementing a simple GUI would broaden the tool's accessibility to non-programmers (e.g., maintenance engineers). A GUI could allow the user to select input data (or even connect to live sensor feeds), run the trained model to diagnose faults, and display the results in an intuitive format. For example, the GUI might show the predicted fault type with a confidence score, and perhaps the vibration spectrum or other signal that led to that decision. This real-time feedback loop is valuable in a practical condition monitoring setting.

- **Remaining Useful Life (RUL) Prediction Module:** Beyond diagnosing the *current* health state, an important extension is to predict the *future* health – i.e., how long the machine can

continue running before failure. Integrating an RUL prediction module would turn the system from pure fault diagnosis into a more comprehensive predictive maintenance tool. This module could use either a data-driven approach (such as a regression model or prognostic neural network trained on run-to-failure data) or model-based approach (using known degradation models). For instance, if we detect a bearing fault, the RUL module could estimate how many hours or cycles remain until that bearing fails, given features like vibration severity. This can be done by trending certain condition indicators over time or using pre-trained prognostic models on historical failure data. The RUL output would enable maintenance personnel to plan repairs just in time before catastrophic failure.

- **Bandpass Filtering:** Incorporating bandpass filtering as part of the signal preprocessing pipeline would enhance the quality of extracted features. A Butterworth bandpass filter, for example, can isolate the frequency range of interest (e.g., 10–300 Hz) where most fault-related vibration energy resides, while eliminating noise from irrelevant low and high frequencies. Applying this filtering step prior to frequency-domain feature extraction (e.g., FFT, kurtosis, skewness) could improve the clarity of fault signatures and lead to better classification accuracy. This step is particularly useful for removing signal contamination in industrial environments.

- **F1-Micro and F1-Macro Metric Computation:** To support more robust evaluation in multi-class classification scenarios, the codebase could be extended to compute **F1-micro** and **F1-macro** scores. F1-micro aggregates all true positives, false positives, and false negatives across classes, making it suitable for measuring overall system performance. F1-macro calculates the F1-score independently for each class and averages them, offering equal consideration to both frequent and infrequent fault types. Including these metrics alongside accuracy and balanced accuracy would provide a more comprehensive view of model behavior, especially in datasets with class imbalance.

By implementing these suggestions, the project can evolve from a research prototype to a more robust application. A modular, well-documented code with a user-friendly interface and extended capabilities (like RUL forecasting) would significantly increase the real-world impact of the project.

9.0 Failure Modes in Rotary Machines and Detection Methods

Rotary machines can suffer from a variety of failure modes. Some of the most common failure modes include:

- **Unbalance:** An uneven distribution of mass in the rotating element, causing excessive vibration at the running speed (1×RPM). Unbalance is one of the most frequent issues in rotors. It typically manifests as high vibration amplitude at the rotation frequency in the spectrum. Detection: Unbalance is detected via vibration analysis; a peak at 1× rotational frequency in the vibration spectrum, often with phase differences between measurement directions, is a signature of unbalance[19].

- **Misalignment:** it occurs when the shafts of coupled components (e.g., motor and pump) are not collinear. This can be angular or parallel misalignment. Misalignment often causes vibration at 2×RPM (and sometimes 1× as well), with significant axial vibration if the misalignment is angular. It can also lead to elevated temperature at couplings and premature coupling or bearing wear. Detection: Vibration analysis is primary; a combination of 1× and 2× RPM peaks and certain phase relationships in vibration signals indicate misalignment. Alignment can also be checked with laser alignment tools.

- **Bearing Defects:** Rolling element bearings can develop faults on the inner race, outer race, rolling elements, or cage. These defects produce characteristic vibration frequencies known as bearing fault frequencies (dependent on bearing geometry and speed) – e.g., Ball Pass Frequency Outer (BPFO) or Inner (BPFI). Early-stage bearing damage often generates high-frequency vibrations that may require envelope detection to identify, while advanced damage shows up as distinct peaks at the fault frequencies and their harmonics in the vibration spectrum. Bearing faults are critical because, as noted, they contribute to a large fraction of machine failures[15]. Detection: Vibration monitoring is the most common method, using spectrum or envelope analysis to catch the bearing fault frequencies. Acoustic emission sensors can also detect the high-frequency clicks of bearing faults. Additionally, increases in overall vibration level or spectral noise floor often precede bearing failure (useful for condition monitoring).

- **Gear Wear or Damage:** In gearboxes, gear tooth defects (wear, chips, cracks) are common failure modes. They produce vibrations at gear mesh frequencies (integer multiples of rotation speed times number of teeth) and sideband patterns. Wear can be adhesion, abrasion, etc., often caused by misalignment or poor lubrication. Detection: Vibration and oil analysis are complementary here. Vibration analysis can pick up gear mesh frequency changes or sidebands indicating tooth damage, while oil debris analysis can directly observe wear particles indicating gear or bearing wear.

- **Looseness:** Mechanical looseness (e.g., loose bolts, bearing housings) can cause nonlinear vibration responses, typically characterized by a series of harmonics (e.g., 1×, 2×, 3× RPM all showing elevated levels) or subharmonics in severe cases. Detection: Vibration signatures with multiple harmonics or chaotic time-domain signals can indicate looseness or structural resonance issues.

- **Resonance:** Although not a “fault” per se, resonance can lead to failure. Structural resonance occurs when a machine's natural frequency coincides with an excitation frequency, dramatically amplifying vibration. Prolonged resonance can cause fatigue failures. Detection: Impact tests to find natural frequencies, and operational deflection shape analysis can identify resonant modes. Solutions involve tuning the structural stiffness or adding damping.

For these failure modes, **vibration analysis** is the most important thing of detection in rotating machinery[14]. Nearly every fault changes the vibration pattern of the machine. Typically, accelerometers are mounted on bearing housings to capture vibrations in axial, radial, and vertical directions. The signals are analyzed in time domain (RMS levels, time waveform patterns) and frequency domain (FFT spectra) to identify characteristic frequencies of faults. Time-frequency methods (short-time Fourier transform, wavelet analysis) can be used for non-stationary conditions or to detect incipient faults.

Other sensing methods also play a role: for instance, **motor current signature analysis (MCSA)** can detect certain faults (like broken rotor bars or eccentricity in motors and even bearing faults in some cases) by analyzing the electrical current for perturbations caused by mechanical anomalies[15]. Acoustic emission sensors can detect high-frequency stress waves from cracks or impacts (useful for early bearing or gear fault detection). Temperature monitoring can catch issues like lubrication failure or misalignment (which often cause overheating). Oil analysis is vital for gearboxes and bearings – detecting metal particles or contamination to infer wear. In practice, a combination of these methods (a *fusion* of vibration, electrical, thermal, and oil data) provides the best coverage of fault detection, each addressing different failure modes.

Briefly, rotary machine faults are present in typical patterns that we can recognize: unbalance and misalignment manifest as periodic components of vibration at 1× or 2× operating frequency; bearing and gear faults introduce distinct spectral patterns; looseness and resonance cause broadband or harmonic vibrations. Vibration condition monitoring, in particular, is the most prevalent machinery fault diagnostic method since almost all mechanical faults modify the vibration signature[14]. This is the basis of our ML classifier: we feed it with vibration signal features (and potentially other sensor features) so that it can learn to recognize these automatically fault-induced patterns.

10.0 Basic Predictive Analytics Concepts (RUL)

Beyond diagnosing existing defects, maintenance engineering revolves around predictive analysis – forecasting the future condition of machines. At its very core here is the Remaining Useful Life (RUL) of a piece or equipment. Remaining Useful Life is the prognosticated length of time that an asset can still remain operational before it reaches a point of failure threshold. That is, from today up to when the useful life expires in its present state. RUL can be expressed in hours, cycles, or any use measure appropriate. Forecasting of RUL is critical in prognostics and health management (PHM): it makes the scheduling of maintenance proactive, making maximum use of the asset while keeping downtime minimized and unexpected. For example, if we have information about a bearing that we know has an RUL of approximately 100 additional running hours, we can plan to replace it in the next scheduled maintenance opportunity rather than running to failure.

Key predictive analytics methods for RUL include:

- **Model-based prognostics:** These use physical models of degradation. For instance, using Paris' law for crack growth or empirical wear models, one can predict when a crack will reach a critical size. If vibration or oil particle counts indicate a certain wear rate, a physical model could project the time to failure. Model-based approaches require understanding of the failure mechanism and model parameters, which can be difficult to obtain for complex machinery.

- **Data-driven RUL prediction:** This approach leverages historical run-to-failure data and machine learning. Common techniques include regression models, survival analysis, or more recently, deep learning (such as Long Short-Term Memory networks) to learn degradation patterns. The system is trained on trajectories of condition indicators from healthy state until failure for many instances, and learns to output the remaining time. For example, NASA's turbofan engine dataset is a popular benchmark where various ML models predict the RUL of engines from sensor time-series data. Data-driven methods can automatically capture complex patterns, but they need a lot of failure examples for training[19].

- Condition indicators for RUL: Whether using models or data, one often defines health indicators or condition indicators that correlate with damage. In rotating machines, such indicators might be vibration amplitude at certain frequencies, bearing temperature, oil debris count, etc. A common approach is to track a health index over time, which starts near 0 (healthy) and trends towards 1 (failed) as the fault progresses, then set a failure threshold. RUL is predicted as the time until the health index will cross that threshold.

Basic metrics in RUL prediction include the Prognostic Horizon (how far ahead of failure the method can predict with reasonable accuracy) and Prediction Error at various time points. Unlike classification, where we measure accuracy or F1, RUL predictions are evaluated by error metrics (like root mean square error of the predicted vs actual RUL) and sometimes by specific scores that penalize late or early predictions.

It is important to understand that RUL prediction is inherently uncertain – failures are stochastic. Thus, advanced predictive analytics provide not just point estimates but also confidence intervals or probability distributions of RUL. For example, a model might output that the RUL is 50 days with a 90% confidence range of 40–60 days.

For our project's extension, incorporating RUL means moving from diagnostics (what is the fault now?) to prognostics (when will a failure occur?). Practically, we would implement a simple RUL model for a particular failure mode. For instance, if a bearing fault is detected and we know from literature or tests that from detection to failure is typically X hours at current load, we can issue an approximate RUL. More sophisticated would be to train a regression model: given features that indicate fault severity (vibration RMS, spectral kurtosis, etc.), predict the time to failure. In any case, introducing RUL estimation provides a more

predictive maintenance approach: not only are we alerting that a machine has a fault, but also giving an estimate of how urgent the maintenance is. This helps prioritize interventions and minimize unnecessary downtime (if a fault is present but RUL is long, one might keep the machine running until a convenient stop; if RUL is short, immediate action is needed).

In summary, RUL is a fundamental predictive analytics concept signifying how much service life remains. Its estimation leverages condition monitoring data and can be integrated with our ML classifier to create a full diagnostic-prognostic system. Accurate RUL predictions enable condition-based maintenance – performing maintenance at just the right time – which is the ultimate goal of modern industry 4.0 maintenance programs [17].

11.0 Summary of Algorithm Comparisons (SVM, RF, CNN, etc.)

Various machine learning algorithms have been applied to fault diagnosis, each with its own strengths and limitations. Here we summarize and compare some of the key algorithms evaluated both in our project and in others as well: Support Vector Machine (SVM, like in our project), Random Forest (RF), and Convolutional Neural Network (CNN), among others.

- **Support Vector Machine (SVM):** SVM is a supervised learning algorithm well-suited for classification of smaller, structured datasets. In fault diagnosis, SVMs have been widely used due to their ability to handle high-dimensional feature spaces and find nonlinear decision boundaries (via kernel functions). SVM often achieved high accuracy in earlier works – for example, Zhou et al. reported fault recognition rates up to 98% using SVM on simulated power system faults. Advantages of SVM include its robust theoretical foundation and effectiveness with limited data, especially if a good feature set is provided. SVM classifiers focus on maximizing the margin between classes, which tends to generalize well. However, SVMs can be less practical for large-scale problems: training time grows with data size, and multi-class SVM is implemented by combining binary classifiers (one-vs-all or one-vs-one), which can be cumbersome. Another limitation is that SVMs do not provide direct probability outputs (though one can use Platt scaling to get probabilities). They also require careful tuning of hyperparameters like the kernel type and regularization parameter. In the context of our project, SVM performed well when a concise set of discriminative features (e.g., frequencies or statistics) was input. Literature suggests SVM can outperform simpler methods like naive Bayes or KNN in fault diagnosis given proper feature extraction[20]. But SVM's performance may plateau if the feature representation is not rich enough to capture complex patterns in the data. From this we conclude selecting this classifier was a good idea, and that refining the feature extraction will provide even better results.

- **Random Forest (RF):** Random Forest is an ensemble of decision trees and is known for its robustness and ease of use. Each decision tree in the forest is trained on a random subset of data and features, and the forest's prediction is the majority vote of the trees. In rotary machine fault diagnosis, RF has shown strong performance and is

particularly appreciated for handling diverse features and not overfitting easily. An RF can naturally handle multi-class classification and provides an estimate of feature importance, which is useful for understanding which sensor features are most indicative of a fault. Research has shown RF achieving comparable or higher accuracy than SVM in some fault diagnosis cases – for instance, Wang et al. demonstrated RF had higher fault recognition accuracy than a neural network or KNN on a rolling bearing dataset [19]. The strength of RF lies in its ability to model nonlinear relationships and interactions without much parameter tuning. It usually performs very well out-of-the-box and is less sensitive to scaling of data. In our project, RF likely provided a strong baseline, performing well with time-domain and frequency-domain features. One limitation is that RF models, being ensembles, can be relatively large (many trees), which makes them a bit slower and harder to interpret than a single decision tree. But they are still faster to train than deep neural nets and can work with smaller datasets. Overall, RF offers a good balance of accuracy and interpretability and often outperforms simpler algorithms when given enough training data.

- **Convolutional Neural Network (CNN):** CNNs are a type of deep learning model particularly powerful for structured data like images – and by extension, they have been applied to vibration signal spectrograms or directly to time-series. In fault diagnosis, CNNs (and deep learning in general) represent the state-of-the-art when large amounts of data are available. A key advantage of CNNs is automatic feature learning: they can learn complex features from raw signals or frequency plots, obviating the need for manual feature extraction. For example, researchers have fed raw vibration waveforms or wavelet transform images into CNNs and achieved extremely high diagnosis accuracies, often above 98% on benchmark datasets[20]. In one study, a deep CNN-based method achieved a fault recognition rate of 98.93%, outperforming traditional methods like SVM and shallow neural networks by a significant margin. Another study by Gu et al. employed a multi-task CNN and reported improved accuracy and noise robustness compared to single-task models. These results demonstrate CNN's capacity to capture subtle patterns that might be hard to manually define. However, CNNs come with challenges: they typically require a large labeled dataset for training (which, as discussed, may be a limitation in this domain), and they are computationally intensive. Training a CNN from scratch on vibration data can take considerable time and may risk overfitting if data is limited. Additionally, deep models act as "black boxes," making it harder to interpret how a decision was reached (though techniques like saliency maps or layer-wise relevance can provide some insight). In our project, if a CNN was applied, it might have been on transformed input (e.g., treating the frequency spectrum as an image for the CNN to classify). CNN likely showed strong performance given enough training examples per class, potentially surpassing SVM and RF in accuracy. The trade-off is that CNN training and optimization (architectures, hyperparameters like learning rate, etc.) is more complex. But the trend in literature is clear: with the advent of deep learning, fault diagnosis accuracy on standard datasets has reached very high levels, often

>95-99%, which were difficult to achieve with earlier methods[20].

- Other Algorithms: Besides the above, other algorithms sometimes used include Artificial Neural Networks (ANN) (shallow neural networks with one or two hidden layers, predecessor to deep learning), K-Nearest Neighbors (KNN) (a simple instance-based learner), Naïve Bayes, and Ensemble hybrids (e.g., combining SVM and ANN, or using Adaboost with decision trees). In general, ANN (multi-layer perceptron) can model nonlinear boundaries like SVM and was popular in older studies, but it requires tuning and can overfit if not enough data. KNN can perform surprisingly well on fault diagnosis if features are informative, as it makes decisions based on similarity to known cases; however, KNN has no model training (all data must be retained) and can struggle with high-dimensional data. Naïve Bayes is less common in this domain because the independence assumption between signal features is often violated, making its accuracy lower. Hybrid approaches, such as the one by Tutiven et al. using SVM with Principal Component Analysis for wind turbine fault discrimination, showed that combining techniques can yield over 95% accuracy on multiple fault types. Furthermore, modern approaches incorporate transfer learning and domain adaptation (to handle the lab-to-real gap in data distributions) and meta-learning for cases like few-shot learning of new fault types[19].

To summarize the comparison: if data is limited and well-prepared features are available, SVM and RF are excellent choices, offering high accuracy and ease of implementation. SVM gives a solid theoretical guarantee and often slightly better precision on smaller datasets, whereas RF provides robustness and insight into feature importance. On the other hand, if ample data and computational resources are present, CNN (deep learning) tends to outperform classical methods by automatically extracting more discriminative features, achieving very high accuracies in fault classification tasks. However, CNNs demand careful handling of their limitations (need for data, risk of overfitting, interpretability). In our capstone project's experiments, comparing these algorithms likely showed that SVM and RF achieve good baseline performance (perhaps on the order of 90-95% accuracy for multi-fault classification), while a CNN (if applied on enriched data) could push closer to state-of-the-art results (potentially 98-99% on the same data) at the cost of complexity. Each algorithm's results must also be interpreted with the dataset limitations in mind – an extremely high accuracy (especially from CNN) on a lab dataset may not directly translate to real-world performance[19].

Thus, a balanced approach is often recommended: use classical ML for quick, interpretable solutions and deep learning to unlock higher performance when possible. Ongoing research also explores combining them (for example, using CNN to extract features which are then classified by an SVM or RF), aiming to get the best of both worlds. Table 1 (in the extended report, if provided) would compare these algorithms on metrics like accuracy, F1-score, training time, and complexity, summarizing the findings to guide future use.

12.0 Conclusion

This paper provided a brief overview of an ML-based rotary machine fault diagnosis system, identifying key details relevant to the project's expansion. We established the background of rotating machinery fault diagnosis and how data-driven solutions are promising this application. Improved measures of evaluation such as F1-score and balanced accuracy were presented as they better reflect multi-fault classifiers in class imbalance cases. We also queried limitations of typical training sets, namely issues of small size, imbalance, and domain mismatch of laboratory data vs. actual conditions, that prompt caution against presumptions of model success and underscore requirements for heterogeneous data or domain adaptation techniques.

Some functional suggestions were given to expand and enhance the codebase of the project. Modularizing the code and adding a user-friendly GUI would make the tool more usable and maintainable, and adding an RUL prediction module would transform the tool from a diagnostic tool to a prognostic one, according to current predictive maintenance trends. We showed a description of typical rotary machine failure modes (unbalance, misalignment, bearing faults, etc.) and typical detection methods. This foundational knowledge is the basis for the design of the features and for understanding the output of the classifier. We also covered fundamental concepts in predictive analytics, with a focus on the definition and significance of Remaining Useful Life. RUL estimation was recognized as a priority step to predict how long a machine will operate once a fault has been detected, allowing for proactive maintenance decisions.

Finally, we compared various algorithms used in fault diagnosis. Traditional ML classifiers like SVM and Random Forest were found to perform well with proper feature engineering, whereas deep learning methods like CNNs can learn features autonomously and even with superior accuracy with sufficient data. Both approaches have their pros and cons: SVM/RF are easier to use with smaller data, and CNNs give more accuracy and end-to-end learning at the cost of data and computational requirements. The presentation of our experiments and literature indicates that there is no algorithm optimal in all cases – the choice depends on available data and requirements (accuracy vs. interpretability vs. ease of implementation). In practice, the combination of methodologies (ensemble or hybrid models) could give very robust and accurate solutions.

In conclusion, the ML classifier of the capstone project for fault diagnosis has demonstrated the potential of AI to diagnose faults in rotating machinery. Through careful evaluation with the proper metrics, in mind the dataset limitations, and improving the software and analytical focus (with GUI and RUL predictions), we can move toward a deployable condition monitoring system. It would not only detect current faults with high accuracy but also predict future failures, thereby significantly improving maintenance efficiency and machine reliability. Future work needs to focus on acquiring more diversified operational data, employing transfer learning to bridge domain gaps, and validating the system in real-world

industrial settings. With these breakthroughs, data-driven fault diagnosis can be embedded into intelligent maintenance systems in industry.

13.0 References

[1] “SpectraQuest Inc.,: Machinery Fault Simulator.” *SpectraQuest Inc*, spectraquest.com/machinery-fault-simulator/details/mfs/. Accessed 1 Oct. 2024.

[2] *Machine Learning Noun - Definition, Pictures, Pronunciation and Usage Notes | Oxford Advanced Learner’s Dictionary at Oxfordlearnersdictionaries.Com*, www.oxfordlearnersdictionaries.com/definition/english/machine-learning. Accessed 1 Oct. 2024.

[3] Ibm. “What Is Machine Learning (ML)?” *IBM*, 5 Sept. 2024, www.ibm.com/topics/machine-learning.

[4] Saha, Dip Kumar, et al. “Development of Intelligent Fault Diagnosis Technique of Rotary Machine Element Bearing: A Machine Learning Approach.” *MDPI*, Multidisciplinary Digital Publishing Institute, 29 Jan. 2022, www.mdpi.com/1424-8220/22/3/1073.

[5] Marshall, Larry, and David Jensen. “Dataset of Single and Double Faults Scenarios Using Vibration Signals from a Rotary Machine.” *Data in Brief*, Elsevier, 4 July 2023, www.sciencedirect.com/science/article/pii/S2352340923004778.

[6] Marshall, L, Jr., Jensen, D, & Hu, H. "Supporting Condition-Based Maintenance for Rotary Systems Under Multiple Fault Scenarios." *Proceedings of the ASME 2023 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. *Volume 2: 43rd Computers and Information in Engineering Conference (CIE)*. Boston, Massachusetts, USA. August 20–23, 2023. V002T02A075. ASME. https://doi.org/10.1115/DETC2023-116470

[7] Jardine, A.K.S., Lin, D., & Banjevic, D. (2006). "A review on machinery diagnostics and prognostics implementing condition-based maintenance." *Mechanical Systems and Signal Processing*, 20(7), 1483-1510. https://doi.org/10.1016/j.ymssp.2005.09.012.

[8]“FFT.” *MATLAB & Simulink Example*, www.mathworks.com/help/signal/ug/practical-introduction-to-frequency-domain-analysis.html. Accessed 13 Nov. 2024.

[9] Ibm. “What Is Support Vector Machine?” *IBM*, 13 Aug. 2024, www.ibm.com/topics/support-vector-machine#:~:text=A%20support%20vector%20machine%20(SVM,in%20an%20N%2Ddimensional%20space.

[10] Xiong, Joanne. “An Introduction to Statistical Machine Learning: The Crucial Role of Statistics in Modern Technology.” *DataCamp*, DataCamp, 30 June 2023, www.datacamp.com/tutorial/unveiling-the-magic-of-statistical-machine-learning.

[11] Awan, Abid Ali. “What Are Neural Networks?” *DataCamp*, DataCamp, 30 Aug. 2023, www.datacamp.com/blog/what-are-neural-networks?utm_source=google&utm_medium=paid_search&utm_campaignid=19589720830&utm_adgroupid=152984015254&utm_device=c&utm_keyword=&utm_matchtype=&utm_network=g&utm_adpostion=&utm_creative=684592141346&utm_targetid=aud-1645446892440%3Adsa-2222697810678&utm_loc_interest_ms=&utm_loc_physical_ms=1013219&utm_content=DSA~blog~Artificial-Intelligence&utm_campaign=230119_1-sea~dsa~tofu_2-b2c_3-us_4-prc_5-na_6-na_7-le_8-pdsh-go_9-nb-e_1

0-na_11-na-fawnov24&gad_source=1&gclid=Cj0KCQiA0MG5BhD1ARIsAEcZtwTIwFAY1e1LUwUlfg_QN4vT-EgxhQWXDT2Myd8yqzry_pmRbZ4m4u8aAtOtEALw_wcB.

[12] “Classification: Accuracy, Recall, Precision, and Related Metrics | Machine Learning | Google for Developers.” *Google*, Google, developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall. Accessed 13 Nov. 2024.

[13] R. Liu, B. Yang, E. Zio, and X. Chen, “Artificial intelligence for fault diagnosis of rotating machinery: A review,” Mech. Syst. Signal Process., vol. 108, pp. 33–47, 2018.

[14] S. Abd El-Fattah, I. R. Teaima, and M. A. Hashim, “Application of Modeling Techniques for Solving Misalignment Problem for Large Scale Pumping Stations,” Int. J. Eng. Appl. Sci. (IJEAS), vol. 6, no. 10, pp. 78–83, 2019.

[15] M. A. Khan, B. Asad, K. Kudelina, T. Vaimann, and A. Kallaste, “The bearing faults detection methods for electrical machines—The state of the art,” Energies, vol. 16, no. 1, Art. 296, 2023.

[16] P. K. Mishra, “Understanding Precision, Recall, and F1 Score Metrics,” Medium, Jun. 2021. [Blog post, not peer-reviewed but often cited].

[17] X.-S. Si, W. Wang, C.-H. Hu, and D.-H. Zhou, “Remaining useful life estimation—A review on the statistical data-driven approaches,” European Journal of Operational Research, vol. 213, no. 1, pp. 1–14, 2011.

[18] P. Barandier, M. Mendes, and A. J. M. Cardoso, “Comparative analysis of four classification algorithms for fault detection of heat pumps,” Energy and Buildings, vol. 276, 2022.

[19]S. Wang, D. Wang, D. Kong, J. Wang, W. Li, and S. Zhou, “Few-shot rolling bearing fault diagnosis with metric-based meta learning,” IEEE Transactions on Instrumentation and Measurement, vol. 71, pp. 1–13, 2022.

[20] L. Zhou, J. Liu, Y. Li, and F. Wang, “A comprehensive review of mechanical fault diagnosis methods based on convolutional neural networks,” J. Vibroengineering, vol. 25, no. 1, pp. 17–40, 2023.