PREDICTING THE REMAINING SERVICE LIFE OF RAILROAD BEARINGS: LEVERAGING MACHINE LEARNING AND ONBOARD SENSOR DATA

A Thesis

by

LEONEL VILLAFRANCA

Submitted in partial fulfillment of the

Requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING

Major Subject: Mechanical Engineering

The University of Texas Rio Grande Valley

July 2022

PREDICTING THE REMAINING SERVICE LIFE OF RAILROAD BEARINGS: LEVERAGING MACHINE LEARNING AND ONBOARD SENSOR DATA

A Thesis
by
LEONEL VILLAFRANCA

COMMITTEE MEMBERS

Dr. Constantine Tarawneh
Co-Chair of Committee

Dr. Mohamadhossein Noruzoliaee
Co-Chair of Committee

Dr. Fatemeh Nazari
Committee Member

Dr. Heinrich Foltz
Committee Member

July 2022

ABSTRACT

Villafranca, Leonel, Predicting The Remaining Service Life Of Railroad Bearings: Leveraging Machine Learning And Onboard Sensor Data. Master of Science in Engineering (MSE), July 2022, 95 pp., 16 tables, 50 figures, 41 references, 27 titles.

Bearing health monitoring is a crucial aspect of the railway industry to avoid human and economic losses due to derailments, unexpected downtime for maintenance, and inefficient performance. The University Transportation Center for Railway Safety (UTCRS) at The University of Texas Rio Grande Valley has made major advancements in developing algorithms and onboard sensor modules to tackle the shortcomings of the spatially dispersed wayside bearing health monitoring systems currently in use in the U.S. rail industry. These sensor modules, which have been extensively tested and validated through both laboratory testing and targeted pilot field tests, can continuously monitor the health of train bearings in terms of their temperature and vibration levels. A data-driven machine learning algorithm is presented, which can unravel the functional relationship between the sensor-acquired bearing health data with several explanatory factors that potentially influence the bearing deterioration. More specifically, a Gradient Boosting Machine (GBM) is trained using the vast amount of data collected at the UTCRS over the course of more than a decade. By strategically combining multiple single decision tree models instead of fitting the best single decision tree, GBM obtains a strong ensemble prediction. The trained GBM is further compared against other statistical methods to test for accuracy in determining the remaining service life of a bearing in terms of the remaining operation mileage.

DEDICATION

This thesis is dedicated to all the people who made it possible for me to reach this milestone. I owe so much to so many people, too many to be mentioned here.
To my parents Leonel Villafranca Garcia and Hilda Castillo Garza, I love you and I am infinitely grateful for your unconditional support and for trusting me to succeed in a new country even when it sometimes meant going weeks without seeing each other. When times were tough you still believed in me. Los amo.

To my sister Hilda Villafranca Castillo, thank you for always being there for me, be it to give a word of advice or simply to listen, thank you for reminding me to take care of myself and for the deep conversations and endless laughs, I love you hermana.

To my daughter Lucinda Isabella Villafranca, I love you so much. When I started this journey, you were barely on your way, now as I'm finishing you just turned two. I dedicated double the effort on this degree because I knew it would allow me to give you a better life, a life where you are free to choose your own path, a life where you would be happy. I want to also thank your mom Andrea Castaneda who loves you so much. Even when things were tough, we always found a way to give you the best. Thank you, Andrea. Lucy, I hope when you're older and you look at your old man's work it inspires you to pursue bigger and better things. Just know that whatever you choose I will always be there for you, with your best interest in my mind. Te amo hija.

ACKNOWLEDGMENTS

I want to thank Dr. Tarawneh for taking me in as an undergrad and later supporting my candidacy to the master's program. I don't think you know just how much your support meant to me. When I was in a tough spot and saw no way out you were always there to lend a helping hand. Thank you for trusting me in pioneering this first-generation project, you have opened a whole new set of possibilities for my professional development. I will take with me your focus on excellence and accountability.

Thank you to Dr. Noruzoliaee, for trusting me in completing this project. Thank you for your patience and guidance. Thank you for reminding me that sometimes in research the course of action must be adjusted, and that is okay. Thank you for your encouragement all along this project and in terms of my professional development.

I want to thank the guys at the center, you all are like a second family to me. I want to specially thank Jesse for providing a place for me to stay when I needed it the most. Thank you also to my best friends Dennis, Alberto and Manuel who were there for me during tough times.

Finally, this study was made possible through funding provided by both the University Transportation Center for Railway Safety (UTCRS) at the University of Texas Rio Grande Valley and the National Science Foundation CREST Center for Multidisciplinary Research Excellence in Cyber-Physical Infrastructure Systems (NSF Award No. 2112650). The opinions expressed in this thesis are solely those of the authors, and do not necessarily represent those of the NSF.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Page

CHAPTER I

BACKGROUND AND INTRODUCTION

1.1 Railroad Industry Overview

Operating over 140,000 miles of track, freight rail represents an $80 billion dollar industry in the United States alone. The U.S. and the Canadian rail network, which extends for an additional 30,000 miles, compose the North American Rail Network. Seven of the railroads operating on this 170,000-mile-long network are classified as “Class 1” which means their operating revenue exceeds $490 million and together represent approximately 94% of the total railroad revenue [1] [2]. Such an extensive network poses complex challenges regarding safe and efficient operations. Among these challenges are derailments and delays due to maintenance, which represent millions of dollars in losses for rail companies, and risk human lives. According to data from the Bureau of Transportation Statistics, there were a total of 1,114 derailments in the year 2020, while a cost-benefit analysis determined that the average cost per derailment in Class 1 railroad mainlines was of $391,479 dollars in 2008 [3] [4].

1.2 Tapered-Roller Bearings and Common Defects

From 2010 to 2020, the Federal Railroad Administration (FRA) reported 65% of axle/bearing related accidents were caused by overheating/defective tapered-roller bearings [5]. Given the economic impact of derailments, there is an incentive to identify defective bearings in active operation to prevent catastrophic failure. Tapered-Roller bearings are a crucial component in freight rail operations; however, they are subjected to harsh operating conditions which make them prone to failure. These bearings are expected to run at speeds of up to 80 mph in addition to carrying the combined weight of the railcar and cargo. Each individual bearing experiences a load of 26 kN or 5,845 lbf when the railcar is empty, which is referred to as 17% load conditions, with this value increasing to 153 kN or 34,400 lbf when fully loaded (100% load conditions) and reaching a maximum of 168 kN or 37,800 lbf if the train is allowed to be overloaded (110% load conditions) [6]. These journal bearings consist of three fundamental elements: rollers, inner rings also called cones, and outer rings also known as cups. Out of these three elements— showcased in Figure 1— the cones and cups are the most susceptible to develop defects such as the spalls shown in Figure 2. These defects cause an increase in friction between the rollers and raceways which in turn increases the bearing’s operating temperature, and if left unattended, may lead to costly malfunctions and dangerous derailments.

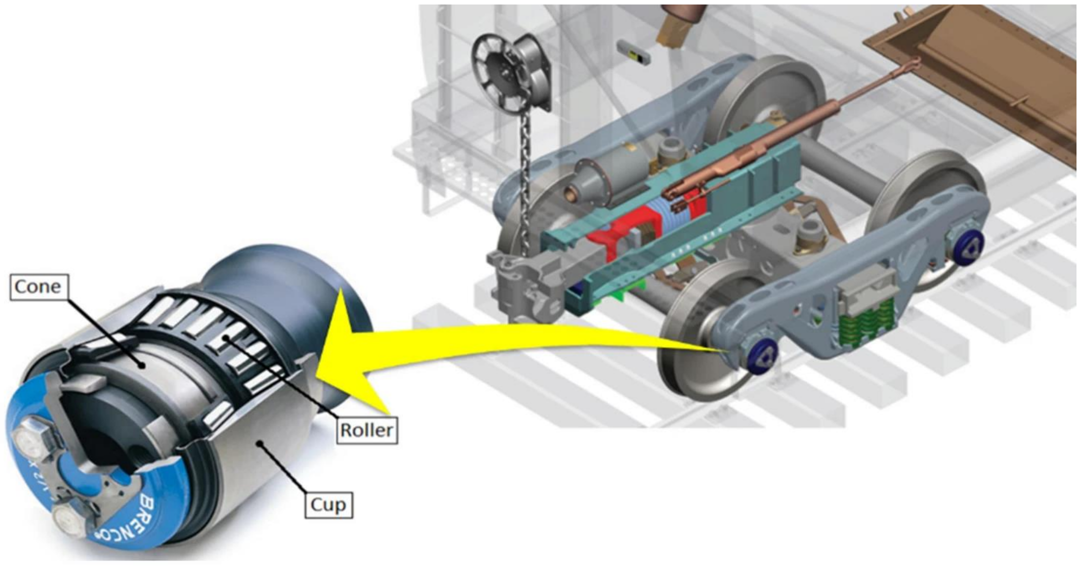


Figure 1: Components of freight railcar tapered-roller bearings. [7].

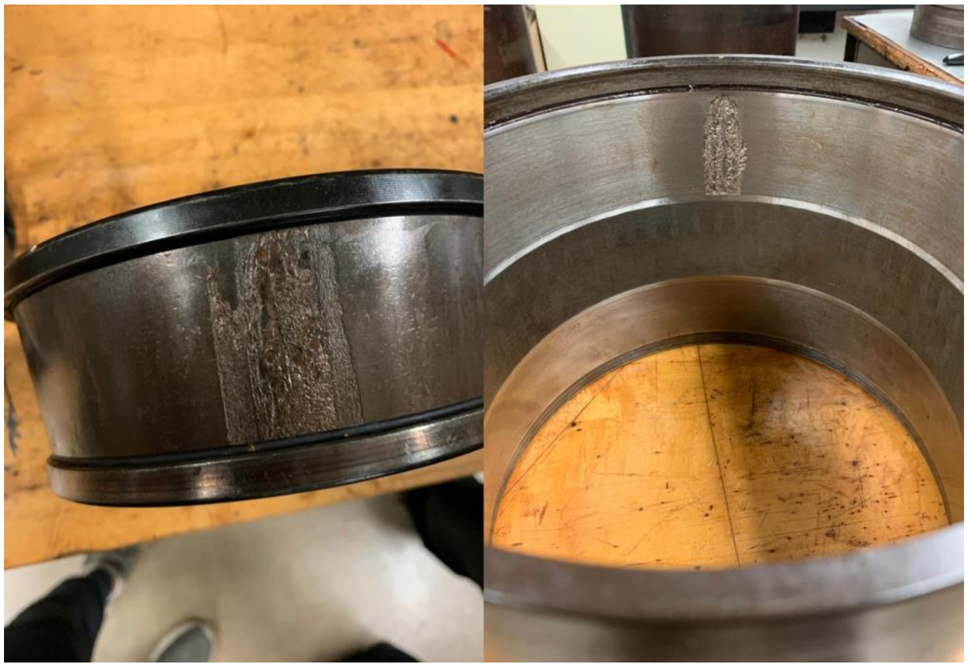


Figure 2: Cone spall (left) and Cup spall (right)

1.3 Wayside Detection Systems

To identify defective bearings in the field, wayside detection systems have been developed, with the two most prominent technologies in use in North America being the Trackside Acoustic Detection System (TADS™) and the Hot-Box Detectors (HBDs) [8]. The TADS™, shown in Figure 3, is geared towards flagging severely defective bearings and works by capturing the acoustic signatures of passing bearings. The acoustic signals are captured using an array of microphones and are then compared against the signals from defective bearings. If the acoustic signal from a bearing matches that of a defective bearing, the bearing is then flagged as potentially defective, and the operator is alerted [8].

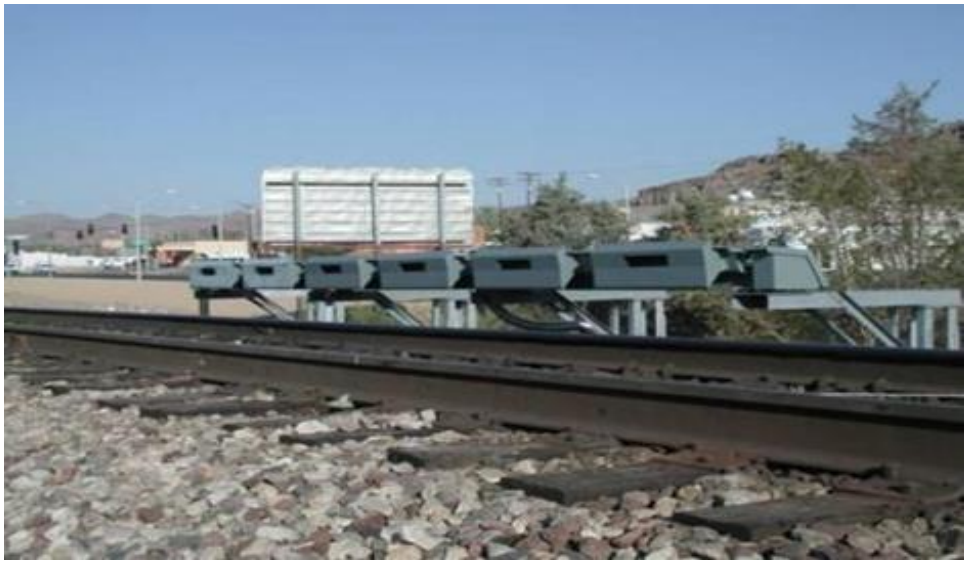


Figure 3: Trackside Acoustic Detector System (TADS™) [8]

In contrast to the TADS™, HBDs, depicted in Figure 4, use infrared sensors to measure the temperature of bearings during operation. The bearing operating temperature is measured as the train passes by these wayside devices, and conductors are alerted when a bearing's temperature is 94.4°C (170°F) above ambient temperature or alternatively if it is 52.8°C (95°F) above the temperature of the mate bearing mounted on the same axle.

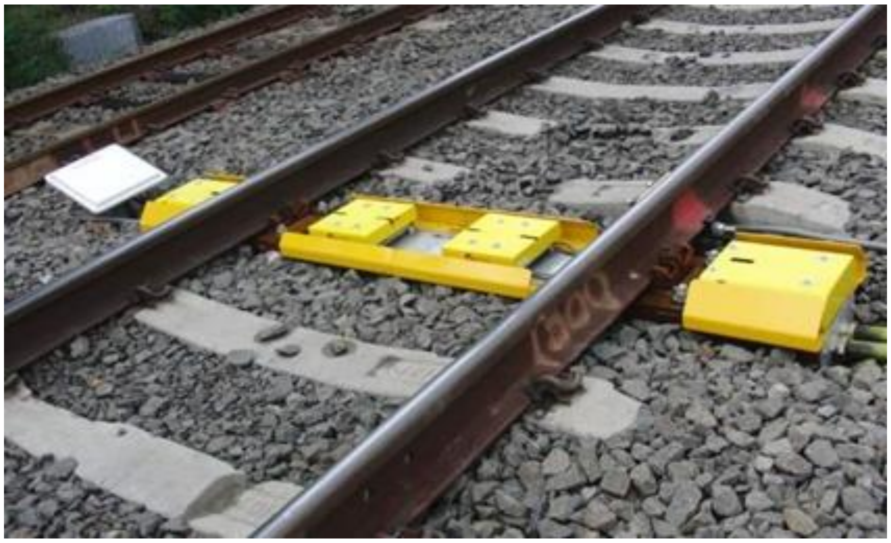


Figure 4: Hot Box Detectors (HBDs) [8]

Although these detection systems have provided favorable results over the years, they remain an unreliable option to detect bearing defects in a timely and accurate manner [6]. There are less than 20 TADS™ stations in service throughout the US and Canada regions combined. Furthermore, TADS™ are not suited to detect certain bearing defects such as those located at the inner cone [8]. While HBDs are far more prevalent in the rail network, with 6,000 units in use in North America, they are still not an optimal solution for defect detection given that some bearings with relatively large defects may still operate at temperatures lower than the established flagging threshold [9].

1.4 Accident Case Study

To highlight the weaknesses in TADS™ and HBD systems, the case study of Canadian Pacific Train 220 is presented. This derailment report is provided by the Transportation Safety Board of Canada which releases detailed information of all derailments in the country. This accident took place on January 26, 2011, with loaded Canadian Pacific Railway freight train 220 traveling southward from Sudbury to Toronto at about 45 mph [10]. As can be appreciated in the map shown in Figure 5, the yellow star represents the point of derailment, and the green triangles represent hot box detectors. No TADS™ stations are present at any point in this route. The derailment took place in a section of the railroad where no hot box detectors were installed. The partial or total lack of both these systems showcases the difficulty in covering a portion of the rail network adequate for timely detection of defective bearings.

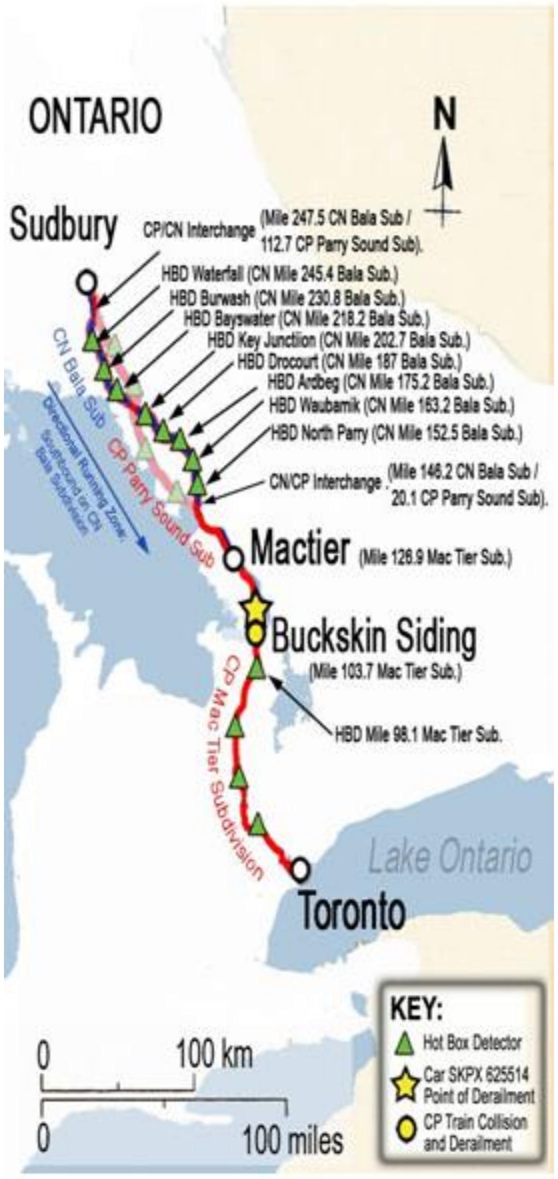

ONTARIO
N
Sudbury
CP/CN Interchange (Mile 247.5 CN Bala Sub / 112.7 CP Parry Sound Sub).
HBD Waterfall (CN Mile 245.4 Bala Sub.)
HBD Burwash (CN Mile 230.8 Bala Sub.)
HBD Bayswater (CN Mile 218.2 Bala Sub.)
HBD Key Junction (CN Mile 202.7 Bala Sub.)
HBD Drocourt (CN Mile 187 Bala Sub.)
HBD Ardbeg (CN Mile 175.2 Bala Sub.)
HBD Waubamik (CN Mile 163.2 Bala Sub.)
HBD North Parry (CN Mile 152.5 Bala Sub.)
CN/CP Interchange (Mile 146.2 CN Bala Sub / 20.1 CP Parry Sound Sub).
CN Bala Sub
Directional Running Zone Southbound on CN Bala Subdivision
CP Parry Sound Sub
Mactier (Mile 126.9 Mac Tier Sub.)
Buckskin Siding (Mile 103.7 Mac Tier Sub.)
HBD Mile 98.1 Mac Tier Sub.
CP Mac Tier Subdivision
Lake Ontario
Toronto
0 100 km
0 100 miles
KEY:
Hot Box Detector
Car SKPX 625514 Point of Derailment
CP Train Collision and Derailment

Figure 5: Map of derailment location of Canadian Pacific train 220 [10]

From this report, it can be appreciated that the impact of derailments is not only seen on the faulty components and the railcar where they are installed but also on adjacent trains, tracks, and the surrounding environment. Figure 6 shows that the damage from this accident was not only seen on derailed train 220 and the tracks where it was traveling but also on train 221 which was hit by the derailed cars while traveling northbound on the siding, clear from the main track. Given the potential spill of the chemical cargo of train 220, an 800-meter exclusion zone was set up and 15 families were evacuated, highlighting how the impact of derailments goes beyond the economical aspect [10].

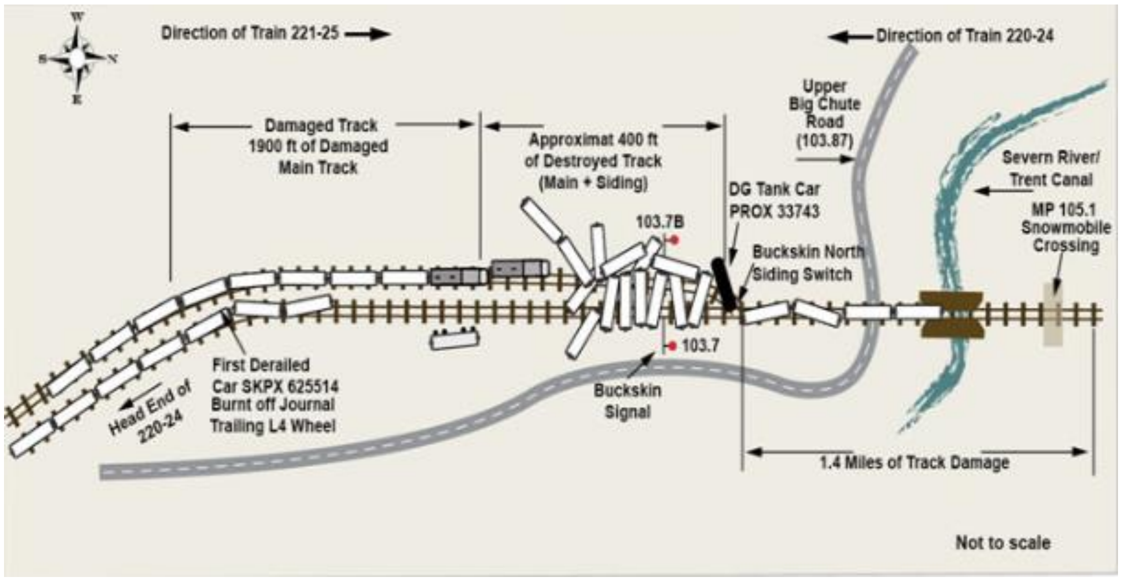

W
S
N
E
Direction of Train 221-25
Direction of Train 220-24
Damaged Track 1900 ft of Damaged Main Track
Approximat 400 ft of Destroyed Track (Main + Siding)
Upper Big Chute Road (103.87)
Severn River/ Trent Canal
MP 105.1 Snowmobile Crossing
103.7B
DG Tank Car PROX 33743
Buckskin North Siding Switch
103.7
Buckskin Signal
First Derailed Car SKPX 625514 Burnt off Journal Trailing L4 Wheel
Head End of 220-24
1.4 Miles of Track Damage
Not to scale

Figure 6: Canadian Pacific train 220 derailment diagram [10]

It was noted on the report for this incident that the defective bearing was on left side of wheelset number 4 in railcar SKPX 625514 referred to as L-4 bearing. Inspection carried out after the derailment showed that this L-4 bearing had a spalled cup raceway as can be appreciated from Figure 7. The report highlights that for four of the last five HBDs that train 220 encountered, the L4 roller bearing on car SKPX 625514 recorded readings that indicated a low-level alert which did not require any action from the conductor [10].

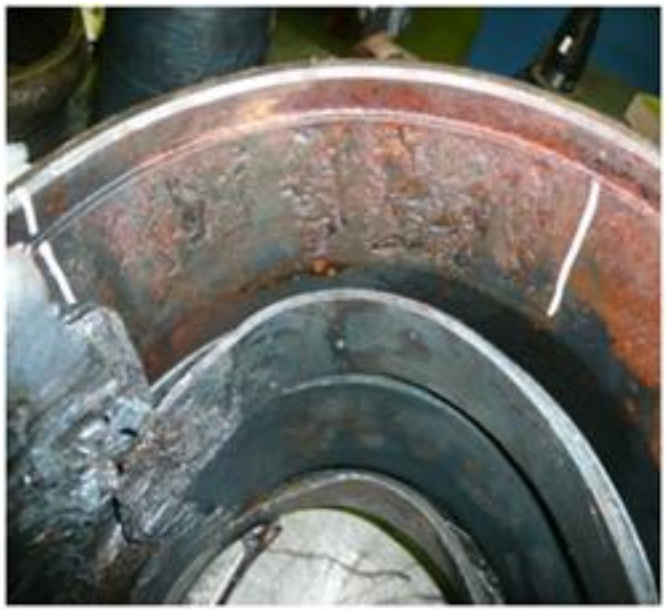


Figure 7: L-4 spalled inboard cup believed to be responsible for derailment [10]

The scarcity of TADS™ stations and their focus on severely defective bearings along with the unreliability of the HBDs alert system, gives place to a significant margin of error where defective bearings can go undetected. Furthermore, the nature of these monitoring systems is reactive, meaning that once a bearing is flagged as defective, operations must be immediately brought to a halt and proper maintenance procedures must be performed to avoid catastrophic failure. A track-obstructing operation, be it due to a derailment or due to maintenance, can cost stakeholders anywhere from $25,000 to $250,000 per hour depending on the location [11]. The challenges associated with these two detection systems highlight the need for the development of alternative methods that can detect defective bearings in an accurate and timely manner.

1.5 On-board Wireless Monitoring System

Having this problem in mind, the research team at the University Transportation Center for Railway Safety (UTCRS) at the University of Texas Rio Grande Valley (UTRGV) has been focused on developing an onboard monitoring sensor which could keep track of a bearing's condition along an entire journey. This onboard monitoring system, shown in Figure 8, provides vibration and temperature readings for a given bearing by using an accelerometer and a thermocouple mounted onto the bearing adapter which has been slightly modified to accept this wireless module [6].

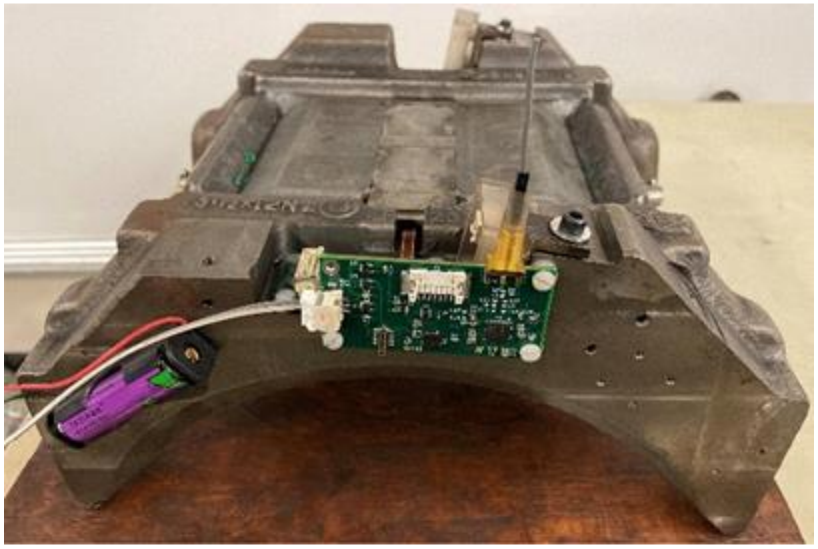


Figure 8: Wireless Onboard Condition Monitoring System powered by a battery pack [6]

Using accelerometers and thermocouples, bearing vibration and temperature data can be accurately measured in a laboratory setting. The team at the UTCRS has been working on this technology for more than a decade, during which vast amounts of data on defective and healthy bearings has been collected.

This study intends to leverage this data archive to develop a machine learning algorithm capable of modeling the future vibration curve of a bearing with the goal of changing the nature of maintenance schedules from reactive to preventive. The modeled vibration curve will provide stakeholders with a prediction of when a bearing's vibration levels will go above an established threshold for safe operation, indicating end of service life. This approach takes advantage of the data-driven nature of machine learning algorithms which strays away from the limitations imposed by traditional statistical models such as the need to know the functional relationship between the response and input variable. More specifically, a Gradient Boosting Regressor and its optimized version, the eXtreme Gradient Boosting Regressor, will be used to perform the predictions. These ensemble-based algorithms combine the predictions of so called “weak learners” to create a “strong learner” to achieve an accurate final prediction [12]. Furthermore, a second project is introduced where these gradient boosting regressors are trained in a privacy-preserving “Federated Learning” framework which allows for multiple parties to simultaneously contribute to the training stage of the model without sharing or disclosing their individual data to any other party involved.

CHAPTER II

MACHINE LEARNING OVERVIEW AND RELATED WORK

2.1 Machine Learning Overview

Machine learning refers to the field of study where computer science and statistics intersect, allowing computational processes to deliver desired results using input data and without the need to be explicitly programmed [13].

Machine learning algorithms can be organized in two main categories: supervised and unsupervised. Supervised learning is used to approximate an unknown function between input and output variables having known input and output samples. In unsupervised learning, only input samples are given to the learning system for it to find trends or other characteristic information on the data (e.g., clustering and estimation of probability density function) [13]. The scope of this project falls under supervised learning and therefore this category will be used to present further discussion on machine learning.

Supervised machine learning algorithms automatically adapt their architecture through repetitive steps which increase their performance, i.e., their ability to achieve a desired outcome, as assessed by a specified metric [13]. This adaptation process is called training, where both input samples and desired outputs are provided for the algorithm to unravel the relation between the two. A key advantage of machine learning algorithms is their optimal self-configuration geared towards providing a desired outcome when presented with new, unseen data. The training phase of the algorithm is the “learning” part of the term machine learning. The training does not have to be limited to an initial adaptation during a finite interval. As with humans, a good algorithm can practice “lifelong” learning as it processes new data and learns from its mistakes [13].

2.1.1 The predictive problem

The focus of this work is on the development of a predictive model. To this end, it is beneficial to establish this goal in mathematical terms to define the steps needed to achieve it and introduce performance indicators to keep track of progress or the lack thereof. This problem of function estimation or “predictive learning” has been widely discussed in the field of machine learning. Definitions of this problem which are highly relevant to this thesis come from the work done by Friedman in 1999 [14] and Natekin and Knoll in 2013 [15].

The problem is framed as having a system consisting of an x and y data space, where x is a set of random “input” or “explanatory” variables $x = \{x_1, \dots, x_n\}$ and y is a random “output” or “response” variable. From this system, “training” samples $\{x_i, y_i\}_1^N$, consisting of known (x, y)-values are used to reconstruct the unknown functional dependence $x \xrightarrow{f} y$, through an estimate $\hat{f}(x)$. The model is then instructed to minimize a specified loss function $L(y, F(x))$, which calculates the discrepancy between actual and predicted values as an indicator of whether the estimate function $\hat{f}(x)$ is a suitable approximation [14] [15]. Equation (1) gives the mathematical representation of this minimization task.

$$\hat{f}(x) = arg \min_{f(x)} L(y, f(x)) \tag{1}$$

A key advantage in the process described up to this point is that no assumptions have been made about the form of the true functional dependence $f(x)$ nor the function estimate $\hat{f}(x)$.

2.1.2 Hyperparameters

There exist several different machine learning algorithms which could be used for predictive tasks, namely, support vector machines, random forest machines, neural networks, and the gradient boosting machine, which is the algorithm of choice for this project, among others. While all these algorithms perform their learning task in a unique way, they all share one fundamental characteristic: hyperparameter values must be assigned before the training phase.

Hyperparameters are parameters inherent to each machine learning algorithm whose values shape the general model structure and have a direct impact on the speed and quality of the learning process. Hyperparameters control the complexity of the model and thus dictate how well the model will be able to unravel the functional dependence between input and output variables. The best results for any model are achieved through optimization of these values which will lead to an adequate balance between bias and variance in the reconstructed functional relationship [16].

2.1.3 Bias and Variance

A prevalent obstacle in training machine learning models is finding the right balance between bias and variance. A biased model will result in an “underfit” of the training data while high variance will cause the model to “overfit” the data, with both instances yielding a model with poor performance on new unseen data. Figure 9 provides an example of these different scenarios.

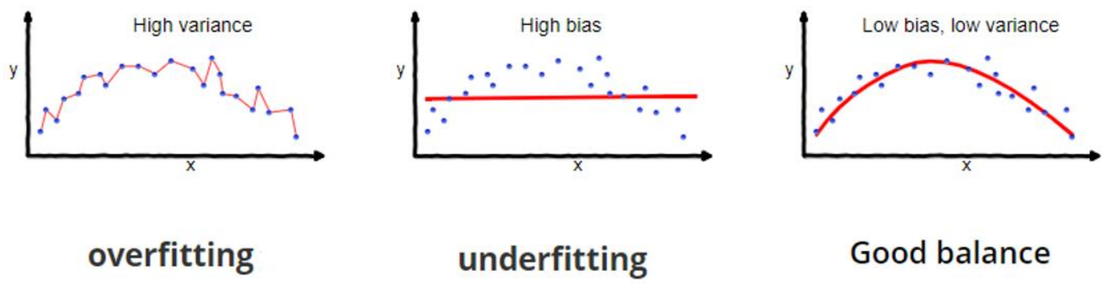


Figure 9: Graphical representation of overfitting, underfitting and optimal fitting [12]

Underfitting happens when a machine learning model cannot effectively unravel the mapping function between input and output variables, with this model type having a poor performance on both the preexisting training data set and newly introduced data. Such a model is said to have high bias and the cause may be attributed to the model generating a functional relationship that is too simple. Another cause for a highly biased model is that the training data is simply insufficient or has such low quality that there is no relation between the input and output variables to be found by the model in the first place. Common solutions to address underfitting problems are increasing the model complexity by tuning hyperparameters to optimal values and increasing the number of input variables or modifying them to reveal more information (feature engineering). Removal of noise from training data, or simply using a different machine learning model may also prove beneficial [17].

Overfitting occurs when a model generates an extremely detailed function mapping input to output variables. A model that overfits the training data is said to have high variance and is characterized by yielding extremely low error rates on the training data set but very high error rates on new data. The issue with using such a complex function for predictive tasks is that this function may be too specific to the training data set and thus will have a poor performance when making predictions using unseen data. Overfitting may occur when a model is too complex, meaning the hyperparameter values set prior to the training phase capture too much detail from the data or even allow the model to interpret noise in the data or inaccurate data entries as relevant information. Another cause for overfitting is a training data set with very few entries or a very simple functional relationship between input and output variables which does not warrant the use of machine learning. Solutions to overfitting include increasing the training data so as to increase the complexity of the input-output functional relationship, reduce model complexity by optimizing or "tuning" hyperparameter values and the addition of regularizing terms which penalize model complexity [17] [18].

2.2 Gradient Boosting Machine

As mentioned in previous sections, there are several machine learning techniques or models available which dictate in which way the learning task will be performed. This thesis focuses on the use of the Gradient Boosting Machine developed by Jerome Friedman [14], specifically Gradient Boosted Decision Trees for a regression task.

Gradient Boosting falls under a category of machine learning models called "ensemble methods" where several models are used simultaneously to reach a single prediction. Ensemble methods come as the counterpart to the more prevalent approach in data-driven modeling where a single strong predictive model is used. The counter-intuitive aspect of this specific ensemble method is that the individual models are—by themselves—weak predictors or "learners" where one would assume that a combination of strong predictors would have a better performance. At each particular iteration, a new weak, base-learner model is trained with respect to the error of the whole ensemble learnt so far, referred to as "boosting" [15]. As mentioned in section 2.1.1, a loss function is used as an indicator of how well a reconstructed functional relationship approximates the true relationship between input and output data. This algorithm employs an iterative optimization process called “gradient descent” to find the local minimum of the specified loss function, explaining the “gradient” term in the name of this machine learning model.

Decision tree models come as a solution to effectively capture the interactions between variables in Gradient Boosting Machine models. The purpose of using decision trees as weak base-learner is to partition the space of input variables into homogenous rectangle areas by a tree-based rule system. Each tree split corresponds to an if-then rule over some input variable. This structure of a decision tree naturally encodes and models the interactions between explanatory variables [15].

2.3 eXtreme Gradient Boosting Machine

Building on top of Dr. Friedman’s work on the Gradient Boosting Machine, the eXtreme Gradient Boosting Machine, also known as XGBoost, was developed by Tianqi Chen [19] as an optimized alternative with better performance and faster computational time. The key differences in the XGBoost algorithm include its ability to build decision trees in a parallel process (as opposed to the traditional sequential) while making use of all the CPU cores during training, using clusters of computers to combine their resources for training, and the ability to automatically handle missing data values through a sparse aware implementation and continued training which allows further model improvement using newly introduced data [20].

2.4 Federated Learning

All of the discussion on machine learning presented up to this point assumes a centralized approach, i.e., the training data is located in a server or machine and can be freely accessed by the model and users. However, there might be instances where parties involved in training a model may not want to or cannot, due to certain regulations or other obligations, share their individual data. For example, a group of hospitals may be interested in training a model which can accurately predict the presence of a given disease (output variable) given a set of symptoms (input variable). Each hospital holds historical data that could be used in conjunction with data from the other hospitals to create an accurate and robust model. The limitation to the collaboration of these hospitals could come from the Health Insurance Portability and Accountability Act of 1996 (HIPAA) which prevents hospitals from sharing a patient's sensitive information without the patient's consent or knowledge [21].

By detaching the ability to train a machine learning model from the need to centralize the training data, a Federated Learning framework can circumvent this limitation. Federated learning works by creating a model for each party involved in the training process so that each party has a distinct model, also called a "local" model. Each party can be thought of as a member of a "federation", thus explaining the name for this framework. The party accepts this local model and trains it utilizing their own data. The updates made to these individual models are then summarized and sent, using encrypted communication, to the federation "administrator" who may or may not have training data and is charged with connecting the individual parties to the federation. The administrator receives the model updates from each party and merges all these updates to modify a "global" model which will ultimately be the model that will benefit all the parties involved. Individual local updates do not directly disclose information on the actual data which, in conjunction with encrypted communication, ensures that training data remains safe and private in the data storage locations of each party [22].

For the purposes of this project, a Federated Learning framework was considered assuming that rail companies may be interested in collaborating with other stakeholders to obtain robust predictive models and benefit from data they may not possess, all while preserving the privacy of their own data.

2.4.1 Federated XGBoost

Federated Learning software is widely available for use with Neural Networks given the nature of these models where each node in the individual layers of the network is automatically assigned a weight during training, with optimal weights giving the best results. These weights do not directly reveal the training data and are thus used as the updates to be sent from the local models to the global model.

Gradient Boosting Machines do not make use of the same architecture as neural networks and, therefore, communicating and merging weights is not an option for this model. Thus, the options to use this algorithm in a federated framework are very limited. Two options were considered for this project: the "Federated XGBoost" library developed by the University of California Berkeley RISE Lab as part of their Multiparty Collaboration and Competition effort "mc^2" [23] and "Federated Learning XGBoost" developed by IBM which comes as a service in their IBM Cloud Pak for Data platform which gives users access to the computational resources owned by IBM [24].

The Federated XGBoost framework developed by RISE Lab required a Linux operating system, which was not readily available at the workstation where this project was conducted. Therefore, IBM services were chosen to carry this federated learning task. It is worthwhile to note that the few available services for federated gradient boosting were based on eXtreme gradient boosting and not its traditional counterpart. This might be attributed to the efficiency in computation and ease of adaptability to different software environments characteristic of XGBoost.

2.5 Related Work

An example of a predictive task performed by using a gradient boosting regressor can be found on the work by Barua et al. [25] where the target variable was the Pavement Condition Index (PCI) which was used to investigate the contributions of several factors on runway and taxiway pavement deterioration at the Chicago O'Hare International Airport. In their study, they show how their developed GBM models outperformed other methods which tend to have strong performances such as neural networks and random forests. An important aspect outlined in this study is their choice for loss function. The "Huber" loss function was chosen over the simpler and more common "square loss function". The justification for this choice comes from the fact that the Huber loss function captures both the square loss and the mean absolute error. To employ both these functions, the Huber loss function uses a multiplier δ, to determine which loss function will be used following Equation (2).

$$L(y, F)_{Huber,\delta} = \begin{cases} \frac{1}{2}(y - F)^2 & if\ |y - F| \leq \delta \\ \delta\left(|y - F| - \frac{\delta}{2}\right) & if\ |y - F| > \delta \end{cases} \tag{2}$$

In this study, a conventional 60-20-20 data split is used to create the training, cross-validation, and testing data sets. While assessing model performance, it was noted that addition of features, in this case the traffic loading variable, proved to be beneficial for the model highlighting the importance of ensuring that relevant features are included in the predictive task of any machine learning model.

An example of ensemble learning used for a prediction task that falls under the field of railway comes from the work performed by Bukhsh et al. [26]. Their work presents the use of tree-based models, namely decision-trees, random forest, and gradient boosted trees to predict the maintenance need, activity type, and trigger status of railway switches. It should be noted that this works presents a classification task as opposed to a regression task, however, the feature processing techniques used provide useful guidelines which could be employed on the project presented as part of this thesis. Part of their feature processing included the elimination of redundant features that would not help the model with its classification task and may even negatively impact it. Removing these unnecessary features also reduces the computational resources needed by the model to perform its task and it comes as a reminder that having more features does not always benefit the model if they are not relevant to the task. Their work showed that out of the three models, gradient boosted trees had the highest accuracy at 86.2% and lowest misclassification rate at 13.7% [26].

Snider and McBean [27] present an eXtreme Gradient Boosting regressor geared towards predicting the time to failure of ductile iron pipes used for water distribution. The performance of the developed XGBoost regressor was compared to a Random Forest and an Artificial Neural Network, which are two machine learning techniques that have provided excellent results in different applications. The obtained root mean square error (RMSE) from the developed XGBoost regressor was of 5.81 which is 1.2% lower than that of the Random Forest model and a 25.9% lower than the one from the Artificial Neural Network [27]. The lower RMSE indicates a stronger performance compared to the other two models as well as a suitable predictive performance for the task at hand.

Li et al. [28] provide a project where machine learning was used to guide maintenance actions in rail operations. In their study, a Support Vector Machine (SVM) was used to analyze large volumes (in the order of Terabytes) of historical data collected from multiple detector systems such as HBDs (previously introduced in this thesis), Wheel Impact Load Detectors (WILDs), Wheel Profile Detectors (WPDs), Hot Wheel Detectors (HWDs), and Truck

Performance Detectors (TPDs). Given the large number of features available to them, they made use of a machine learning based dimensionality reduction technique called Principal Component Analysis (PCA) in order to reduce the number of features by keeping only the ones that provide the most information regarding the mapping function between the features and response variable. Through PCA they were able to reduce the number of features from 55 to 12, which provided a significant reduction in the computational resources needed to develop the model.

The task in this study was to classify train components as either defect-free or defective. Model performance was assessed by determining the true-positive rates (TPR), meaning the percentage of correctly identified defective components, and the false-positive rate (FPR), meaning the percentage of defect-free components classified as defective. Several models were trained and the TPR and FPR was assessed for each one. The highest TPR achieved by any model was of 97.6% along with a FPR of 5.7%. This study assigned high value on achieving a low FPR and therefore the models with the lowest FPR were of interest. The lowest FPR achieved by any model was 0%, however the TPR was only of 7.5%. A satisfactory threshold for FPR was determined to be 0.014%. The model that achieved this FPR threshold had a TPR of 38.5% which, for the purposes of this study, represented a satisfactory balance between the two metrics [28]. It is worth mentioning that although large amounts of data were used to train these models, it was obtained solely from stationary detection systems as opposed to onboard condition monitoring devices, thus, providing an area of opportunity which the work presented in this thesis can address.

CHAPTER III

EXPERIMENTAL SETUP AND PROCEDURES

3.1 Laboratory Setup

Freight operating conditions must be closely replicated in a laboratory setting to obtain relevant information on bearing condition monitoring that could later be leveraged for use in field operations. The team at the UTCRS has developed methods to test bearings and record their vibration and temperature information as response variables while controlling the load to which they are subjected, the speed at which they operate, and in special cases, the ambient temperature that surrounds them. To this end, specially designed dynamic bearing test rigs were built at the UTCRS that utilize the sensor technologies built into the bearing adapter and termed the Smart Adapter.

3.2 Smart Adapter

The Smart Adapter, shown in Figure 10, is a regular AdapterPlus™ that has been machined to accept four custom 70 g ADI ADXL accelerometers placed in the smart adapter (SA) and mote (M) locations at the inboard and outboard sides of the bearing. The Smart Adapter is also retrofitted with one 500 g PCB accelerometer placed in the outboard radial (R) location, and four K-type bayonet thermocouples. Additionally, seven K-type thermocouples are placed around the circumference of the bearing cup which are held tightly in place by a hose clamp [6].

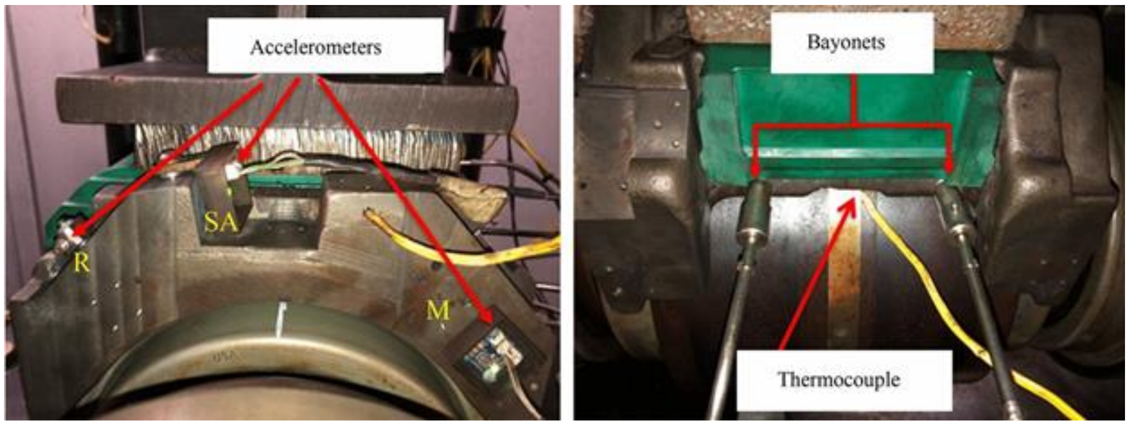


Figure 10: Accelerometer and Thermocouple placement on Smart Adapter

The wired accelerometers and thermocouples provide accurate readings of vibration and temperature, respectively, that can be used to validate the measurements obtained by the wireless module presented in Chapter I. Given that the readings from this wired Smart Adapter monitoring system are reliable and readily available at the UTCRS database, they were chosen as the data source to train and test the machine learning models to be developed.

3.3 Dynamic Bearing Test Rigs

There are two test configurations to run the bearings, the single-bearing tester (SBT) and the four-bearing tester (4BT). Figure 11 showcases the setup for the 4BT which consists of a 22 kW (30 hp) variable speed motor, a pulley system to transfer the energy from the motor to the axle where four bearings are press-fit mounted, two Smart Adapters, a hydraulic cylinder to apply the desired load, and an I-beam used to evenly distribute the load between the two middle bearings. In this configuration, the load applied to all four bearings is equal, however, the two middle bearings are top-loaded exactly like field conditions, whereas the two outer bearings are bottom-loaded. Therefore, Smart Adapters equipped with all sensors are only placed on bearings 2 and 3 (two middle bearings) given that these two bearings are loaded similar to rail service conditions in this current configuration. The 4BT allows the team to collect data from two bearings simultaneously, which provides a more efficient allocation of resources.

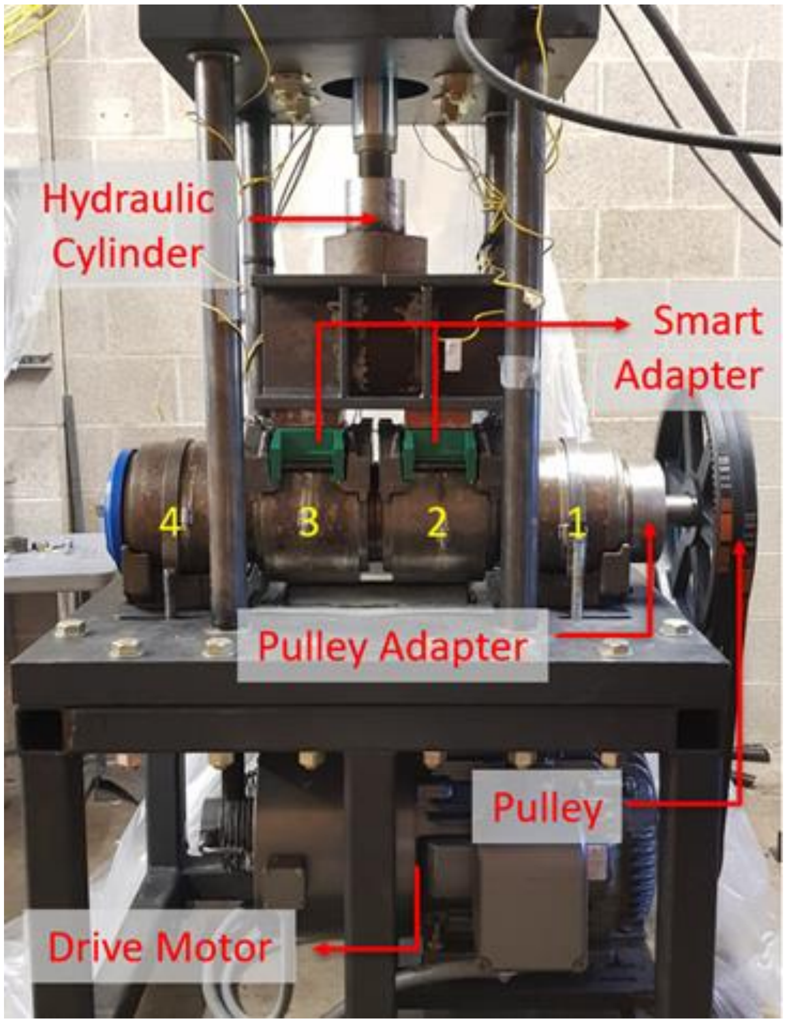


Figure 11: Four-Bearing Tester (4BT)

Note that, for this study, only data acquired from Association of American Railroads (AAR) Class F and K bearings were used, which are the most widely utilized bearings in North American freight rail system (~85% of all bearings in North America are Class F and K). Two main loading conditions were used, namely, fully loaded railcar condition (100% load) which corresponds to 153 kN (34.4 kips) per bearing and empty or unloaded railcar condition (17% load) which corresponds to 26 kN (5.85 kips) per bearing. In some experiments, to simulate an overloaded railcar condition, bearings were loaded to 168 kN (37.8kips) per bearing, which represents. 110% load.

The Single-Bearing Tester (SBT), depicted in Figure 12, was specifically designed to mimic the operating conditions that bearings experience in rail service. The SBT more closely resembles the cantilever beam configuration of a bearing mounted at the end of a wheel axle assembly. Like the 4BT, the SBT consists of a drive motor which can simulate different train traveling speed, a load cell to simulate the bearing load experienced in field operations, and the Smart Adapter which acquires the bearing's vibration and temperature information. Unlike the 4BT, which can only apply vertical loads that simulate the railcar cargo, the SBT can also provide impact and lateral loads which can replicate wheel and rail impacts and forces generated when railcars negotiate turns.

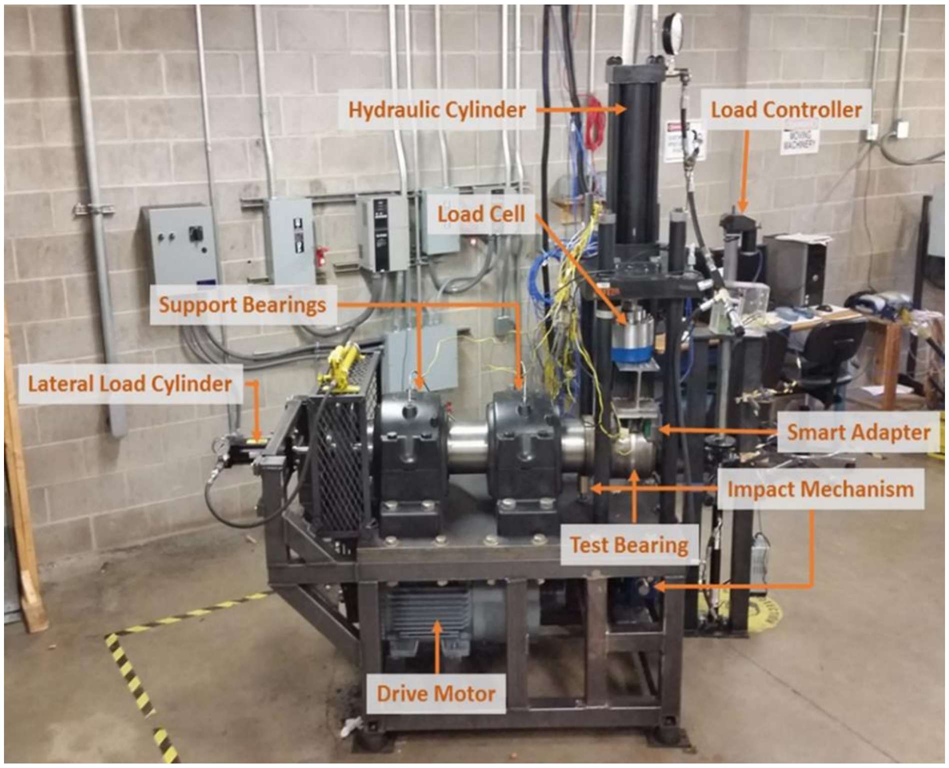


Figure 12: Single-Bearing Tester (SBT)

To replicate field operating conditions more closely, the motors in both test rigs are controlled by variable frequency drives (VFD) that accurately maintain the desired angular speeds to within 0.5%. Furthermore, the bearings are air-cooled by two industrial-size fans that produce an air stream traveling at an average speed of 6 m/s (13.4 mph) simulating the forced convection which train bearings would experience during rail service due to the wind currents as the train makes its journey [6].

3.4 Experiments

The dynamic test rigs were used to perform experiments where healthy or defective bearings had their vibration and temperature profiles monitored. Each experiment was identified by a number, e.g., experiment 150. If there was a need for bearing modifications, such as a component replacement, change of location on the axle, tear down for inspection, then the experiment was considered to have had an iteration. A letter was added to the experiment number to identify said iterations, e.g., experiment 150A, 150B etc. The specified speed and load conditions could vary throughout the duration of the experiment; for example, experiment 150A ran for 1 hour at 30 mph and 17% load (unloaded or empty railcar condition) followed by a segment of 3 hours at 80 mph and 100% load (fully loaded railcar) which gives a total experiment duration of 4 hours.

3.5 Data Acquisition and Thresholds

The data used to train the algorithms came from historical records of bearing vibration levels. This data was collected from experiments performed at the UTCRS on both the single-bearing and the four-bearing tester. The data acquisition equipment consisted of NI 9239, NI USB-6008, and a NI 9234 cards, which were used to record and collect the accelerometer data, measured using the universal gravity constant as a standard denoted as ‘g’, at a sampling rate of

5,120 Hz for four second periods, in ten-minute intervals [6]. Once these data points were collected, their root mean square (RMS) value was calculated, effectively providing a data point every ten minutes. The RMS value was chosen for analysis given the better insight it provides into the magnitude of the signal being monitored, especially one with positive and negative values such as vibration [29]. Equation (3) provides the formula to calculate the RMS value where "N" refers to the total number of data points and "x_n" to the "n-th" data point. It should be noted that the RMS operation does not modify the units of the data introduced.

$$RMS = \sqrt{\frac{{x_0}^2 + {x_1}^2 + {x_2}^2 + \cdots + {x_{N-1}}^2}{N}} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} {x_n}^2} \qquad (3)$$

Data acquisition is a critical step towards gaining insight on defective bearing behavior. However, once data is available it is also crucial to put it into context. To this end, preliminary and maximum thresholds were established by the team at the UTCRS to determine whether the vibration signature of a bearing indicated defective behavior.

In 2015, Gonzalez [30] established these thresholds in her work aimed towards developing a bearing defect detection system. The preliminary threshold was established through a statistical analysis of potential thresholds based on correlations of speed and mean RMS values of defect-free bearing vibration signatures. Table 1 contains all possible thresholds and their corresponding percentages of misclassified bearings. The thresholds came from the μ, μ ± ½ σ, μ ± σ, and upper and lower bounds of the 90%, 95%, and 99% confidence intervals (CI) for the mean RMS, where the confidence intervals represent a proportion of the samples averaged which contain the true mean. The variables μ and σ represent, respectively, the mean and standard deviation of defect-free bearing RMS values for each speed [6]. The optimal preliminary threshold should minimize both the number of defective bearings below the threshold and the number of defect-free bearings above the threshold [6].

Table 1: Percentages of defect-free and defective bearing RMS values that fall outside of possible threshold correlations [30]

Possible threshold	**Defect-free bearing RMS values above possible threshold [%]**	**Defective bearing RMS values below possible threshold [%]**
Upper Bound 99% CI (μ)	13.8	33.2
Upper Bound 95% CI (μ)	19.3	27.4
Upper Bound 90% CI (μ)	21.8	26.6
$\mu + \sigma$	26.8	23.5
$\mu + ½ \sigma$	33.9	20.5
μ	43.8	15.8
$\mu - ½ \sigma$	52.1	11.6
$\mu - \sigma$	61.0	8.5
Lower Bound 90% CI (μ)	65.3	6.9
Lower Bound 95% CI (μ)	72.6	6.2
Lower Bound 99% CI (μ)	87.9	3.5

From the results listed in Table 1, both percentages of misclassified bearings were added together for each potential threshold, with the lowest summation indicating the optimal threshold. Such was the case for the upper bound of the 95% confidence interval and it was thus selected as the preliminary threshold. Equation (4) shows this chosen threshold as a function of speed in different units.

$$\begin{cases} T_p = 7.331 \times 10^{-2}\, V - 9.059 \times 10^{-2} & V \text{ in } km/h \\ T_p = 4.556 \times 10^{-2}\, V - 9.059 \times 10^{-2} & V \text{ in } mph \\ T_p = 4.879 \times 10^{-3}\, V - 9.059 \times 10^{-2} & V \text{ in } rpm \end{cases} \tag{4}$$

The maximum threshold was established such that all bearings with a vibration signature exceeding it are flagged as defective. The maximum threshold was developed by obtaining a correlation between the maximum RMS value of defect-free bearings at different speeds. From this correlation, the upper bound of the 45% confidence interval, plotted in Figure 13, was selected as the maximum threshold given that none of the maximum RMS values of defect-free bearings at each speed value surpassed it, indicating that defect-free bearings would not be misclassified as defective. Equation (5) presents the maximum threshold as a function of speed in various units [6].

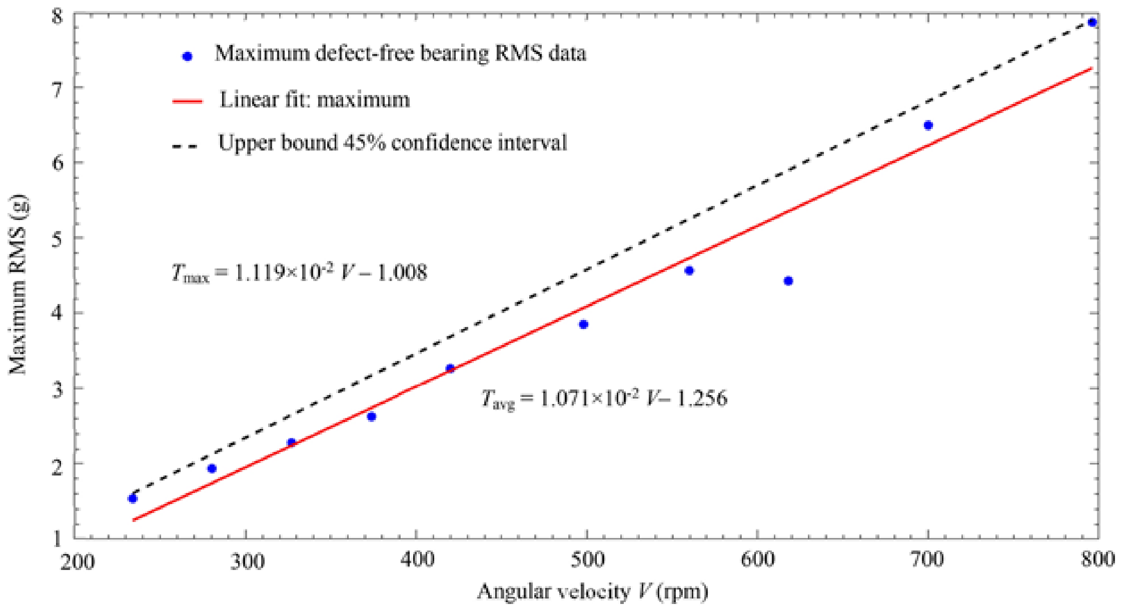


Figure 13: Maximum defect-free bearing RMS values versus velocity [6]

$$\begin{cases} T_{max} = 1.788 \times \ 10^{-1}\,V - 1.008 & V\ in\ km/h \\ T_{max} = 1.111\ \times\ 10^{-1}\,V - 1.008 & V\ in\ mph \\ T_{max} = 1.119\ \times\ 10^{-2}\,V - 1.008 & V\ in\ rpm \end{cases} \tag{5}$$

The development of these thresholds allows for bearings to be classified as potentially defective, defective, or healthy if their vibration levels surpass the preliminary threshold, the maximum threshold, or do not surpass any thresholds, respectively.

3.6 Target Variable

Although all experiments at the UTCRS include the bearing temperature and vibration profiles, it has been noted across multiple experiments that vibration levels usually signal the onset of defect formation before the temperature levels surpass established thresholds. One experiment that showcases this phenomenon is experiment 200, shown in Figure 14. It can be observed from these two plots that the bearing vibration levels reached values that markedly exceed the maximum threshold for healthy bearings while the temperature profiles recorded by the thermocouples remained below the threshold for defective bearings indicating a defect-free bearing. Upon teardown and inspection, the bearing in this experiment was found to have multiple defects, as pictured in Figure 15, that would not have been revealed by temperature measurements alone. Therefore, the vibration levels were established as a more reliable indicator of defect formation or growth, and thus the target variable for prediction was chosen to be the vibration RMS.

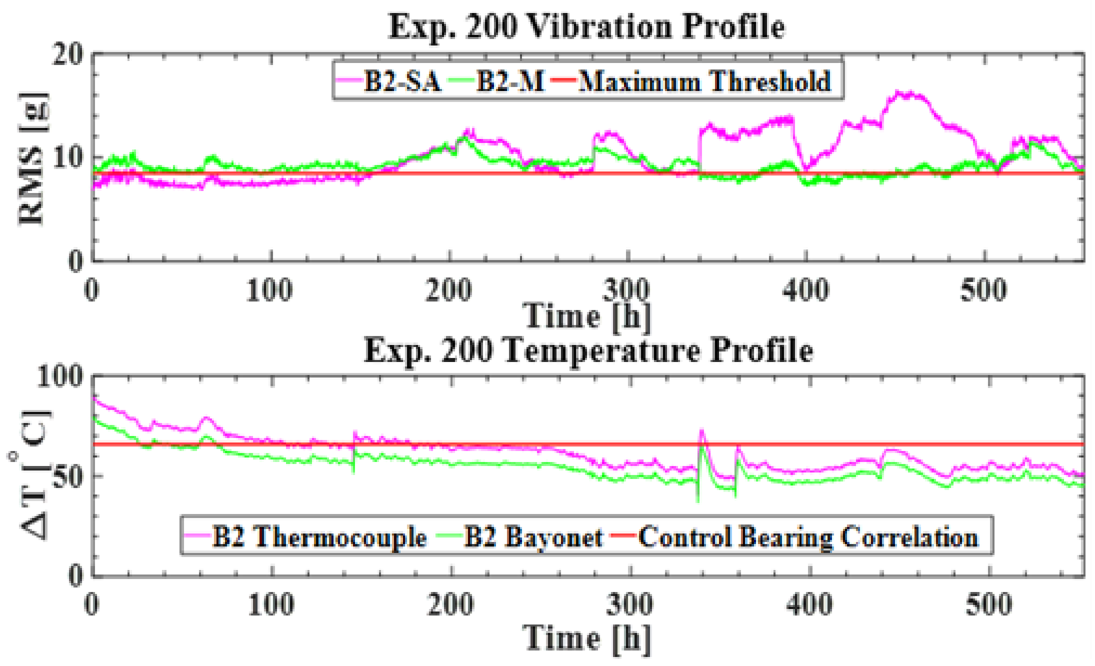


Figure 14: Bearing vibration and temperature profiles for experiment 200

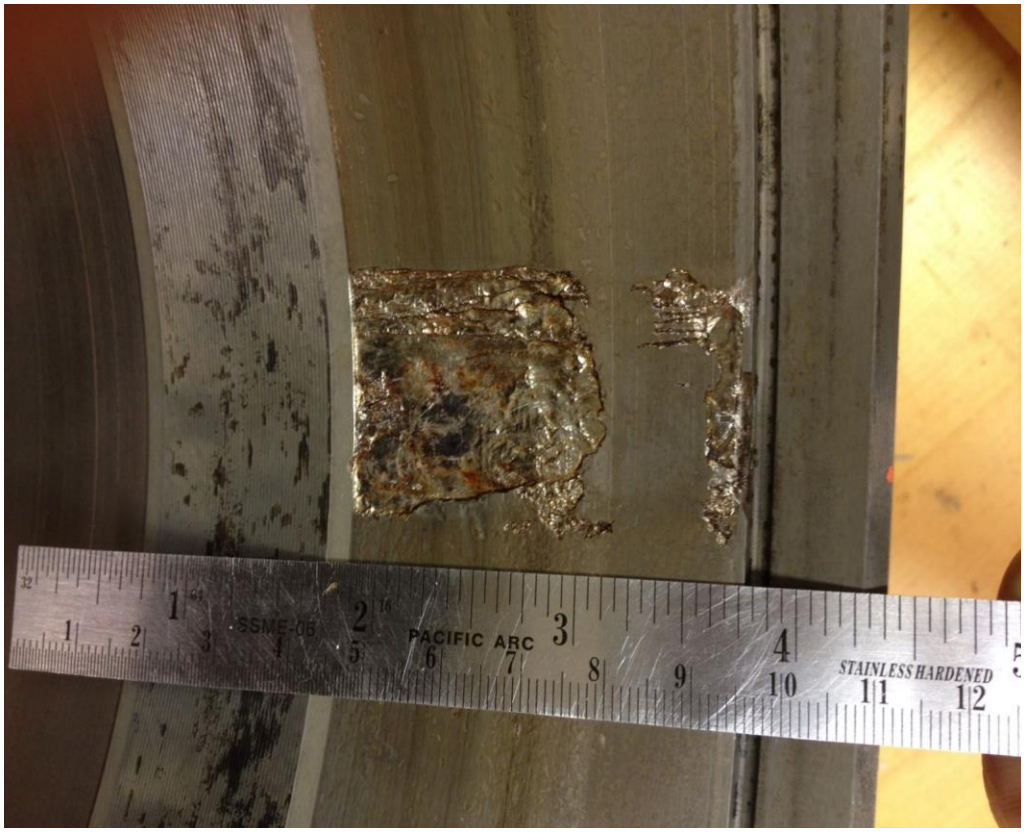


Figure 15: Picture of spall on B2 bearing cup from experiment 200

CHAPTER IV

METHODOLOGY

4.1 Chosen Experiments

All experiments conducted at the UTCRS provide data on RMS vibration levels, however, the total mileage the bearing ran is available for only a select number of experiments. These experiments with full mileage records come from past Service-Life tests where specific bearings were chosen right after manufacturing to be run in a laboratory setting until their vibration levels surpassed the established threshold, indicating defect formation and end of service life. Table 2 contains the identification numbers of the bearings chosen for Service-Life tests along with their respective experiment numbers. The vibration data derived from these experiments was used to train the algorithms.

Table 2: Bearing Identification Numbers and Experiment Numbers in which these bearings ran

Identification Number	**Experiment Number**
C-10-23432	200A, 200B, 200C, 206, 207, 220
C-10-23443	200A, 201A, 201B
H-13-613028	175A, 175B, 175C, 175D, 175E
H-13-613042	175F, 175G, 175H
H-13-613079	175B, 175C, 175D, 175E, 175F, 175G, 175H, 175I, 175J, 175K

Data records from service life tests were obtained for 19 experiments where a total of five bearings were constantly monitored. The information readily available included the RMS histories for all bearings tested. The mileage, speed, and load data were then assigned to their corresponding RMS values by means of simple data processing. A more detailed description of the data preparation is presented in the following section. A total of 53,258 data points with no missing values were obtained and split in a 60-20-20 ratio for the training, cross-validation, and test sets respectively, following convention for data sets with less than 1,000,000 entries [31]. The 60-20-20 split aims at using 60% of the total data for training purposes, once the model is trained, another 20% of the data, the cross-validation data set, is used to assess how well the model is performing. Model improvement is based on the metrics obtained using the cross-validation data set, meaning that the model is further refined until satisfactory metrics are obtained on this set. Once the model is deemed to have achieved its peak performance (i.e., further modifications do not improve metrics), then the final 20% percent of the data, the test data set, is used to assess model performance once again, which provides results that will more closely resemble those obtained when new unseen data is used to make predictions.

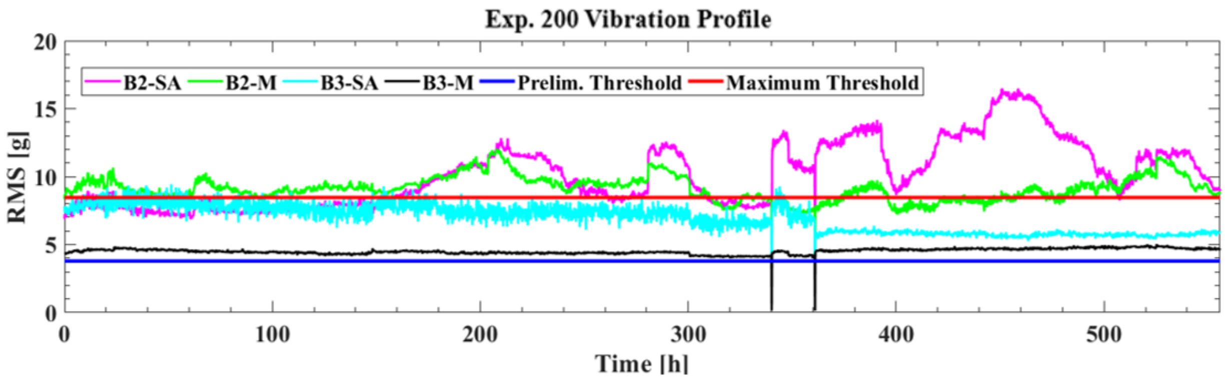


Figure 16: Vibration Profile for Experiment 200A (4BT)

Figure 16 depicts the total RMS history curve for experiment 200A, which is similar to all the available RMS curves for the rest of the experiments considered. It should be noted that this experiment was carried out utilizing the four-bearing tester, which explains why readings for two bearings, bearings B2 and B3, are plotted simultaneously. The legends SA and M denote the accelerometers in the Smart Adapter and Mote positions, respectively, as presented in Chapter 3. Accelerometer readings from the Smart Adapter (SA) position are always recorded in every experiment, while Mote (M) readings were not available for all experiments, therefore, the Smart Adapter vibration readings were chosen as the target variable over the Mote.

4.2 Centralized Gradient Boosting Methodology

The centralized gradient boosting models were developed in Python [32]. The Gradient Boosting Machine comes as one of many available machine learning models in the open source framework for machine learning in Python named "Scikit-Learn" [33], while the eXtreme Gradient Boosting Machine model is part of a separate framework called "XGBoost" [34].

Although vast amounts of data were available at the UTCRS database, there was a need to "preprocess" it, i.e., organize it into a format that could later be used by the algorithm for training and testing purposes [35]. To make this process more manageable, given the large amounts of data to be handled, the data for each experiment was organized in separate excel spreadsheets. Once the data for each experiment was arranged in the appropriate format, it was merged into a single spreadsheet as a comma separated value (csv) file, which is compatible with both the Scikit-Learn and XGBoost frameworks.

The coefficient of determination, R^2, was chosen as the assessment metric for the models developed in this project. This metric is used to determine the ability of a model to predict or explain an outcome in a regression task [36]. The R^2 value is calculated following Equation (6) where $\hat{y}_i$ represents the predicted value, y_i the true value, and $\bar{y}_i$ the mean of all predicted values.

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y}_i)^2} \tag{6}$$

More specifically, R^2 provides the proportion of the variance in the output variable that is explained by the input variable [36]. A good model for the data would have a R^2 value closer to 1. As previously mentioned, the data is split into the training, cross-validation, and test sets. To assess model performance, a R^2 value is obtained for each of these three sets, with an optimal model yielding similar values across all sets.

4.2.1 Data preprocessing

For the purposes of this project, both the features and the target variable had to be arranged as columns in a table. During this process, it was critical to correctly match each RMS value to its respective mileage, speed, and load.

The RMS curves include variations in speed and load all along the duration of each experiment and therefore must be segmented to account for each individual interval where speed and load conditions change. For example, for the first 2 hours of a given experiment (segment 1), the bearing ran at 60 mph and 17% load, then it ran another 2 hours (segment 2) at 85 mph and 100% load. The first thing to consider is that the mileage for these two intervals must be calculated separately given the difference in speeds. Furthermore, separating those two segments allows for the load data to be assigned to the respective RMS values.

To obtain accurate predictions, the total mileage of each bearing had to be considered, i.e., the mileage boundary conditions. The first step in obtaining the mileage boundary conditions was to look at the UTCRS database and identify all the experiments where the bearing of choice was used. Once all experiments were identified, then the mileage for each experiment was either calculated using a MATLAB script or obtained from experiment reports that indicated the number of miles the bearing had operated for, and these values were added up. Once the cumulative mileage prior to the experiment of interest was determined, it was then added to the mileage of the selected interval. Referring to the ongoing example, the total mileage for segment 1 was (60mph × 2hrs) 120 miles, and if the cumulative bearing mileage prior to this experiment was 500,000 miles, then after segment 1, the bearing of interest has run 500,120 miles.

However, having the cumulative mileage and segment mileage was not the final step. Just like speed and load, a value for mileage had to be assigned to each RMS reading. Given the data collection specifications, discussed in Chapter 3, a value for RMS was obtained every 10 minutes, thus, the mileage for each RMS value reflected a change equivalent to the bearing operating for 10 minutes at the specified speed. Going back to the example with segment 1, suppose the bearing of interest had 500,000 miles of operation prior to the experiment, so the first RMS value recorded for the current experiment was assigned a mileage value of 500,010 miles given that at a speed of 60 mph, a 10-minute interval represents 10 miles. This process was performed for all 19 experiments and all data points were merged into a table, which resulted in a total of 53,258 data points. A summary of the data and its organization is showcased in Table 3.

Table 3: Data Summary and Organization

Row Index	**RMS**	**Mileage**	**Speed [mph]**	**Percent Load**
0	4.66	351,038	89.7	100
1	4.60	351,053	89.7	100
2	4.76	351,068	89.7	100
⋮	⋮	⋮	⋮	⋮
202	5.24	352,824	66.5	110
203	5.09	352,835	66.5	110
204	4.81	352,846	66.5	110
⋮	⋮	⋮	⋮	⋮
7,609	16.65	457,806	66.5	17
7,610	16.10	457,817	66.5	17
7,611	15.90	457,828	66.5	17
⋮	⋮	⋮	⋮	⋮
24,696	3.15	25,771	84.5	125
24,697	3.21	25,785	84.5	125
24,698	3.13	25,799	84.5	125
⋮	⋮	⋮	⋮	⋮
53,257	21.75	195,837	86.5	100

4.2.2 Data visualization and encoding

Once the data was consolidated and organized in a suitable format for the algorithm, then the model development process started. The first step was to visualize the data distribution to gain some insights into improving the results. For this purpose, scatter plots were used.

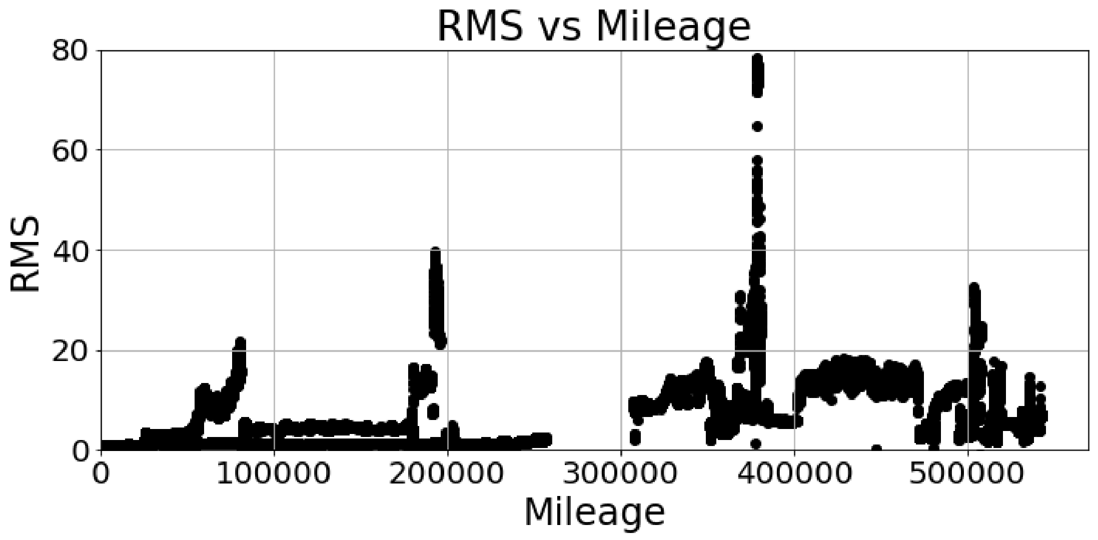


Figure 17: RMS vs Mileage visualization

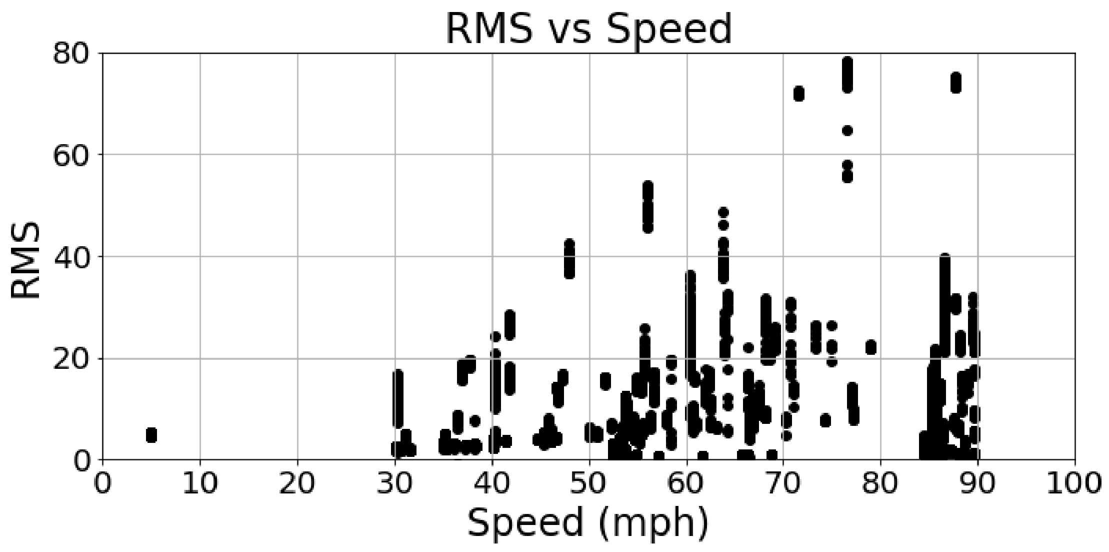


Figure 18: RMS vs Speed visualization

Figure 17 and Figure 18 showcase the complex relation between RMS and mileage and RMS and speed. The complexity of the relation between input and output variables would be an obstacle if traditional statistical methods were being used for the purposes of this project. However, this same functional relation complexity supports the proposed solution of utilizing machine learning algorithms to unravel an appropriate estimate function that can predict RMS values.

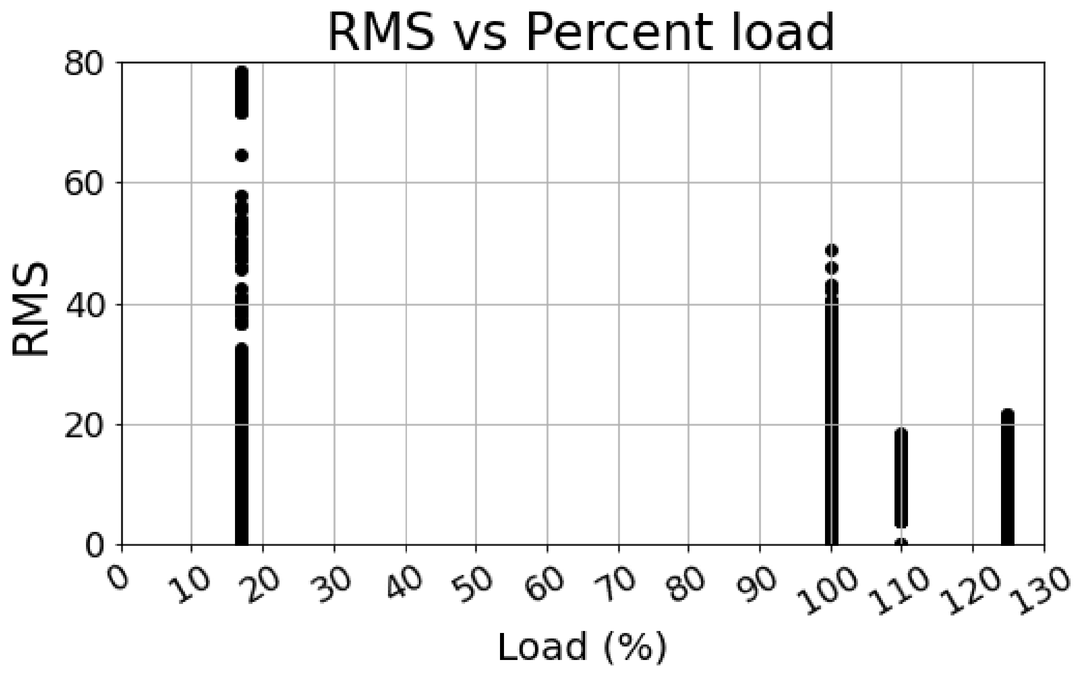


Figure 19: RMS vs Percent load visualization

From the scatter plot depicted in Figure 19, it can be appreciated that load only takes on four values: 17, 100, 110, and 125 percent. The values for load could be considered discrete or "categorical". Certain machine learning models benefit from encoding categorical values in the feature space. Encoding refers to converting variables which contain categories or labels, which are finite in number, into numerical values such that the model extracts useful information from them [37].

"Baseline models" of both the Gradient Boosting Regressor and the XGBoost Regressor, further described in section 4.2.3, were trained with encoded and non-encoded "Percent Load" to determine whether encoding the values for load would influence model performance. Table 4 shows the encoded version of the dataset summary while Table 3 was used for the non-encoded assessment.

Table 4. Data set with encoded values for load

Row Index	**RMS**	**Mileage**	**Speed (mph)**	**Percent Load**	**17**	**100**	**110**	**125**
0	4.66	351,038	89.7	100	0	1	0	0
1	4.6	351,053	89.7	100	0	1	0	0
2	4.76	351,068	89.7	100	0	1	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
202	5.24	352,824	66.5	110	0	0	1	0
203	5.09	352,835	66.5	110	0	0	1	0
204	4.81	352,846	66.5	110	0	0	1	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
7,609	16.65	457,806	66.5	17	1	0	0	0
7,610	16.10	457,817	66.5	17	1	0	0	0
7,611	15.90	457,828	66.5	17	1	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
24,696	3.15	25,771	84.5	125	0	0	0	1
24,697	3.21	25,785	84.5	125	0	0	0	1
24,698	3.13	25,799	84.5	125	0	0	0	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
53,257	21.75	195,837	86.5	100	0	1	0	0

The specific encoding method used for this project is known as “one-hot encoding” which does not assume any hierarchy among categorical values [37]. As shown in Table 4, one-hot encoding makes use of binary variables to indicate the presence of a category, denoted by a 1, and the absence of the other categories by assigning a value of 0 to them. It should be noted that the “Percent Load” column was included in Table 4 for clarity purposes only; in practice, this column is removed from the data set before the training phase to avoid feature redundancy.

Table 5. Effect of encoding in GBR and XGBR models

	Gradient Boosting Regressor (GBR)		**eXtreme Gradient Boosting Regressor (XGBR)**	
Metric	Encoded Percent Load	Non-encoded Percent Load	Encoded Percent Load	Non-encoded Percent Load
Train R^2	0.54	0.54	0.86	0.86
Validation R^2	0.53	0.53	0.77	0.77
Test R^2	0.58	0.58	0.77	0.76

From Table 5, it can be observed that the encoding of the load variable did not influence model performance, highlighting the ability of the Gradient Boosting Machine to aptly deal with categorical data. The model was therefore further trained and optimized with non-encoded values for load given that encoding increases the number of features which in turn increases model complexity and computational requirements, all while providing no improvement to model performance.

4.2.3 Baseline Models

Machine learning algorithms come with pre-defined, or default values for hyperparameters which can be modified by the user as needed. The first step in assessing improvement in model performance was to establish a baseline. To this end, the Gradient Boosting Regressor (GBR) and the eXtreme Gradient Boosting Regressor (XGBR) models were trained using their default hyperparameters and their performance was tested. These initial models are referred to as baseline models. The only hyperparameter modified in these baseline models was the loss function in the GBR which was set to “Huber” as opposed to the default “squared loss”. Table 6 shows the default values for hyperparameters of both models and Table 7 contains the coefficient of determination R^2 values for the preliminary models which are identical to the values from Table 5. From Table 7 it is observed that the GBR has a weak initial performance but R^2 values remain relatively consistent across all three data sets. The XGBR had a much stronger initial performance, however, the score for the training data set is higher than that of the other two data sets. Although the difference may not be large, it could point to a possible, and undesired, overfit of the training data set which may reduce the model performance on unseen data.

Table 6: Default hyperparameter values

Hyperparameter	GBR	XGBR
Learning Rate	0.1	0.3
Maximum Depth	3	6
Number of Estimators	100	600
Minimum Sample of Leaf	1	N/A
Regularization Lambda	N/A	1
Gamma	N/A	0

Table 7: Preliminary model performance

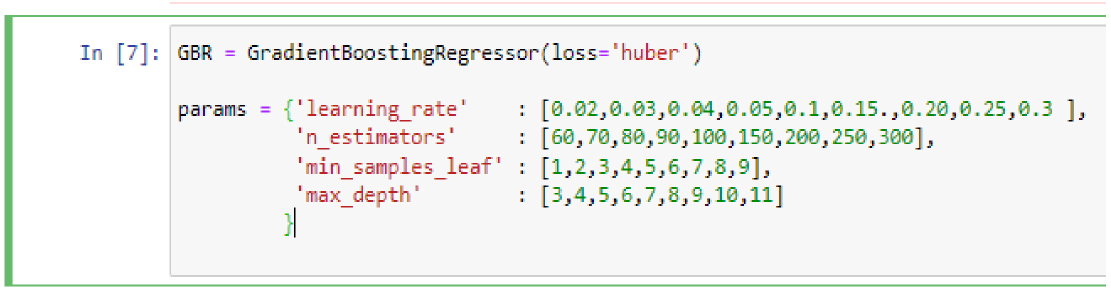
Model	**Data Set**	**R^2 value**
GBR	Train	0.53
	Validation	0.54
	Test	0.58
XGBR	Train	0.86
	Validation	0.77
	Test	0.77

4.2.4 Hyperparameter Tuning

After establishing a baseline for the performance of each model, the next step was to optimize their hyperparameter values. The hyperparameters chosen for optimizing the Gradient Bosting Regressor were “Number of Estimators”, “Learning Rate”, “Maximum Depth” and “Minimum Sample of Leaf”. The Number of Estimators in a Gradient Boosting Algorithm refers to the number of decision trees (weak learners) that will be used to reach a decision, or in this specific case, calculate a value. The Learning Rate lessens the contribution of each decision tree by a specified value, which is important given that Friedman determined empirically that reaching a prediction in smaller steps yielded better results [14]. The Maximum Depth limits the number of nodes in the tree, where the optimal value is dependent on the interaction between the input variables. Deeper trees will increase model complexity and may help the model gather more information on the training data but if the depth is excessive it may lead to overfitting. Lastly, the Minimum Sample of Leaf refers to the minimum number of samples required to be at a terminal or leaf node. A leaf node is the terminal (last) node in a decision tree and contains residual values which will be used to calculate the final predicted value [38].

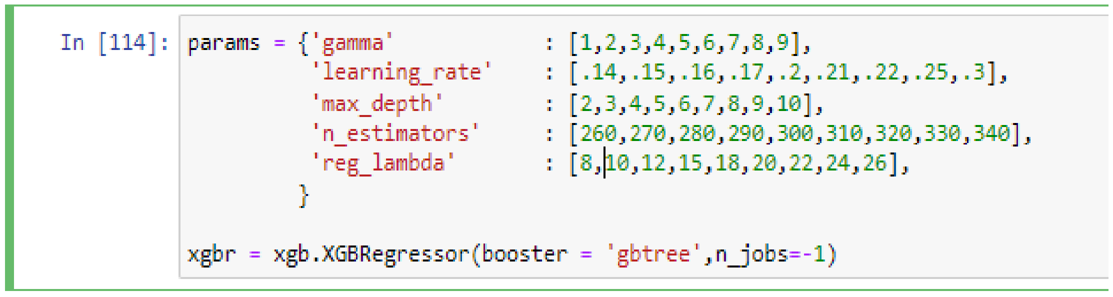
Akin to the GBR tuning, the hyperparameters chosen for optimization of XGBR were “Learning Rate”, “Maximum Depth”, and “Number of Estimators”, while the “Minimum Sample of Leaf” was not a hyperparameter available for modification on this model. Due to its optimized framework, the XGBR can easily overfit training data making it important to look for optimal values for regularization hyperparameters to avoid overfitting, thus “gamma” and “regularization lambda” were also included in the tuning process. The higher the gamma and regularization lambda values, the more conservative the predictions will be, since these hyperparameters penalize model complexity.

“GridSearchCV”, a built-in function included in Scikit-learn, was used for this optimization task. GridSearchCV allows the user to create a “grid” which contains specified values for the hyperparameters targeted for optimization. The function then trains a model for each combination of hyperparameter values, assesses their performance using a specified metric, and outputs the model with the best performance i.e., the highest metric value along with its respective hyperparameter values. Figure 20 and Figure 21 provide sample grids used during this process for each model.



```
In [7]: GBR = GradientBoostingRegressor(loss='huber')

        params = {'learning_rate'    : [0.02,0.03,0.04,0.05,0.1,0.15.,0.20,0.25,0.3 ],
                  'n_estimators'     : [60,70,80,90,100,150,200,250,300],
                  'min_samples_leaf' : [1,2,3,4,5,6,7,8,9],
                  'max_depth'        : [3,4,5,6,7,8,9,10,11]
                 }
```

Figure 20: Hyperparameter grid for GBR



```
In [114]: params = {'gamma'            : [1,2,3,4,5,6,7,8,9],
                     'learning_rate'    : [.14,.15,.16,.17,.2,.21,.22,.25,.3],
                     'max_depth'        : [2,3,4,5,6,7,8,9,10],
                     'n_estimators'     : [260,270,280,290,300,310,320,330,340],
                     'reg_lambda'       : [8,10,12,15,18,20,22,24,26],
                    }

          xgbr = xgb.XGBRegressor(booster = 'gbtree',n_jobs=-1)
```

Figure 21: Hyperparameter grid for XGBR

Nine values were assigned for each hyperparameter with a custom step size (increase or decrease between values) at each grid to ensure a thorough exploration of possible combinations was achieved. The first grid consisted of values which included the default values along with higher and lower values, i.e., the default values were used as the median. Default values were chosen as the median to explore whether increases or decreases in hyperparameter values yielded better results. If one of the optimal hyperparameter values resulted in the maximum or minimum value specified, then that value was taken to be the median in a subsequent grid search with a smaller step size. Six grids containing different values for all hyperparameters to be tuned were used to obtain optimal models and then compare their performances to ultimately choose the one with the best predictive performance. From these six grids, which yielded six different models, the two models with the highest R^2 values were chosen and further assessed. Table 8 and Table 9 show the scores for the two finalist GBR and XGBR models.

Table 8: Finalist GBR model scores

	Value	
Metric	GBR1	GBR2
Train R^2	0.80	0.81
Validation R^2	0.74	0.73
Test R^2	0.74	0.73
Train RMSE	2.68	2.65
Validation RMSE	3.02	3.08
Test RMSE	2.91	2.98

Table 9: Finalist XGBR model scores

	Value	
Metric	XGBR1	XGBR2
Train R^2	0.83	0.84
Validation R^2	0.82	0.82
Test R^2	0.81	0.81
Train RMSE	2.48	2.41
Validation RMSE	2.53	2.54
Test RMSE	2.50	2.51

When comparing the R^2 values from Table 7 to those from Table 8, it can be appreciated that the optimization process had a significant effect on model performance for the GBR. For the XGBR, Table 9 shows that the main impact of the tuning process was achieving uniform R^2 values across all data sets, indicating good generalization.

Given the similarity between R^2 values for finalist models, the Root Mean Square Error (RMSE) was used as an additional metric to assess their performance. The RMSE can be calculated using Equation (7) where N is the number of data points, y(i) is the true value and $\hat{y}(i)$ is the predicted value. This metric provides a measure of how far predictions fall from measured values using Euclidean distance [39].

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \|y(i) - \hat{y}(i)\|^2}{N}} \qquad (7)$$

The addition of RMSE, included in Table 8 and Table 9 as an evaluation metric did not prove helpful in distinguishing model performance given that the error values were also very similar. Therefore, a third metric was utilized to evaluate performance: computational time.

Once the models were deemed accurate and further optimization did not seem to improve metrics, another relevant aspect in judging their performance was the speed at which they produce results, i.e., the models requiring the least computational resources.

Table 10 shows the computational times for the finalist GBR and XGBR models. Note that the specifications of the computer that was used to run these models were as follows: AMD Ryzen Threadripper 3970X 32-Core Processor 3.69 GHz, with 256GB RAM, using a 64-bit Windows operating system.

Table 10: Computational time of finalist GBR and XGBR models

Metric	GBR1	GBR2	XGBR1	XGBR2
Computational Time	26.4 s	1 m 36 s	0.811 s	1.31 s

The XGBR models had a far lower computational time as compared to the GBR, which is expected given that one of the key objectives when developing the XGBoost machine was to significantly improve computation speed [20]. The significant difference in computational time between both finalist GBR and XGBR models, was ideal for the task of differentiating their performance. The lower computational time can be explained when comparing the hyperparameter values shown on Table 11 and Table 12. The fastest models were also the simpler ones, meaning, the number of estimators were lower and the learning rate slightly higher. The fastest GBR and XGBR were selected as the strongest candidates to be used for the predictive task. The predictive performance of both models was compared by using them to predict the RMS curve of specific experiments. These predictions can be found in chapter 5.

Table 11: Hyperparameter values for two best GBR models

Hyperparameter	GBR1	GBR2
Learning Rate	0.01	0.005
Maximum Depth	8	10
Number of Estimators	400	600
Minimum Sample of Leaf	3	1

Table 12: Hyperparameter values for two best XGBR models

Hyperparameter	XGBR1	XGBR2
Learning Rate	0.2	0.22
Maximum Depth	3	3
Number of Estimators	180	290
Regularization Lambda	20	24
Gamma	1	1

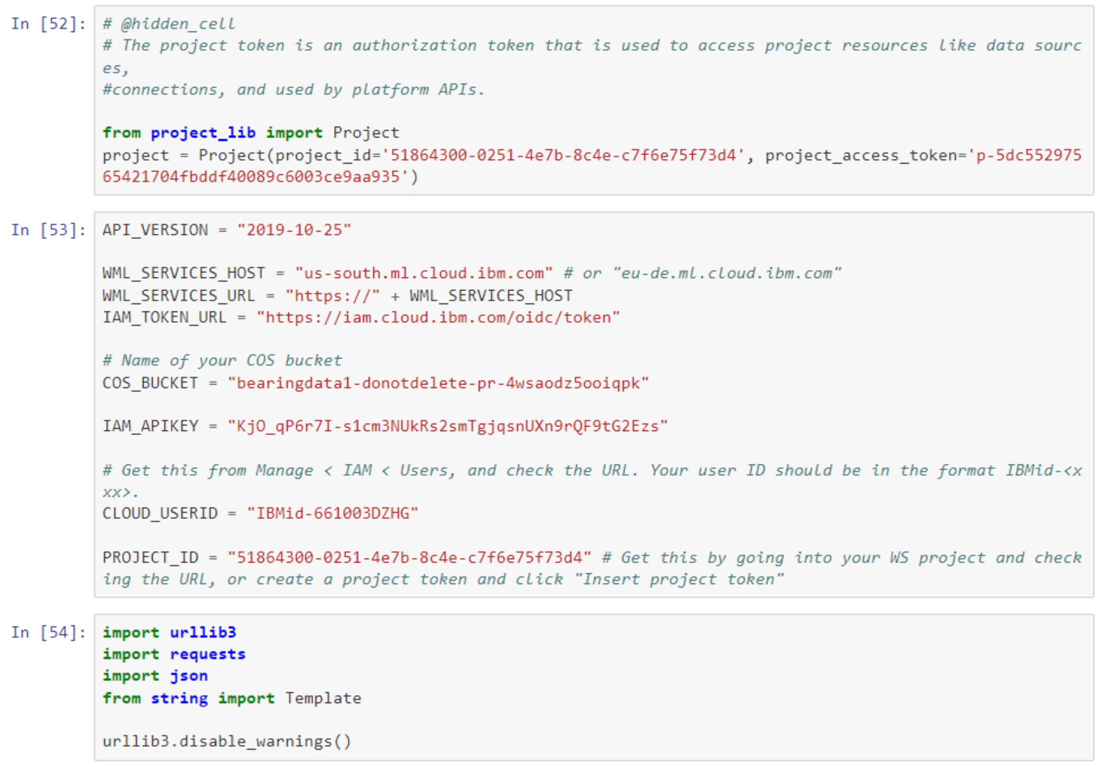
4.3 Federated Gradient Boosting Methodology

The Federated Learning experiment built upon prior work performed for the centralized approach previously described with a few key differences, namely, the split of the data set to simulate multiple parties collaborating with one another, and the use of additional scripts to setup the model.

Given that the purpose of this framework is to preserve the privacy of the data, a simple hypothetical scenario was set up where two distinct clients had their own shares of data and wished to join a federation to obtain a global model that could benefit them both. To this end, the data set presented in section 4.2.1 was split randomly and evenly into two data sets with each client possessing one of them. Each client or party hosted their data set in their own workstation and one of them served as the administrator for the federation.

4.3.1 Administrator and Party Setup

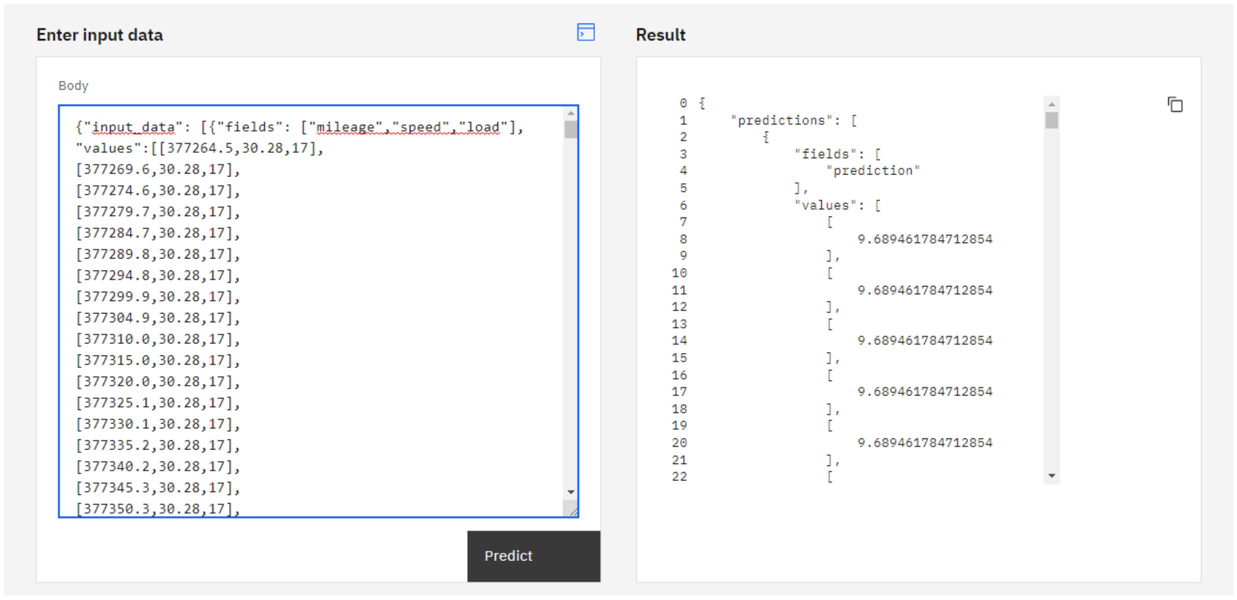
As stated in section 2.4.1, IBM's services were used to develop this Federated Learning model. As part of their services, a tutorial on how to train a Federated XGBoost model for classification was provided. These sample scripts were modified to perform a regression task and then used to carry out the training process. The tutorial included two scripts, one for the administrator and one to be used by each of the parties in the federation which only consisted of two members for this project.



```
In [52]: # @hidden_cell
         # The project token is an authorization token that is used to access project resources like data sourc
         es,
         #connections, and used by platform APIs.

         from project_lib import Project
         project = Project(project_id='51864300-0251-4e7b-8c4e-c7f6e75f73d4', project_access_token='p-5dc552975
         65421704fbddf40089c6003ce9aa935')

In [53]: API_VERSION = "2019-10-25"

         WML_SERVICES_HOST = "us-south.ml.cloud.ibm.com" # or "eu-de.ml.cloud.ibm.com"
         WML_SERVICES_URL = "https://" + WML_SERVICES_HOST
         IAM_TOKEN_URL = "https://iam.cloud.ibm.com/oidc/token"

         # Name of your COS bucket
         COS_BUCKET = "bearingdata1-donotdelete-pr-4wsaodz5ooiqpk"

         IAM_APIKEY = "KjO_qP6r7I-s1cm3NUkRs2smTgjqsnUXn9rQF9tG2Ezs"

         # Get this from Manage < IAM < Users, and check the URL. Your user ID should be in the format IBMid-<x
         xx>.
         CLOUD_USERID = "IBMid-661003DZHG"

         PROJECT_ID = "51864300-0251-4e7b-8c4e-c7f6e75f73d4" # Get this by going into your WS project and check
         ing the URL, or create a project token and click "Insert project token"

In [54]: import urllib3
         import requests
         import json
         from string import Template

         urllib3.disable_warnings()
```

Figure 22: Administrator Script for IBM Federated Learning XGBoost

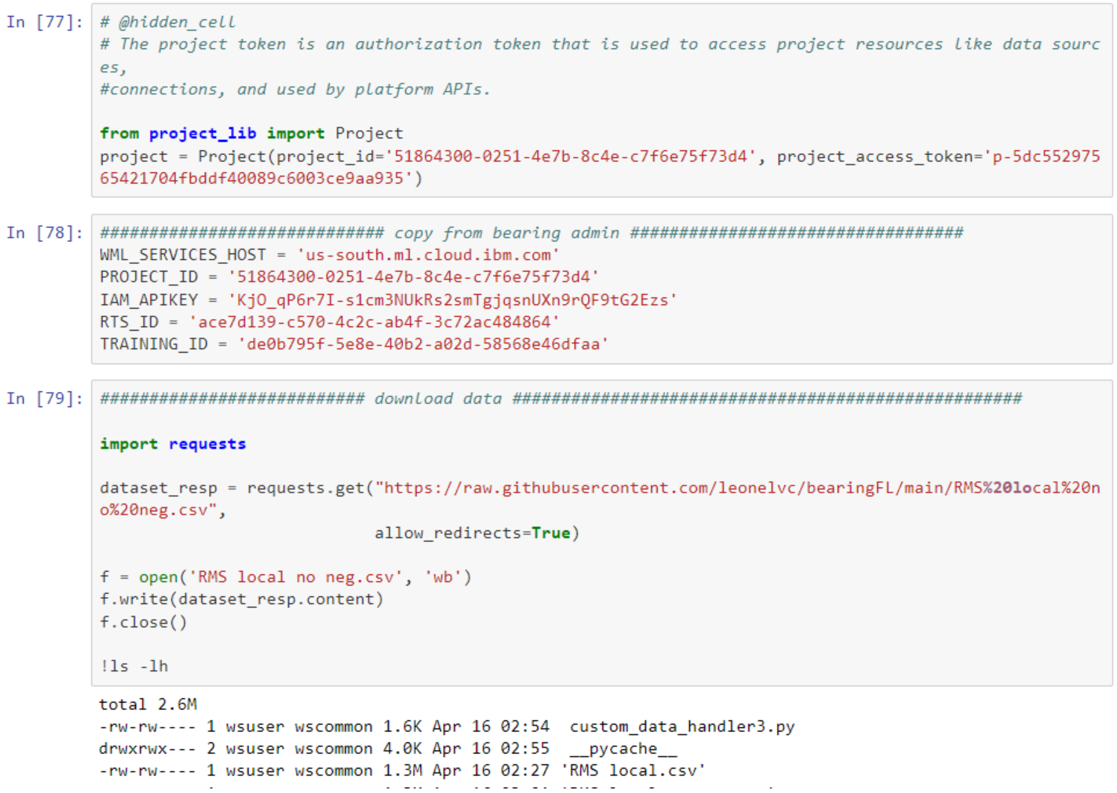
Figure 22 shows a sample of the script for the administrator with the full code available in the Appendix. In this administrator script, a project identification number was assigned along with an access token which ensured that only parties in possession of this token would be able to join the federation. The administrator script also enables the “remote training system”, which allows parties to connect to the federation and send their local model updates to the administrator. The model task, which in this case was set to regression, is defined by the administrator, along with a few hyperparameters that could be customized, namely the “learning rate”, and the “number of rounds”. The number of rounds is a hyperparameter not previously introduced in the centralized approach, given that it is unique to the Federated Learning framework. The number of rounds refers to the number of times each party will train their respective local model and the number of updates sent to the administrator to modify the global model. The administrator script also provides the option to define a loss function, however, IBM’s services currently support only the squared loss function and not the Huber loss function which was used for the centralized GBR. The last portion of this script is geared towards saving the trained model in IBM’s platform, which allows the user to use the trained model to obtain predictions on new unseen data as shown in Figure 23.



```
{"input_data": [{"fields": ["mileage","speed","load"],
"values":[[377264.5,30.28,17],
[377269.6,30.28,17],
[377274.6,30.28,17],
[377279.7,30.28,17],
[377284.7,30.28,17],
[377289.8,30.28,17],
[377294.8,30.28,17],
[377299.9,30.28,17],
[377304.9,30.28,17],
[377310.0,30.28,17],
[377315.0,30.28,17],
[377320.0,30.28,17],
[377325.1,30.28,17],
[377330.1,30.28,17],
[377335.2,30.28,17],
[377340.2,30.28,17],
[377345.3,30.28,17],
[377350.3,30.28,17],
```

```
{
    "predictions": [
        {
            "fields": [
                "prediction"
            ],
            "values": [
                [
                    9.689461784712854
                ],
                [
                    9.689461784712854
                ],
                [
                    9.689461784712854
                ],
                [
                    9.689461784712854
                ],
                [
                    9.689461784712854
                ],
                [
```

Figure 23: IBM's platform for online predictions.



```
In [77]: # @hidden_cell
         # The project token is an authorization token that is used to access project resources like data sourc
         es,
         #connections, and used by platform APIs.

         from project_lib import Project
         project = Project(project_id='51864300-0251-4e7b-8c4e-c7f6e75f73d4', project_access_token='p-5dc552975
         65421704fbddf40089c6003ce9aa935')

In [78]: ############################# copy from bearing admin ##################################
         WML_SERVICES_HOST = 'us-south.ml.cloud.ibm.com'
         PROJECT_ID = '51864300-0251-4e7b-8c4e-c7f6e75f73d4'
         IAM_APIKEY = 'KjO_qP6r7I-s1cm3NUkRs2smTgjqsnUXn9rQF9tG2Ezs'
         RTS_ID = 'ace7d139-c570-4c2c-ab4f-3c72ac484864'
         TRAINING_ID = 'de0b795f-5e8e-40b2-a02d-58568e46dfaa'

In [79]: ########################### download data #####################################################

         import requests

         dataset_resp = requests.get("https://raw.githubusercontent.com/leonelvc/bearingFL/main/RMS%20local%20n
         o%20neg.csv",
                                     allow_redirects=True)

         f = open('RMS local no neg.csv', 'wb')
         f.write(dataset_resp.content)
         f.close()

         !ls -lh

         total 2.6M
         -rw-rw---- 1 wsuser wscommon 1.6K Apr 16 02:54  custom_data_handler3.py
         drwxrwx--- 2 wsuser wscommon 4.0K Apr 16 02:55  __pycache__
         -rw-rw---- 1 wsuser wscommon 1.3M Apr 16 02:27 'RMS local.csv'
```

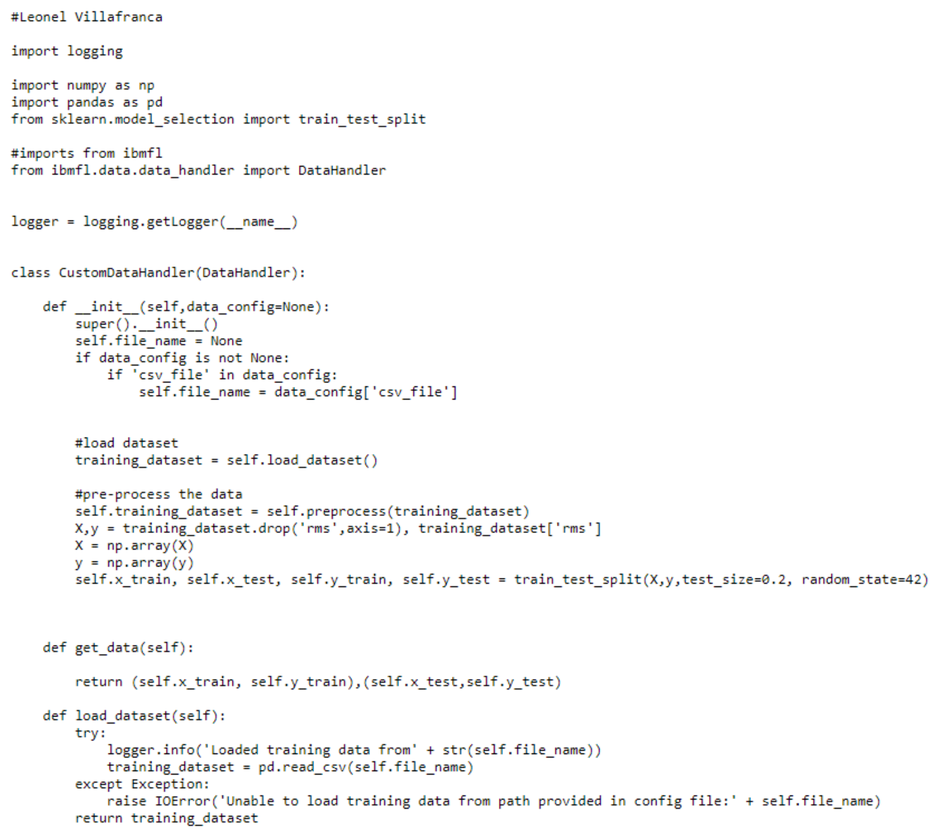
Figure 24: Party Script for IBM Federated Learning XGBoost

Figure 24 contains a sample of the script for the parties with the extended code made available in the Appendix. This script begins by taking inputs from the administrator script, specifically the identifier for the remote training system and the training task, which allows the party to connect with the administrator. Once all parties connect with the administrator the training begins. The following parts consist of commands to indicate where the data can be found in the workstation and the name and location of the “data handler” file which is of fundamental importance in this experiment.

4.3.2 Data Handler

The inclusion of multiple parties in a training task poses unique challenges with one of the most important ones being the difference in data organization. Each party may follow different standards for data acquisition and recording, which presents an issue for the algorithm. Machine learning algorithms are usually programmed to find data in a certain format or location within a structure. For example, in the centralized approach for this project, the data was organized in a table where the RMS values were in the first column. From this table, the RMS column was then removed and assigned to a “y” vector, while the remaining columns were used for the feature space in an “X” array which the algorithm would use for training. If a data file were given to the algorithm where the RMS values existed in a different column other than the first one, then the algorithm would either fail to run or would use incorrect values for the feature space which would lead to a poor performance and inaccurate predictions.

To address this issue, IBM introduced the “data handler” file, which aims at formatting the data from different parties into a common structure to ensure that the correct values for input and output variables are being considered. In addition to providing homogeneity to the training data of each party, the data handler can also take care of feature engineering or data preprocessing tasks. For example, if one of the parties suggests an additional feature to be introduced to the dataset, such as combining two pre-existing features by multiplying their values, then the data handler can be coded to perform this operation for the data belonging to each party. Each party gets a copy of the data handler file, shown in Figure 25 which ideally will be identical, but can be customized for a specific party if there is a special need. The data handler is a Python Class which defines the needed functions. For the scope of this project, the only functions defined in the data handler were those used to identify the file containing the training data, and a few pre-processing steps that involved dividing the features and output variables into "X" and "y" arrays and splitting the data into training and test sets.



```
#Leonel Villafranca

import logging

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

#imports from ibmfl
from ibmfl.data.data_handler import DataHandler


logger = logging.getLogger(__name__)


class CustomDataHandler(DataHandler):

    def __init__(self,data_config=None):
        super().__init__()
        self.file_name = None
        if data_config is not None:
            if 'csv_file' in data_config:
                self.file_name = data_config['csv_file']


        #load dataset
        training_dataset = self.load_dataset()

        #pre-process the data
        self.training_dataset = self.preprocess(training_dataset)
        X,y = training_dataset.drop('rms',axis=1), training_dataset['rms']
        X = np.array(X)
        y = np.array(y)
        self.x_train, self.x_test, self.y_train, self.y_test = train_test_split(X,y,test_size=0.2, random_state=42)


    def get_data(self):

        return (self.x_train, self.y_train),(self.x_test,self.y_test)

    def load_dataset(self):
        try:
            logger.info('Loaded training data from' + str(self.file_name))
            training_dataset = pd.read_csv(self.file_name)
        except Exception:
            raise IOError('Unable to load training data from path provided in config file:' + self.file_name)
        return training_dataset
```

Figure 25: Data handler file

CHAPTER V

RESULTS AND DISCUSSION

5.1 Centralized Gradient Boosting

After the hyperparameter tuning process was completed, the performance of the finalist GBR and the XGBR models was assessed using the coefficient of determination R^2 and the Root Mean Square Error (RMSE). Figure 26 to Figure 29 show scatter plots of the predicted versus actual RMS values for the training, validation, and testing data sets for all models along with their respective R^2 and RMSE values. It should be noted that the red line on these plots is a "perfect fit" line and not a "best fit" line. A perfect fit is obtained when all predicted values are identical to their corresponding actual values, while a best fit line is obtained through simple linear regression and has the best fit between all plotted values. A higher concentration of values falling on the perfect fit line indicates the superior performance of a model over its counterpart.

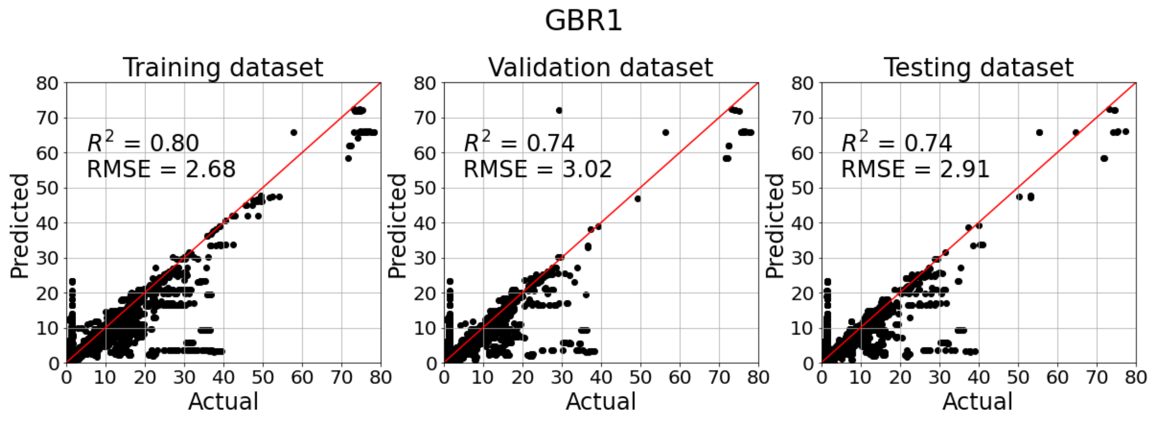


Figure 26: Predicted versus actual RMS values for the GBR1 model

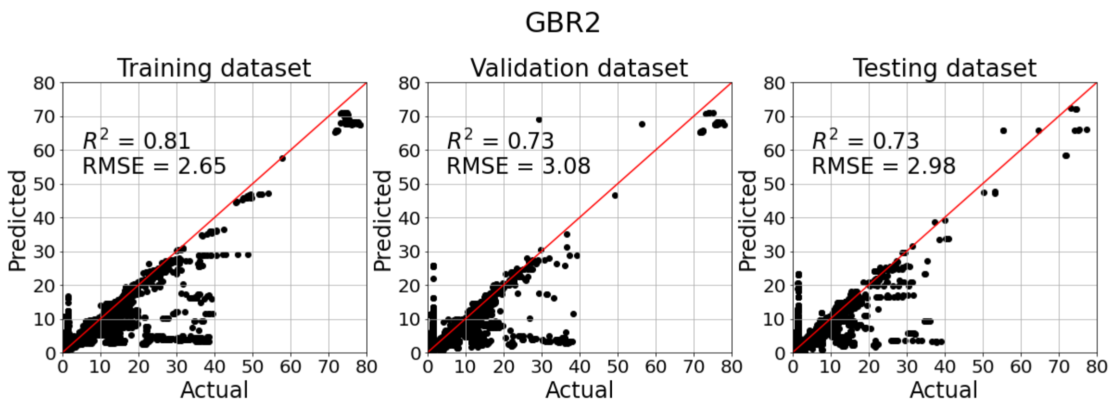


Figure 27: Predicted versus actual RMS values for the GBR2 model

In the case of the finalist GBR models it is apparent that the scatter plots have a similar distribution. The plots for these two models show a cluster of points where actual values are in the range between 20 and 40 while their corresponding predicted values are in the range of 0 to 10, indicating the model struggled to make reasonable predictions at that range. This cluster of points was not present in the XGBR models.

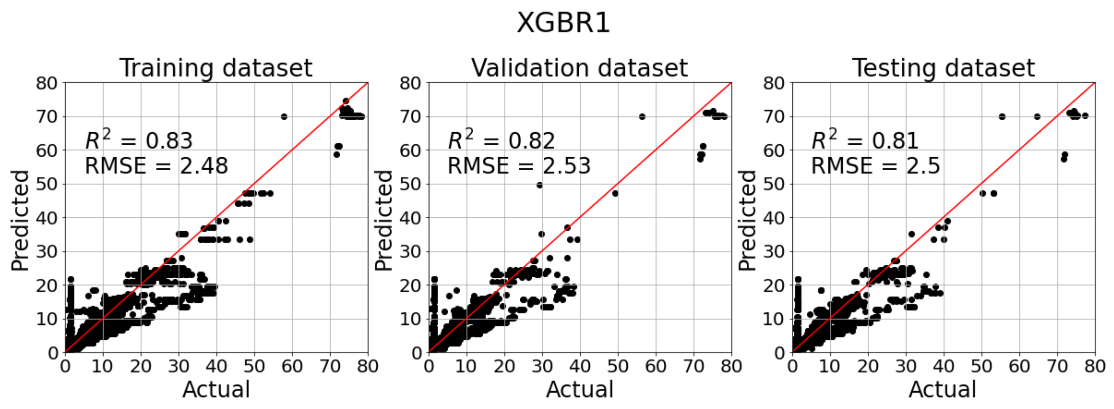


Figure 28: Predicted versus actual RMS values for the XGBR1 model

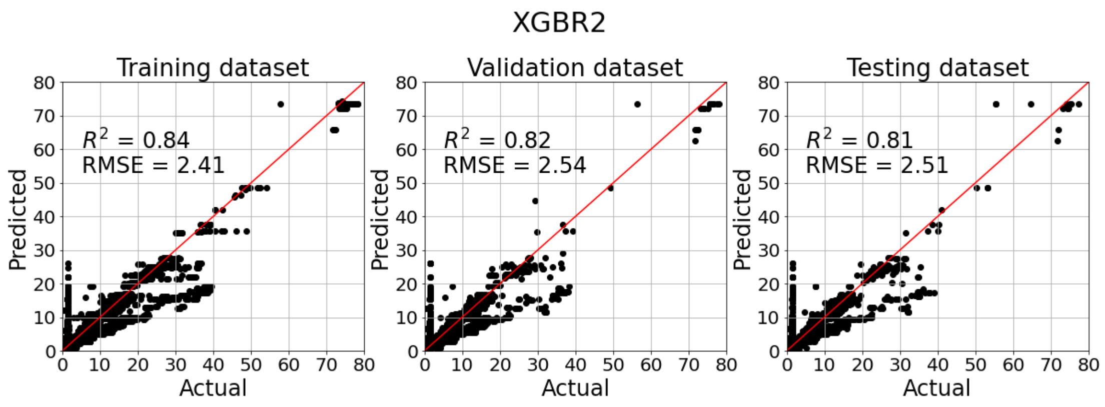


Figure 29: Predicted versus actual RMS values for the XGBR2 model

The plots for the finalist XGBR models showed very similar results where the points fall close to the best fit line, which indicates a strong performance. These plots did not show one XGBR model having an advantage over the other.

Overall, the XGBR models had higher values for R^2 as compared to the GBR models, but both had strong predictive performances. The better fit of the XGBR model compared to the GBR model can be attributed to its optimized framework.

5.1.1 Comparison with traditional statistical methods

As mentioned in section 2.1.2, when discussing overfitting issues, there are instances where the use of machine learning algorithms is not necessary. These cases can occur when datasets are simple and explanatory variables provide relevant and easily extractable information to predict the response variable. Traditional statistical methods, such as linear and polynomial regression have the advantage of using significantly less computational resources than machine learning algorithms.

The performance of these two methods was tested on the dataset using R^2 values as a metric to facilitate comparing their performance with that of the trained GBR and XGBR models.

Table 13: Coefficient of determination for linear and quadratic regression

Model	**Data Set**	$\boldsymbol{R^2}$ **value**
Linear Regression	Training	0.30
	Validation	0.29
	Test	0.29
Quadratic Regression	Training	0.37
	Validation	0.36
	Test	0.37

As seen in Table 13, the R^2 values for these models are significantly lower than those obtained by the gradient boosting models. The quadratic regression model showed an improvement in performance as compared to the simple linear regression model. Given that an increase in polynomial degree yielded an increase in performance, there may be a higher order polynomial regression that would adequately predict the bearing vibration curve and nullify the need for the established machine learning model. To verify this prospect, an optimization process was performed to find the polynomial order which would yield the highest R^2 value, which was then compared against the machine learning models developed for this project.

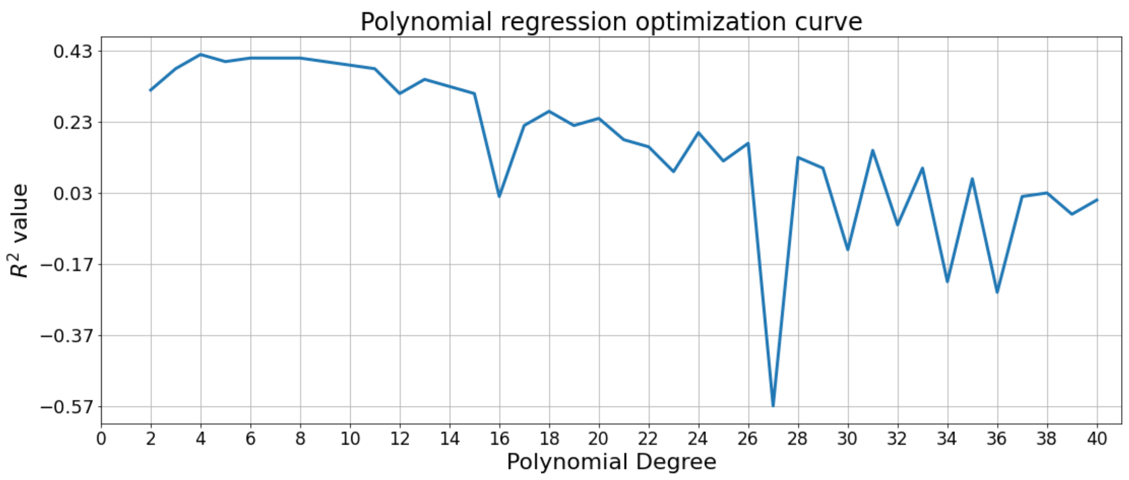


Figure 30: Optimization of Polynomial Regression

Figure 30 shows the optimization curve for polynomial regression. Polynomial degree values ranging from 2 to 40 were tested in the optimization process. An R^2 value was obtained for every value of polynomial degree and plotted for visualization. The optimization curve reveals an ideal value for a polynomial degree between 4 and 8. Given that a higher order polynomial regression is more computationally expensive, a degree of 4 was chosen as the optimal value since it was the lowest polynomial degree that yielded the highest R^2 value. Table 14 shows the results for the quartic regression model.

Table 14: Optimal Polynomial Regression Scores

Model	**Data Set**	**R^2 value**
Polynomial Regression (4th order)	Training	0.41
	Validation	0.41
	Test	0.41

The results for the optimization process show that even the polynomial regression model with the highest values for R^2 performs significantly worse when compared to the machine learning models developed in this study (see Table 8 and Table 9)

5.1.2 Individual experiment predictions

The next step was to assess the predictive performance of the developed models by visually comparing them to actual measured vibration curves. One experiment from the 4BT modality and another one from the SBT modality were chosen to verify if the models could account for the change in testing modality without a decline in performance. Experiment 200 was chosen for the 4BT case while experiment 201B was selected for the SBT case. Both experiments showcase high RMS values which serve to test whether the model can account for high variance in the data and still yield accurate predictions. Figure 31 to Figure 38 show the actual RMS values obtained throughout the duration of the experiments plotted against the GBR and XGBR model predictions.

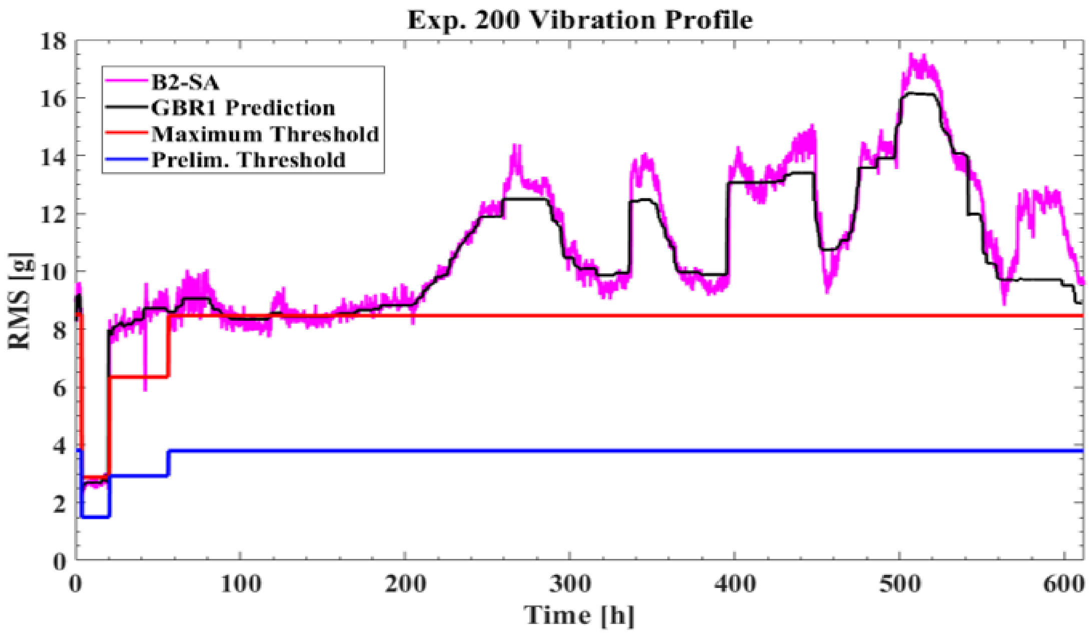


Figure 31: GBR1 model predictions for experiment 200

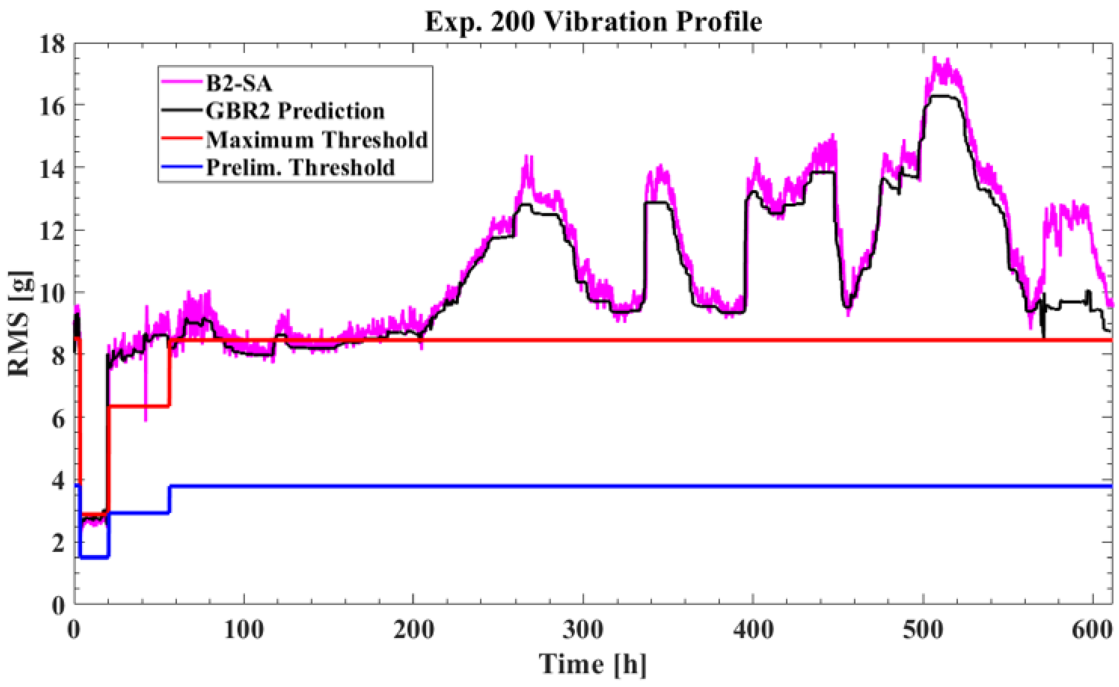


Figure 32: GBR2 model predictions for experiment 200

Figure 31 and Figure 32 allow for a visual comparison of the predictive performance of the two finalist GBR models on experiment 200. Both models had a strong performance with a slight deviation towards the end of the plot. As has been observed in previous comparisons, this visual assessment did not provide a clear answer as to which model was superior to the other.

Figure 33 and Figure 34 show the prediction plots of the XGBR models for experiment 200, where a similar behavior to the GBR models is observed. Both models had a strong performance with a slight deviation towards the end. The XGBR2 showed slightly higher peak values as compared to the XGBR1 which can be attributed to its increased “number of estimators” (as discussed in section 4.2.4). These allow the model to obtain more information from the input variables and recreate a more detailed functional relation, causing the predicted values to be closer to actual values.

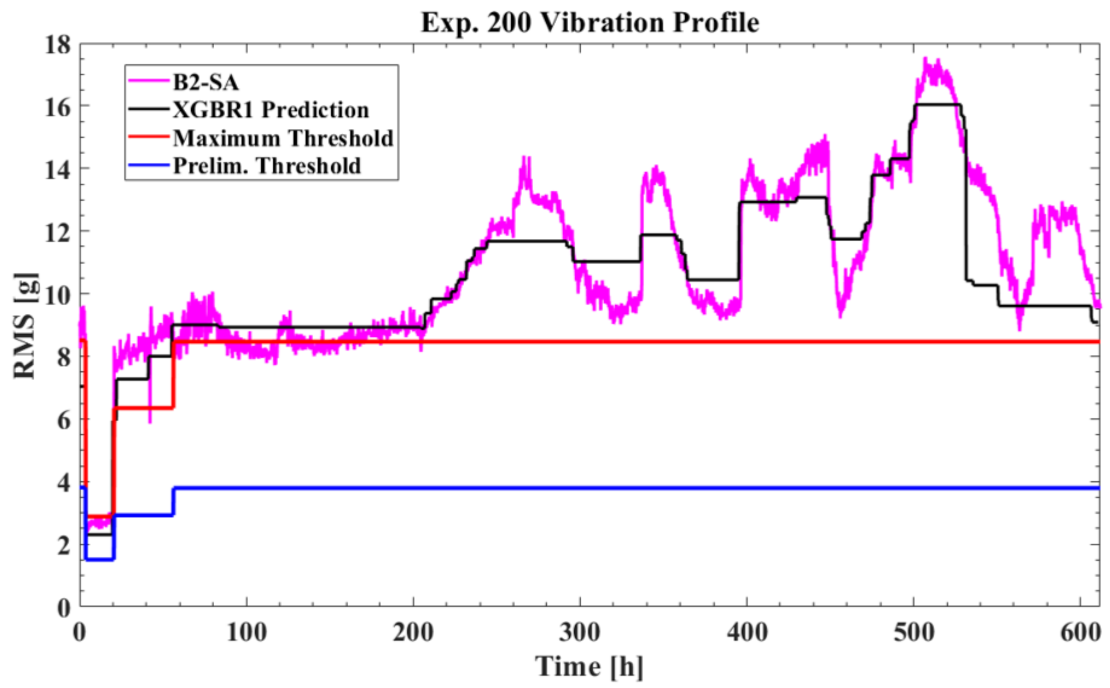


Figure 33: XGBR1 model predictions for experiment 200

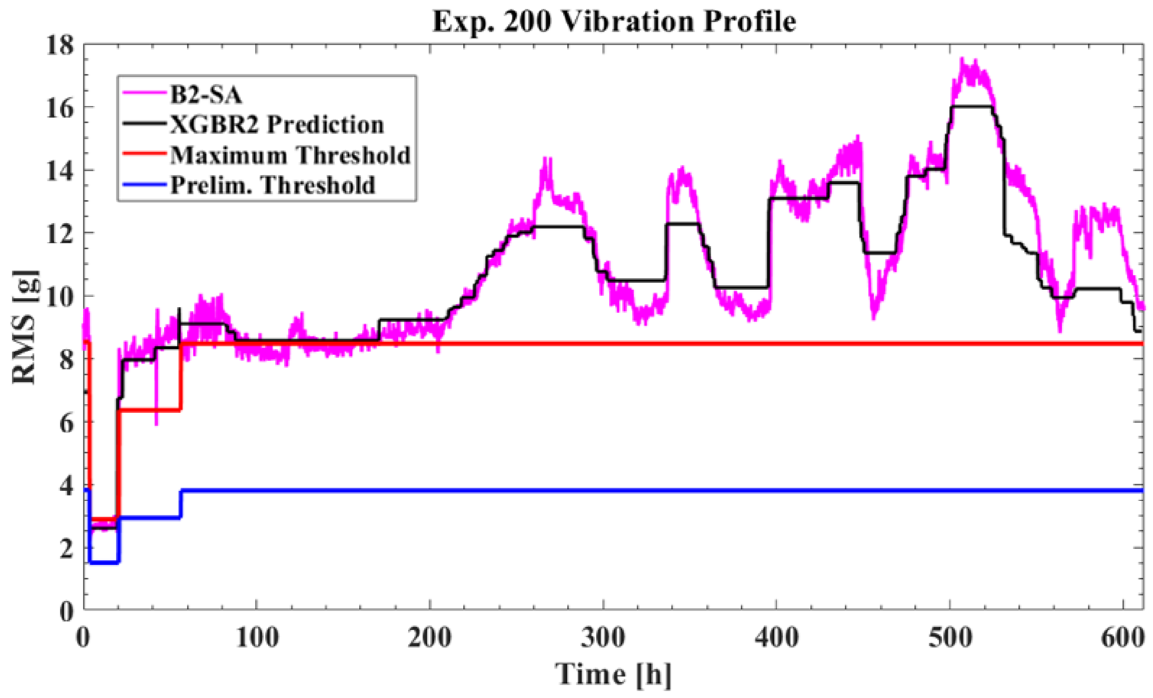


Figure 34: XGBR2 model predictions for experiment 200

There is a key difference between the GBR and XGBR models which is worth noting. Compared to the GBR models, the XGBR had “flatter” predictions on the peaks of the actual RMS data. These plateaus are indicative of more “conservative” predictions which can be attributed to the regularization hyperparameters, gamma and lambda, which are part of the XGBR but not the GBR. Overall, both models had a strong performance, and succeeded in one very important aspect: both models predicted the maximum threshold violation at the same point in time where the actual RMS data surpassed said threshold. The prediction of threshold violation is a crucial criterion to be met by these models since it is an indicator of end of service life.

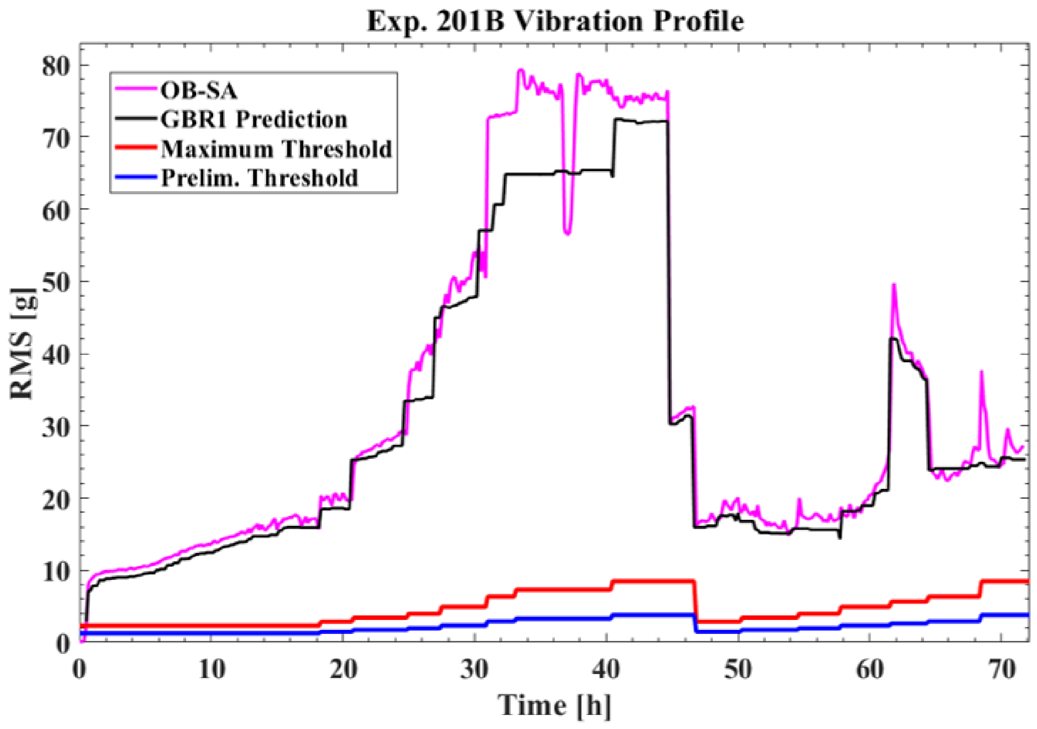


Figure 35: GBR1 model predictions for experiment 201B

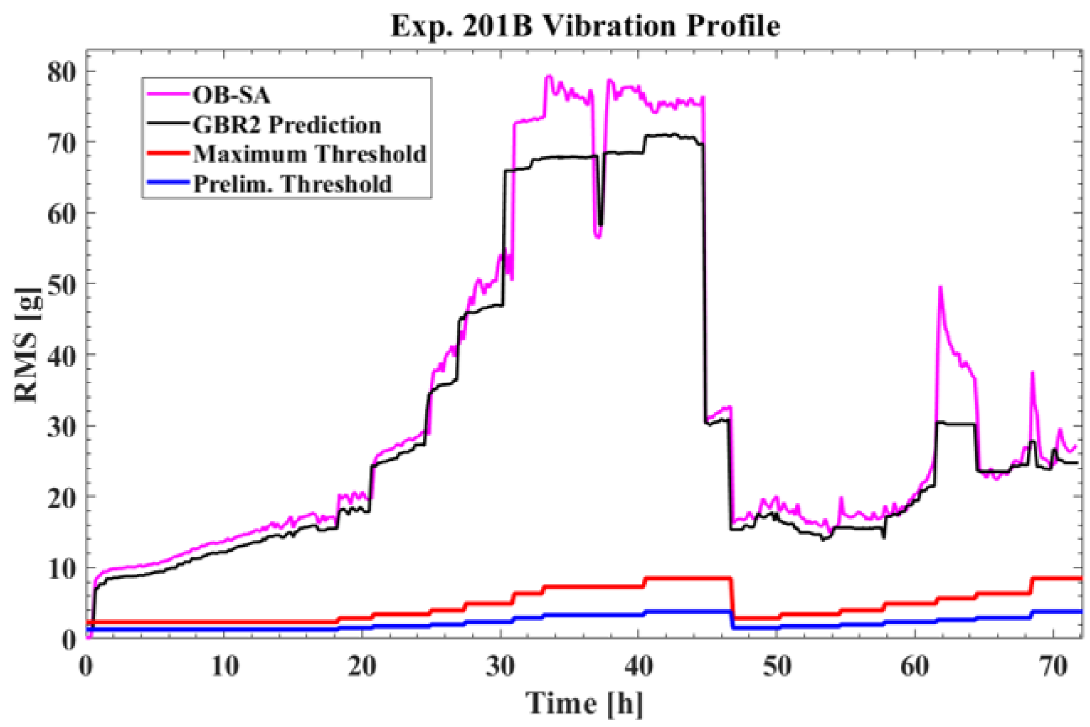


Figure 36: GBR2 model predictions for experiment 201B

Figure 35 and Figure 36 show the prediction plots of the finalist GBR models for experiment 201B. The GBR1 model failed to capture the sudden decrease in RMS at the 36-hour mark, but it effectively captured the sudden increase towards the end at the 62-hour mark. The opposite was true for the GBR2 model, which did capture the decrease in RMS at the 36-hour mark but had a more conservative prediction for the sudden increase at the 62-hour mark. The trade-off between predictive instances suggests that the GBR model only allows for some degree of certainty in predictions and that increasing model complexity will not result in better predictions given that the GBR2 is more complex than the GBR1. The GBR2 model still failed to make reasonable predictions for both sudden variations in RMS.

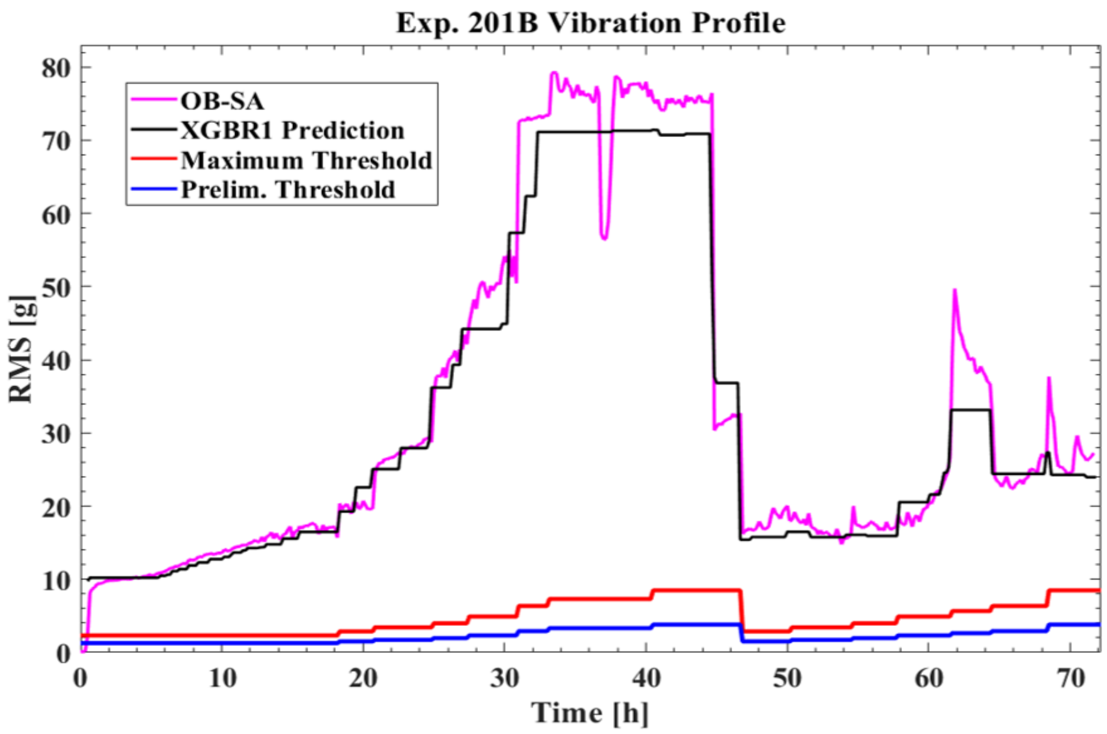


Figure 37: XGBR1 model predictions for experiment 201B

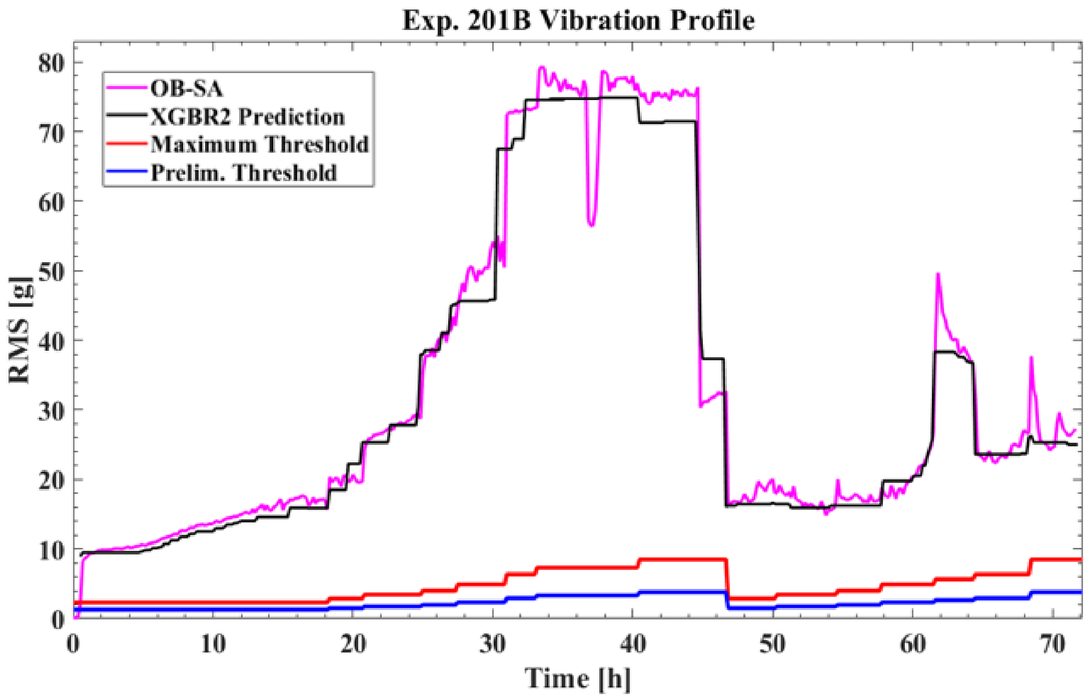


Figure 38: XGBR2 model predictions for experiment 201B

Figure 37 and Figure 38 show that for experiment 201B, both finalist XGBR models predict values closer to actual values when compared to the GBR predictions. Compared to the XGBR1, the XGBR2 model had predicted higher-fidelity values shown by how closely the predicted RMS curve matches the actual RMS curve, which is especially apparent when considering the sudden increase in RMS at the 62-hour mark. Both XGBR models neglected the decrease in RMS at the 36-hour mark, which is partly attributed to the regularization hyperparameters which prevent the model from reconstructing a relational function complex enough to capture them.

The plots reveal that the predictions for all models closely match the vibration curves. It is worthwhile to note that the XGBR models had flatter predictions when actual values showed peaks, which could be attributed to the regularization terms built into the algorithm. The key takeaway from these plots is that for all experiments, the models were able to accurately mark the point in time at which the vibration levels surpassed the established threshold. If a one-time violation of this established threshold is assumed to be the end of service life, then it can be concluded that both models can give a suitable estimate of the remaining service life of a bearing. Given that all models had similar performance, the final choice for the ideal model becomes that which has the lower computational time, which was the case for the GBR1 and XGBR1 models.

5.1.3 Mileage plots

The RMS curve was plotted against mileage, as opposed to time, to provide stakeholders a more concise and actionable value to reveal when the vibration levels for a given bearing will surpass the established threshold.

Figure 39 and Figure 40 show these mileage plots for experiment 200 with predicted curves from the GBR1 and XGBR1 models since they were chosen as the optimal models.

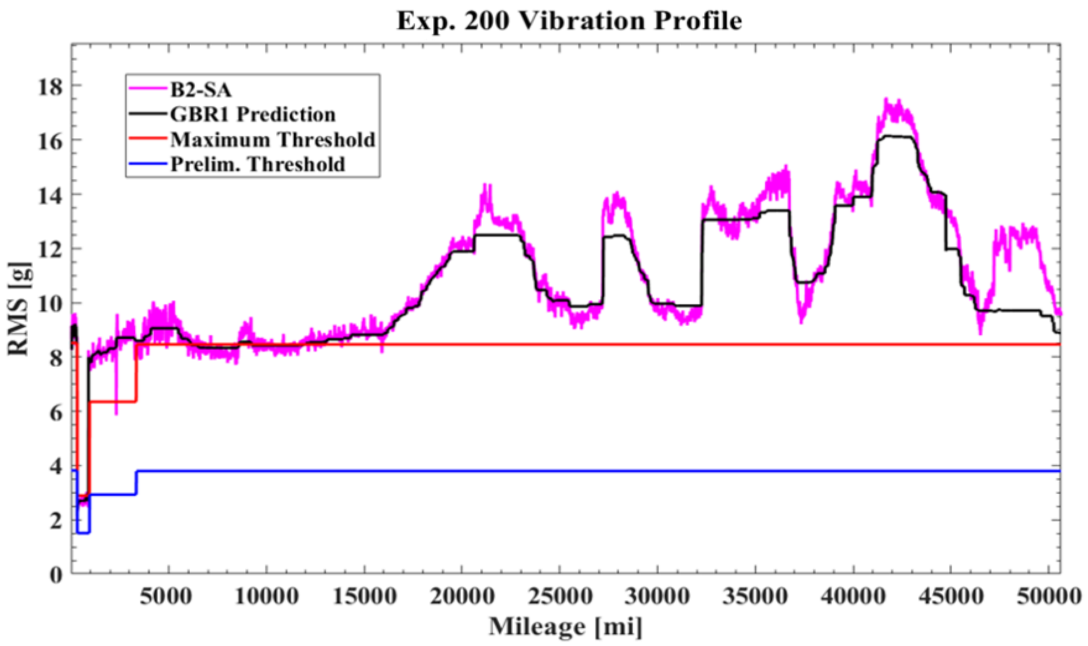


Figure 39: Experiment 200 RMS versus mileage plot for GBR1

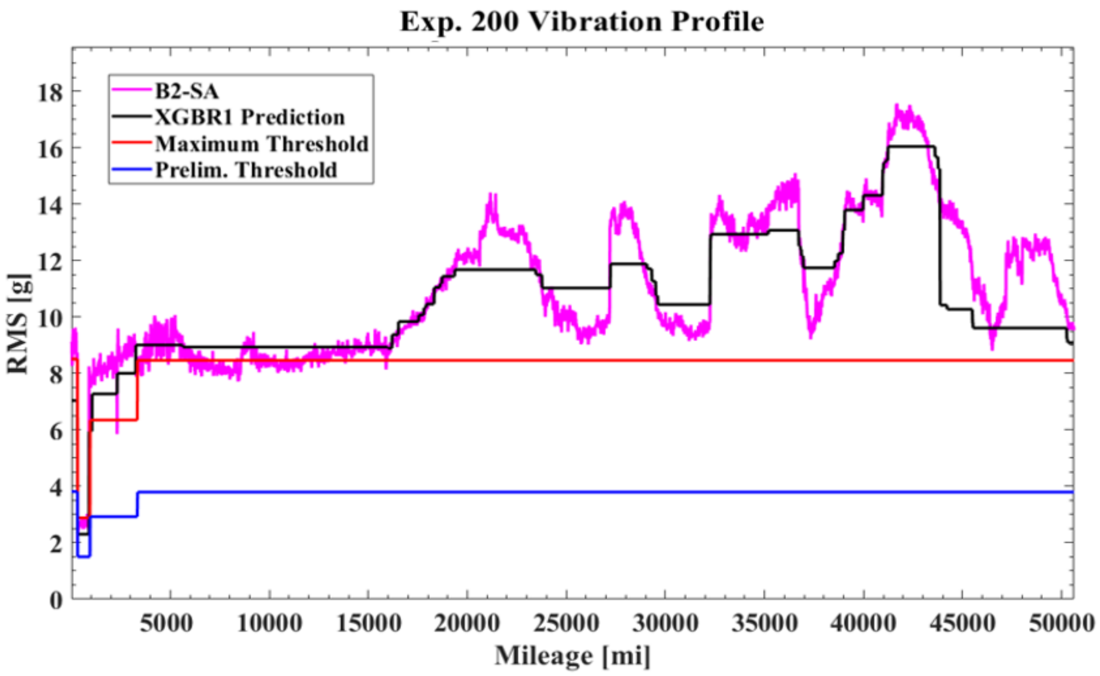


Figure 40: Experiment 200 RMS versus mileage plot for XGBR1

Figure 41 and Figure 42 show the GBR1 and XGBR1 predicted RMS curves versus mileage for experiment 201B. It should be noted that the mileage shown in these plots is only pertaining to the mileage for the given experiment and not the cumulative bearing mileage. The predicted curves indicate where the RMS levels surpass the established threshold, even on a per experiment basis, which is advantageous in displaying when the bearing reaches a critical state by way of milage in each trip, making it more communicable to stakeholders.

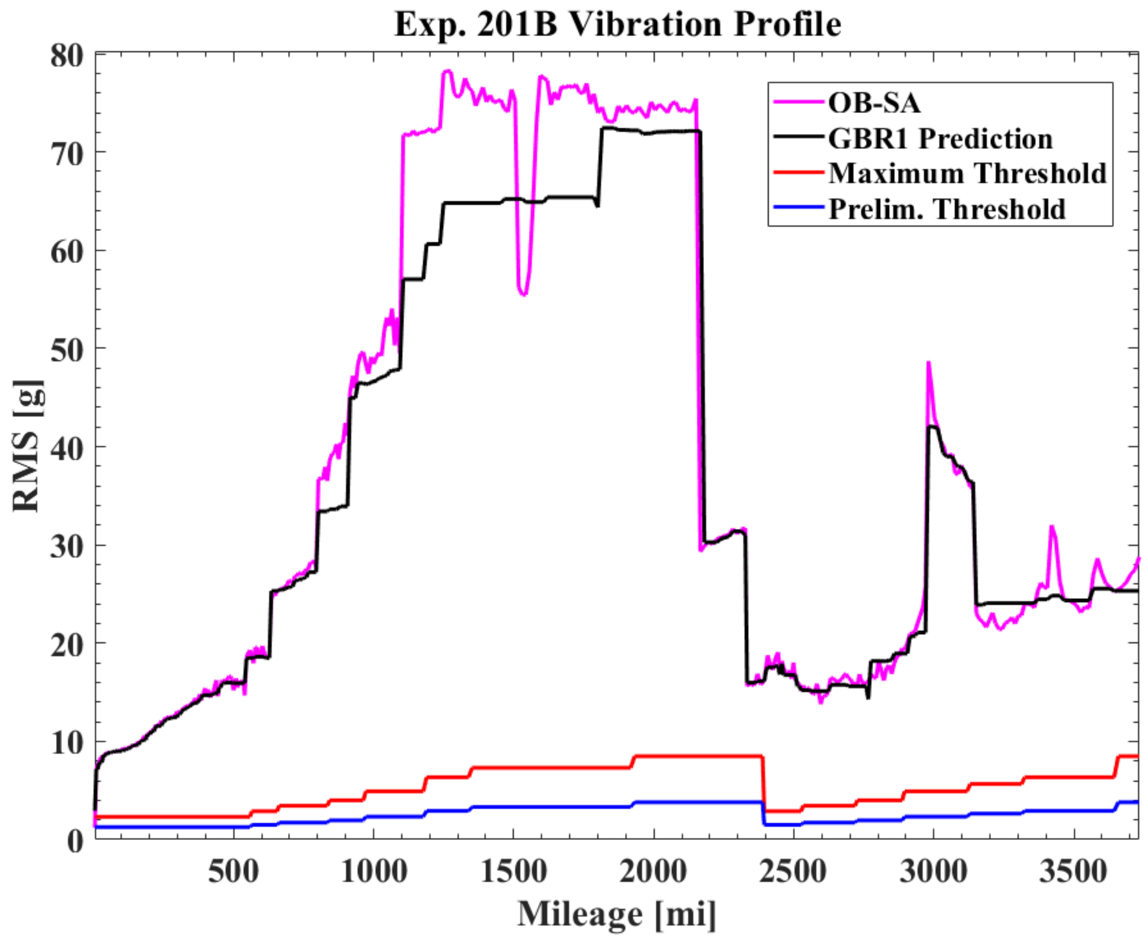


Figure 41: Experiment 201B RMS versus mileage plot for GBR1

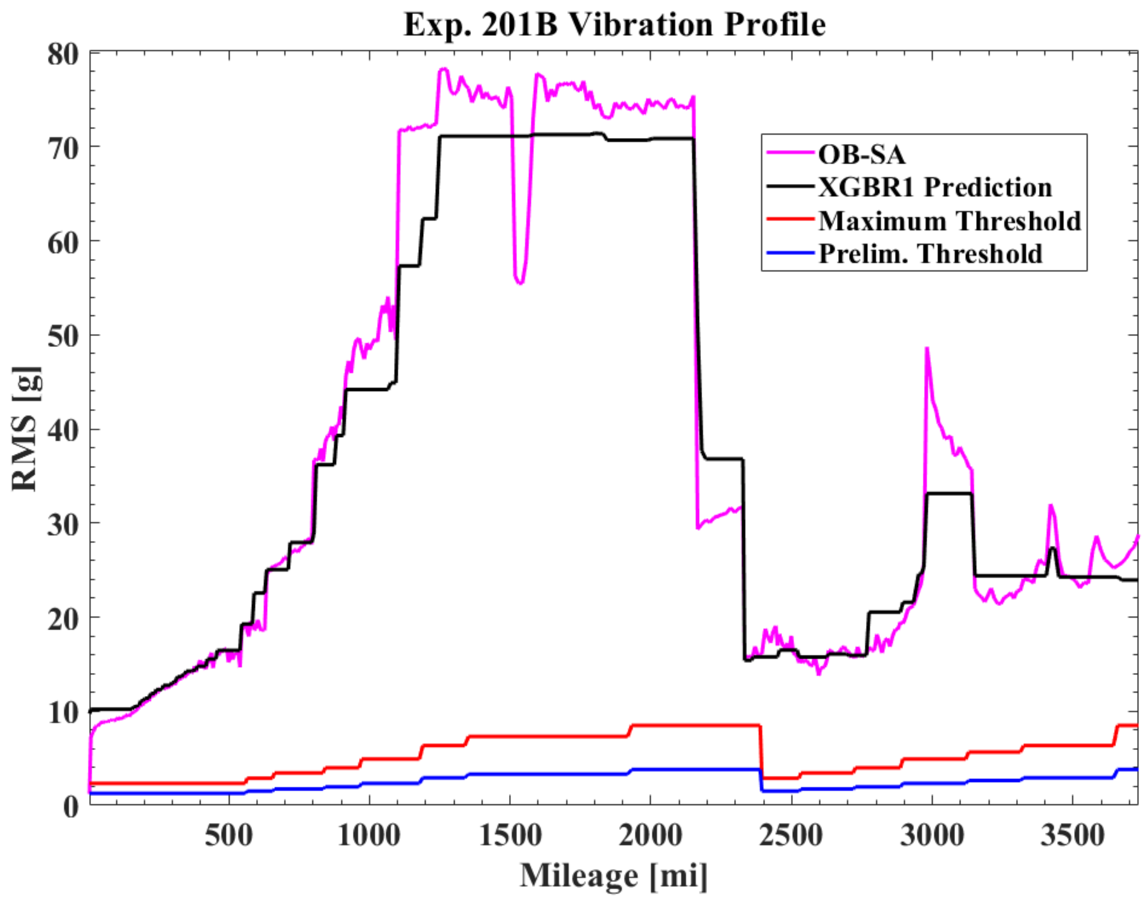


Figure 42: Experiment 201B RMS versus mileage plot for XGBR1

To convey the value of these graphs, experiment 200 can be considered an estimated route with pre-defined speeds and loads at each section. If strictly adhering to the case where a one-time violation of the maximum threshold indicates end of service life, then the bearing used for experiment 200 could only be used for 1,000 miles since that is when the predicted RMS surpasses the maximum threshold as seen in Figure 39 and Figure 40. Alternatively, 15,000 miles can be predicted as the remaining service life for that specific bearing given that RMS values dramatically increase above the threshold after that milestone. These prediction curves will help stakeholders make data-driven decisions, plan their maintenance schedules preventively and adjust the expected distance, speed, and load conditions of their trains.

5.1.4 Feature Importance

Once the predicted and actual RMS values were compared and the models yielded a satisfactory performance, the next step was to assess which variables were the most crucial in the predictions performed by the algorithm. That is, which variables provided the most information for the algorithm to reconstruct a functional relation and predict the RMS of a given bearing. For this purpose, the relative feature importance was obtained, along with SHAP explanative plots.

Developed in 1953 by Lloyd Shapley [40], Shapley or SHAP values help with the interpretation of machine learning predictions. The SHAP value is the average marginal contribution of a feature value across all possible feature combinations [41].

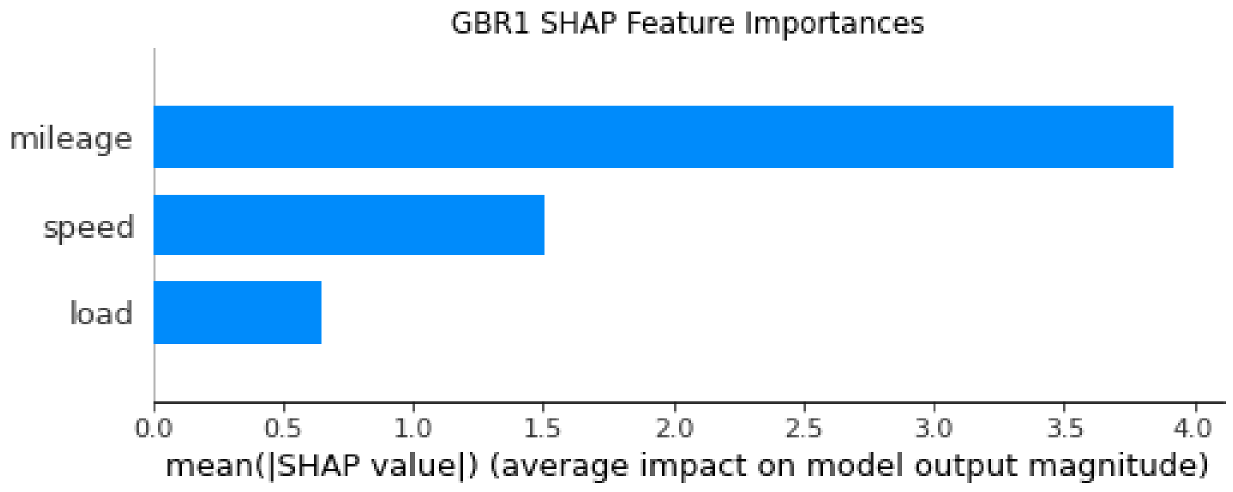


Figure 43: SHAP feature importance for GBR1

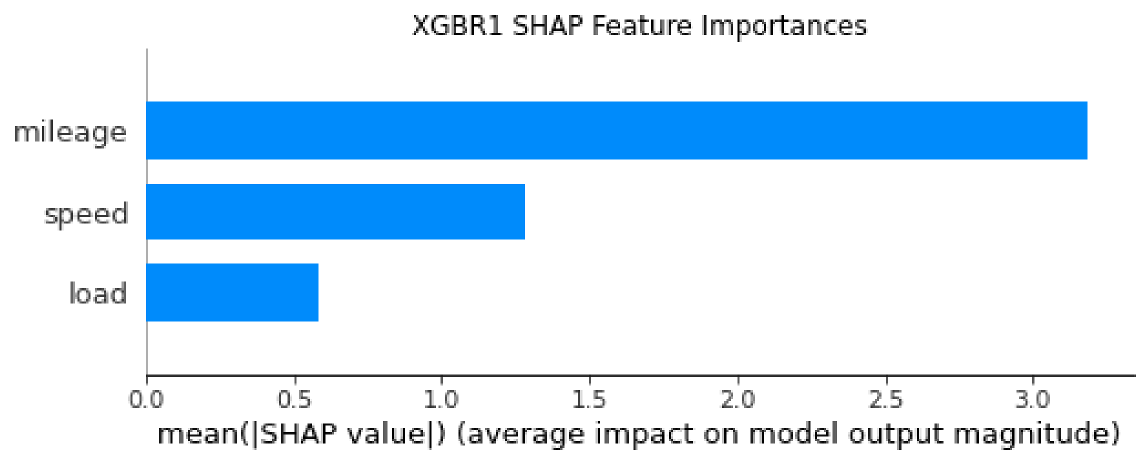


Figure 44: SHAP feature importance for XGBR1

Figure 43 and Figure 44 show that for both the GBR1 and XGBR1, the feature which provided the most information for RMS prediction is the cumulative bearing mileage. The cumulative bearing mileage is the only feature of the three chosen for this project which has "age-related" information. The speed and load features are the instantaneous values the bearing is experiencing at the time when RMS values were recorded, and while they have a positive impact on predictions, their contributions are lesser than that of mileage. A lesser contribution in comparison to that of other features does not indicate a negative impact on predictions or that the feature has no correlation to the response variable.

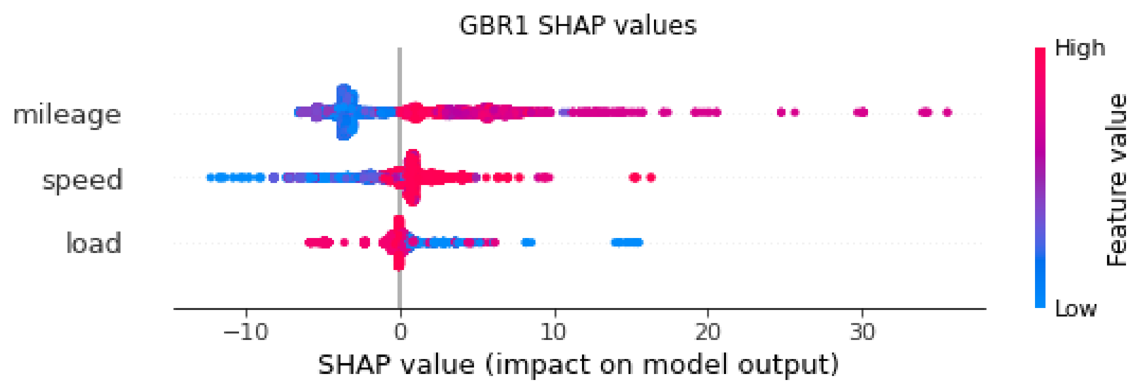


Figure 45: SHAP values for GBR1

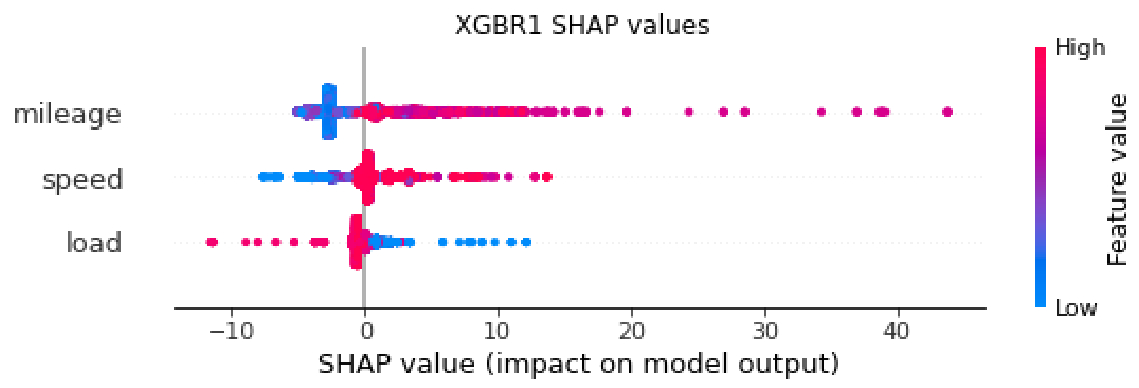


Figure 46: SHAP values for XGBR1

Figure 45 and Figure 46 showcase graphs for the SHAP values of each feature and a more detailed visualization of their contribution towards RMS prediction. An advantage of SHAP values is that they are measured in the same units as the response variable, which in this case means these SHAP values are measured in ‘g’. Each feature value is color coded where blue represents a low feature value, e.g., a mileage value of 10,000 miles, and magenta represents a high feature value, e.g., a mileage value of 500,000 miles. These feature values are then plotted according to their SHAP values which indicate their impact on the final prediction. A positive SHAP value indicates that the contribution of that specific feature value increased the value of the final prediction and a negative SHAP values indicates the feature value had a decreasing effect on the final predicted value. For example, in Figure 46 a point for mileage can have a SHAP value of close to 40; this point is colored in magenta which means that it was a high value for mileage and its contribution to the final prediction was near to 40 g.

The GBR and XGBR SHAP value graphs show similar trends. High values for mileage resulted in high SHAP values suggesting that higher mileage translates to higher RMS values, which is in agreement with the observations made on the experiments conducted at the UTCRS. Low values for speed were associated with negative SHAP values, suggesting that bearings operating at low speeds yield low values for RMS. In the case for load, low load values resulted in high SHAP values, this may be attributed to the fact that the majority of RMS readings were obtained at 17% load operating conditions as was shown in Figure 19 in section 4.2.2 which also reveals that peak RMS values also happen at 17% load conditions.

5.2 Federated XGBoost

The performance of the XGBoost model developed in a federated learning framework was assessed in a similar way to that of the centralized approach. It should be noted that the environment provided by IBM for federated learning did not allow for a 60-20-20 data split since only a training and test set could be defined using their services, therefore, 80% of the data was allocated to the training set and the remaining 20% to the test set. Furthermore, the metrics were automatically provided once the training stage started and were based solely on the test set. As with the centralized models, the first step was to determine a baseline performance by training and testing the model with default hyperparameter values to then optimize the hyperparameters and compare the results. Table 15 contains the default and tuned hyperparameter values for this model while Table 16 shows their respective R^2 and RMSE values on the testing data set.

Table 15: Federated XGBoost default and tuned hyperparameter values

Hyperparameter	**Baseline Model**	**Tuned Model**
Learning Rate	0.1	0.05
Number of Rounds	100	35

Table 16: Federated XGBoost metric values

Metric	Baseline Model	Tuned Model
Test R^2	0.66	0.71
Test RMSE	12.54	10.54

There was a slight increase in performance between the baseline and the tuned model which indicates the hyperparameter tuning process did influence the model, however, the change was not as significant as with the centralized models.

The R^2 value of the tuned model for the test dataset was not much lower than that which was obtained with the finalist GBR models developed in a centralized framework, suggesting that the federated framework would give similar results. The RMSE, however, was significantly higher than that of the centralized models which signaled a possible decrease in performance as compared to its centralized counterpart. It is worthwhile to note that this comparison of multiple metrics highlights the advantage of not relying on a single metric value to assess model performance.

To further explore the federated XGBoost performance, scatter plots for the actual and predicted values for the whole data set were generated to see how closely the curves matched.

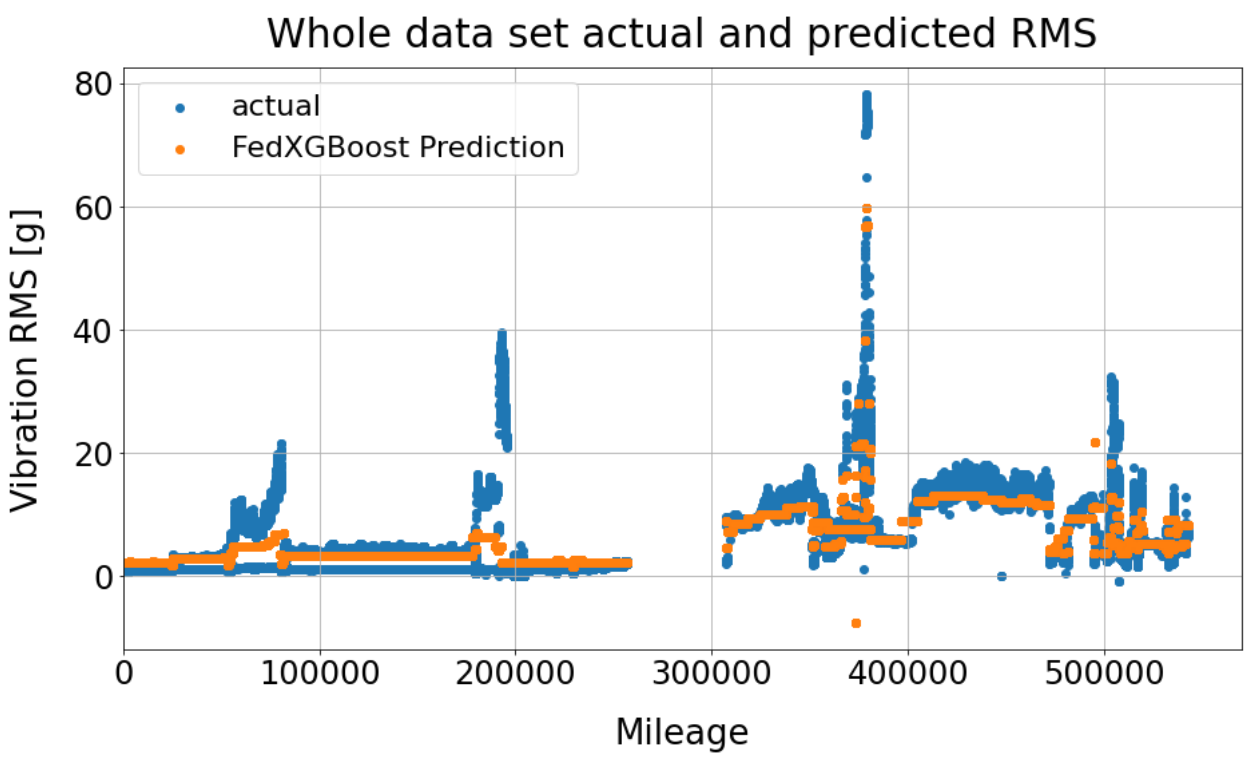


Figure 47: Federated XGBoost predictions on whole data set as compared to actual values

Figure 47 shows the scatter plot of actual and predicted values against mileage. This plot used for preliminary assessment suggested that the model had an acceptable performance, however, further analysis was required to determine whether the model was indeed able to generate satisfactory predictions. To this end, a similar scatter plot as those shown in section 5.1 was generated where all predicted and actual values were graphed to see how accurately the cluster of points aligned to the perfect fit line.

Figure 48 shows that the cluster of points has a significantly different distribution as compared to the scatter plots from the centralized models. The scatter plot for the federated XGBoost model shows a tendency for predicted values to spread along the x-axis which deviates from the perfect fit line. Furthermore, the plot shows that there are instances where the model predicted negative values, which is inconsistent with the RMS formula where part of the equation indicates to square all the values to be considered, effectively eliminating all negative values.

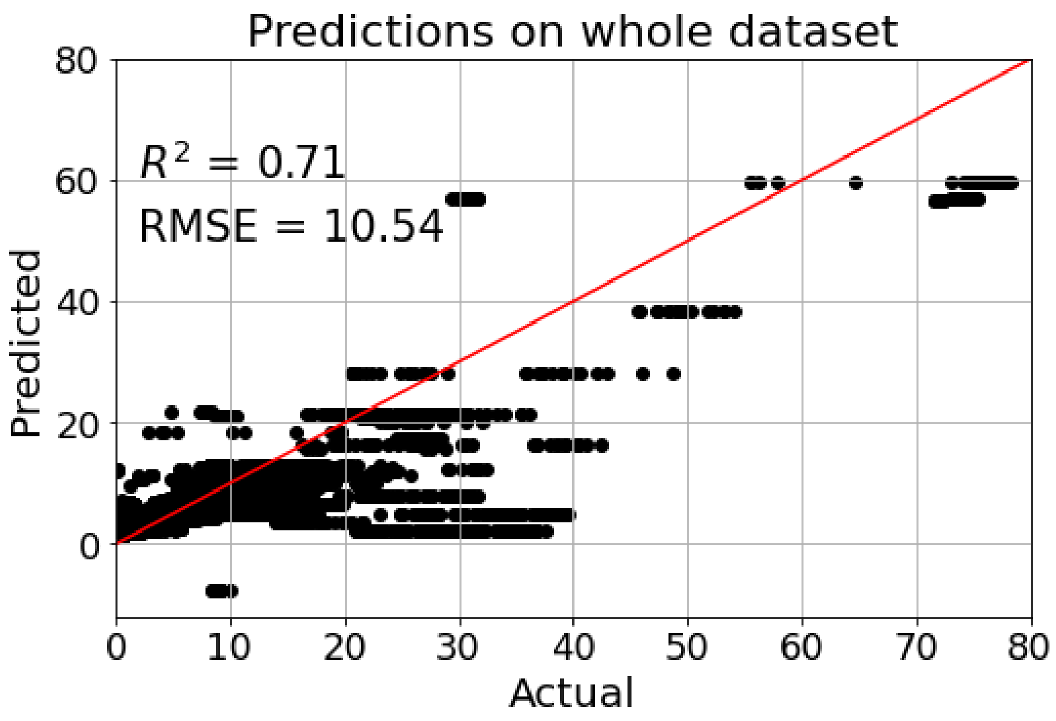


Figure 48: Predicted versus actual RMS values for the Federated XGBoost model

Figure 48 provided further signs that suggested the model was not successfully carrying out its predictive task. The last step needed to confirm that the model was indeed not providing satisfactory predictions was to generate predicted RMS curves for individual experiments as shown in section 5.1.2. Figure 49 and Figure 50 show the predicted RMS curves for experiments 200 and 201B which provide a consistent point of comparison to the centralized GBR and XGBR models. The predicted curves differ significantly from the actual curves, confirming that the federated XGBoost model was highly biased. A biased model signifies that the reconstructed functional relation was not complex enough to effectively capture the relation between input and output variables. This decrease in performance between centralized and federated machine learning models highlights the trade-off between performance and privacy preservation.

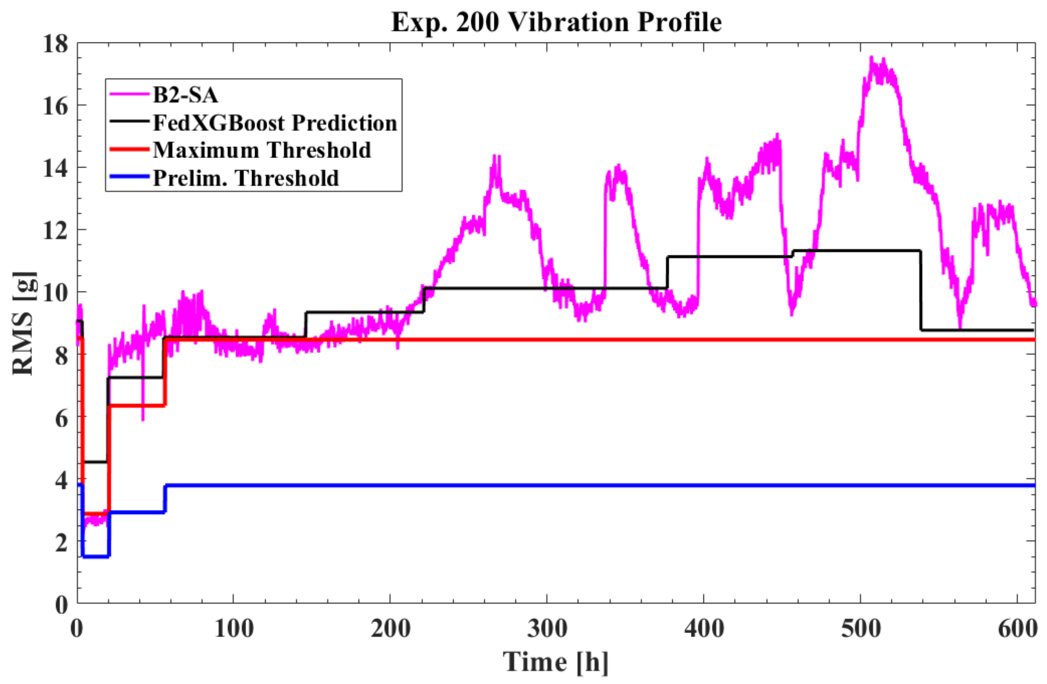


Figure 49: Federated XGBoost model predictions for experiment 200

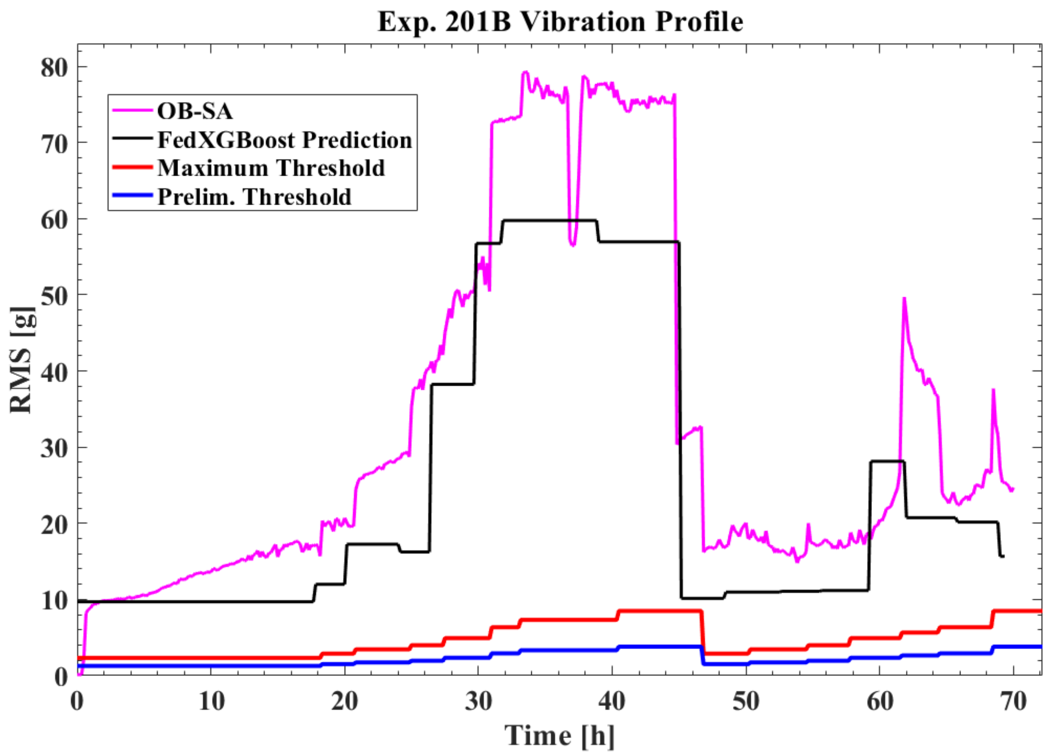


Figure 50: Federated XGBoost model predictions for experiment 201B

The bias in the Federated XGBoost model can be attributed to several factors. As discussed in section 4.3.1, the federated framework provided by IBM only allowed for customization of two hyperparameters which were the learning rate and the number of rounds. Optimization of other hyperparameters could have proven beneficial in improving model performance. Another aspect to be considered is the fact that the only available choice for loss function was the squared loss function. It is possible that the use of the Huber loss function could also have had a positive impact on model performance. Lastly, as discussed in section 2.4.1, federated learning models make use of summary information or “updates” from local models to train the global model. The use of these updates as training information could explain the negative predictions obtained with this model given that if the model had used the actual RMS data, no negative values would have been introduced in the training data set and thus no negative predictions would be possible.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

Defective bearings in operation must be detected in a timely manner to avoid inefficient performance and even catastrophic failure. The wayside detection technologies currently in use in the majority of the North American Railroad, such as HBDs and TADS are unreliable and thus alternative detection systems have been in development by several groups. For over a decade, the UTCRS team at the University of Texas Rio Grande Valley has taken significant steps towards this objective through the development of onboard monitoring sensors and data collection. These onboard sensors continuously capture the temperature and vibration levels of bearings in operation and alert operators when readings exceed established thresholds for safe operation. The continuous monitoring of bearings provides a more reliable solution to flagging defective bearings in an accurate and timely manner.

The work presented in this thesis aims at leveraging the data collected by the team at the UTCRS by using it to train machine learning algorithms that can predict future bearing vibration levels. Gradient Boosting and eXtreme Gradient Boosting Regression models in a centralized and federated framework were developed to predict the remaining service life of train bearings.

Both centralized models proved to be significantly more accurate in their prediction as compared to traditional statistical methods such as linear and polynomial regression. These models come as a practical option to predict the RMS vibration curve of bearings given that only three input variables are needed, namely cumulative bearing mileage, operating speed, and load.

Operating speed and load are easy to obtain since they must be specified before a train is scheduled for operation. In contrast, cumulative bearing mileage may not be readily available given that in field operations individual bearing mileage is not usually recorded, however, simple mileage record keeping can be implemented in operations to obtain this information.

Analysis of the feature importance of the final models showed results that were consistent with trends previously observed at the UTCRS. More specifically, it was observed that the models had more difficulty using data points with low values for speed to predict RMS, which is consistent with observations at the center where RMS readings have significant variance at low speeds. Another observation from feature importance analysis which matched historical trends was the fact that bearings with higher mileage yielded higher RMS values regardless of speed and load, which comes as an expected result given that they have been subjected to more wear.

The federated model did not have a satisfactory performance. The trade-off between performance and privacy preservation is one of the main challenges facing the recently developed field of federated learning which is rapidly expanding. The decrease in model performance when using the federated framework may be attributed to the limitations associated with it. One critical limitation imposed by the chosen federated learning framework is the inability to modify and optimize hyperparameters other than the learning rate and number of rounds. Furthermore, using the squared loss function as opposed to the Huber loss function may have hindered model performance.

Further improvement of these models can be achieved through different methods. Namely, an increase in the amount of training data can prove beneficial, especially if coming from additional bearings not considered in this study. The addition of explanatory variables such as the cumulative speed and load the bearing has experienced through its service life (as opposed to the instantaneous speed and load considered in this project) may also help create more robust models. Another explanatory variable worthy of consideration is the “maximum g” value which refers to the highest number included in the RMS calculation.

The mentioned options for model refinement are currently being considered in an iteration of this project also performed by the UTCRS team. This second-generation project aims at expanding the amount of data used to train the machine learning algorithms. This additional data will come from a wider scope of bearings for which data is available at the UTCRS records, which includes bearings for which full mileage records or other values may not be available. To address these missing values, data processing techniques will be implemented in order to make a more efficient use of the vast records at the UTCRS database.

The developed models along with onboard sensors provide a data-driven tool that stakeholders in the rail industry can use to guide their decisions regarding maintenance of train bearings. These models will allow for the creation of predictive maintenance schedules which will make rail operations safer and markedly more cost-efficient.

REFERENCES

[1] “Freight Rail Overview | FRA.” https://railroads.dot.gov/rail-network-development/freight-rail-overview.

[2] “Rail regulations.” https://www.cpr.ca/en/safety/rail-regulations (accessed Apr. 13, 2022).

[3] United States. Department of Transportation. Bureau of Transportation Statistics, “National Transportation Statistics (NTS),” 2019, doi: 10.21949/1503663.

[4] “Liu-et-al-IHHA-2011-Track-Class-Upgrade-Final.pdf.” Accessed: Nov. 16, 2021. [Online]. Available: http://railtec.illinois.edu/wp/wp-content/uploads/pdf-archive/Liu-et-al-IHHA-2011-Track-Class-Upgrade-Final.pdf

[5] “FRA-Homepage.” https://safetydata.fra.dot.gov/OfficeofSafety/default.aspx

[6] C. Tarawneh, J. Montalvo, and B. Wilson, “Defect detection in freight railcar tapered-roller bearings using vibration techniques,” *Rail. Eng. Science*, vol. 29, no. 1, pp. 42–58, Mar. 2021, doi: 10.1007/s40534-020-00230-x.

[7] “Freight Car Components,” *Amsted Rail*. https://www.amstedrail.com/products/freight-car-components/ (accessed Apr. 13, 2022).

[8] M. F. Stewart and E. Flynn, “An Implementation Guide for Wayside Detector Systems,” p. 99.

[9] C. Tarawneh, L. Sotelo, A. Villarreal, N. De Los Santos, R. Lechtenberg, and R. Jones, “Temperature Profiles of Railroad Tapered Roller Bearings With Defective Inner and Outer Rings,” Apr. 2016, p. V001T06A018. doi: 10.1115/JRC2016-5816.

[10] T. S. B. of C. Government of Canada, “Railway Investigation Report R11T0016 - Transportation Safety Board of Canada,” Dec. 07, 2011.

[11] “Rail Insider-When a train derails, many railroads turn to contractors to contain and clean up spills. Information For Rail Career Professionals From Progressive Railroading Magazine,” *Progressive Railroading*.

[12] H. Singh, “Understanding Gradient Boosting Machines,” *Medium*, Nov. 04, 2018.

[13] I. El Naqa and M. J. Murphy, “What Is Machine Learning?,” in *Machine Learning in Radiation Oncology: Theory and Applications*, I. El Naqa, R. Li, and M. J. Murphy, Eds. Cham: Springer International Publishing, 2015, pp. 3–11. doi: 10.1007/978-3-319-18305-3_1.

[14] J. H. Friedman, “Greedy Function Approximation: A Gradient Boosting Machine,” *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[15] A. Natekin and A. Knoll, “Gradient boosting machines, a tutorial,” *Frontiers in Neurorobotics*, vol. 7, 2013.

[16] K. Nyuytiymbiy, “Parameters and Hyperparameters in Machine Learning and Deep Learning,” *Medium*, Mar. 28, 2022.

[17] “ML | Underfitting and Overfitting,” *GeeksforGeeks*, Nov. 23, 2017.

[18] “Regularization in Machine Learning || Simplilearn,” *Simplilearn.com*. https://www.simplilearn.com/tutorials/machine-learning-tutorial/regularization-in-machine-learning

[19] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, Aug. 2016, doi: 10.1145/2939672.2939785.

[20] J. Brownlee, “A Gentle Introduction to XGBoost for Applied Machine Learning,” *Machine Learning Mastery*, Aug. 16, 2016.

[21] “Health Insurance Portability and Accountability Act of 1996 (HIPAA) | CDC,” Feb. 21, 2019.

[22] “Federated Learning: Collaborative Machine Learning without Centralized Training Data,” *Google AI Blog*. http://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[23] *Federated XGBoost*. mc2-project, 2022. [Online]. Available: https://github.com/mc2-project/federated-xgboost

[24] “IBM Docs,” Mar. 30, 2022. https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-

south.containers.appdomain.cloud/docs/en/cloud-paks/cp-data/4.0?topic=samples-federated-learning-xgboost-tutorial

[25] L. Barua, B. Zou, M. Noruzoliaee, and S. Derrible, “A gradient boosting approach to understanding airport runway and taxiway pavement deterioration,” *International Journal of Pavement Engineering*, vol. 22, no. 13, pp. 1673–1687, Nov. 2021, doi: 10.1080/10298436.2020.1714616.

[26] Z. Allah Bukhsh, A. Saeed, I. Stipanovic, and A. G. Doree, “Predictive maintenance using tree-based classification techniques: A case of railway switches,” *Transportation Research Part C: Emerging Technologies*, vol. 101, pp. 35–54, Apr. 2019, doi: 10.1016/j.trc.2019.02.001.

[27] B. Authors: Snider, “Improving Time to Failure Predictions for Water Distribution Systems Using Extreme Gradient Boosting Algorithm: (049),” *WDSA / CCWI Joint Conference Proceedings*, vol. 1, Jul. 2018 [Online]. Available: https://ojs.library.queensu.ca/index.php/wdsa-ccw/article/view/12099

[28] H. Li *et al.*, “Improving rail network velocity: A machine learning approach to predictive maintenance,” *Transportation Research Part C: Emerging Technologies*, vol. 45, pp. 17–26, Aug. 2014, doi: 10.1016/j.trc.2014.04.013.

[29] Mathuranathan, “Significance of RMS (Root Mean Square) value,” *GaussianWaves*, Jul. 23, 2015.

[30] A. Gonzalez, “Development, optimization, and implementation of a vibrtion-based detection algorithm for railroad bearings.,” University of Texas Rio Grande Valley, Edinburg, Tx, 2015.

[31] S. Kumar, “Data splitting technique to fit any Machine Learning Model,” *Medium*, May 01, 2020.

[32] “What is Python? Executive Summary,” *Python.org*.

[33] “About us,” *scikit-learn*. https://scikit-learn/stable/about.html

[34] “XGBoost Documentation — xgboost 1.6.0 documentation.” https://xgboost.readthedocs.io/en/stable/index.html

[35] “When to Use One-Hot Encoding in Deep Learning?,” *Analytics India Magazine*, Aug. 25, 2021. https://analyticsindiamag.com/when-to-use-one-hot-encoding-in-deep-learning/

[36] “coefficient of determination | Interpretation & Equation | Britannica.” https://www.britannica.com/science/coefficient-of-determination

[37] “What is Categorical Data | Categorical Data Encoding Methods,” *Analytics Vidhya*, Aug. 13, 2020. https://www.analyticsvidhya.com/blog/2020/08/types-of-categorical-data-encoding/

[38] “sklearn.ensemble.GradientBoostingRegressor,” *scikit-learn*. https://scikit-learn/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

[39] “Root Mean Square Error (RMSE),” *C3 AI*. https://c3.ai/glossary/data-science/root-mean-square-error-rmse/

[40] L. S. Shapley, “A Value for N-Person Games,” RAND Corporation, Mar. 1952.

[41] C. Molnar, *9.5 Shapley Values | Interpretable Machine Learning*. [Online]. Available: https://christophm.github.io/interpretable-ml-book/shapley.html

APPENDIX

APPENDIX

Full administrator script for Federated XGBoost

```
# @hidden_cell
# The project token is an authorization token that is used to access proje
ct resources like data sources,
#connections, and used by platform APIs.

from project_lib import Project
project = Project(project_id='51864300-0251-4e7b-8c4e-c7f6e75f73d4', proje
ct_access_token='p-5dc55297565421704fbddf40089c6003ce9aa935')

API_VERSION = "2019-10-25"

WML_SERVICES_HOST = "us-south.ml.cloud.ibm.com" # or "eu-de.ml.cloud.ibm.c
om"
WML_SERVICES_URL = "https://" + WML_SERVICES_HOST
IAM_TOKEN_URL = "https://iam.cloud.ibm.com/oidc/token"

# Name of your COS bucket
COS_BUCKET = "bearingdata1-donotdelete-pr-4wsaodz5ooiqpk"

IAM_APIKEY = "KjO_qP6r7I-s1cm3NUkRs2smTgjqsnUXn9rQF9tG2Ezs"

# Get this from Manage < IAM < Users, and check the URL. Your user ID shou
ld be in the format IBMid-<xxx>.
CLOUD_USERID = "IBMid-661003DZHG"

PROJECT_ID = "51864300-0251-4e7b-8c4e-c7f6e75f73d4" # Get this by going in
to your WS project and checking the URL, or create a project token and cli
ck "Insert project token"

import urllib3
import requests
import json
from string import Template

urllib3.disable_warnings()
```

```
payload = "grant_type=urn:ibm:params:oauth:grant-type:apikey&apikey=" + IA
M_APIKEY
token_resp = requests.post(IAM_TOKEN_URL ,
                          headers={"Content-Type": "application/x-www-form
-urlencoded"},
                          data = payload,
                          verify=True)

print(token_resp)

token = "Bearer " + json.loads(token_resp.content.decode("utf-8"))["access
_token"]
print("WS token: %s " % token)

wml_remote_training_system_asset_one_def = Template("""
{
  "name": "Bearing RTS",
  "project_id": "$projectId",
  "description": "Remote Training System for Bearing Data1",
  "tags": [ "Federated Learning" ],
  "organization": {
    "name": "IBM",
    "region": "US"
  },
  "allowed_identities": [
    {
      "id": "$userID",
      "type": "user"
    }
  ],
  "remote_admin": {
    "id": "$userID",
    "type": "user"
  }
}
""").substitute(userID = CLOUD_USERID,
                projectId = PROJECT_ID)


wml_remote_training_system_one_resp = requests.post(WML_SERVICES_URL + "/m
l/v4/remote_training_systems",
                                                    headers={"Content-Type
": "application/json",
                                                             "Authorizatio
n": token},
                                                    params={"version": API
_VERSION,
                                                             "project_id":
PROJECT_ID},
                                                     data=wml_remote_traini
ng_system_asset_one_def,
                                                     verify=False)

print(wml_remote_training_system_one_resp)
status_json = json.loads(wml_remote_training_system_one_resp.content.decod
e("utf-8"))
print("Create remote training system response : "+ json.dumps(status_json,
indent=4))

wml_remote_training_system_one_asset_uid = json.loads(wml_remote_training_
system_one_resp.content.decode("utf-8"))["metadata"]["id"]
print("Remote Training System id: %s" % wml_remote_training_system_one_ass
et_uid)

training_payload = Template("""
{
  "name": "FL Aggregator",
  "tags": [
    {
      "value": "tags_jobs_fl",
      "description": "Sample FL Aggregator"
    }
  ],
  "federated_learning": {
    "fusion_type": "xgb_regressor",
    "learning_rate": 0.1,
    "loss": "squared_error",
    "max_bins": 255,
    "rounds": 100,
    "metrics": "loss",
    "remote_training" : {
      "quorum": 1.0,
      "remote_training_systems": [ { "id" : "$rts_one", "required" : true
} ]
    },
    "software_spec": {
      "name": "runtime-22.1-py3.9"
    },
    "hardware_spec": {
      "name": "XS"
    }
  },
  "training_data_references": [],
  "results_reference": {
    "type": "container",
    "name": "outputData",
    "connection": {},
    "location": {
      "path": "."
    }
  },
  "project_id": "$projectId"
}
""").substitute(projectId = PROJECT_ID,
                rts_one = wml_remote_training_system_one_asset_uid)

create_training_resp = requests.post(WML_SERVICES_URL + "/ml/v4/trainings"
, params={"version": API_VERSION},
                                     headers={"Content-Type": "application
/json",
                                              "Authorization": token},
                                     data=training_payload,
                                     verify=False)

print(create_training_resp)
status_json = json.loads(create_training_resp.content.decode("utf-8"))
print("Create training response : "+ json.dumps(status_json, indent=4))

training_id = json.loads(create_training_resp.content.decode("utf-8"))["me
tadata"]["id"]
print("Training id: %s" % training_id)

get_training_resp = requests.get(WML_SERVICES_URL + "/ml/v4/trainings/" +
training_id,
                                 headers={"Content-Type": "application/jso
n",
                                          "Authorization": token},
                                  params={"version": API_VERSION,
                                          "project_id": PROJECT_ID},
                                  verify=False)

print(get_training_resp)
status_json = json.loads(get_training_resp.content.decode("utf-8"))
print("Get training response : "+ json.dumps(status_json, indent=4))

print("WML_SERVICES_HOST = '%s'" % WML_SERVICES_HOST)
print("PROJECT_ID = '%s'" % PROJECT_ID)
print("IAM_APIKEY = '%s'" % IAM_APIKEY)
print("RTS_ID = '%s'" % wml_remote_training_system_one_asset_uid)
print("TRAINING_ID = '%s'" % (training_id))

############################### Run Bearing Party and then come back to th
is part ####################################
```

```
COS_ENDPOINT_URL = "http://s3.us.cloud-object-storage.appdomain.cloud" # G
o to your bucket and check Configurations > Endpoints

# Go into your COS instance (found in https://cloud.ibm.com/resources) and
click Credentials.
# Enter your bucket used by this project ex. myproject-donotdelete-pr-XXX.
# Expand to find the access key ID and secret access key.

COS_ACCESS_KEY_ID = "1f2246fedefe4753b1808b7d43b901c7"
COS_SECRET_ACCESS_KEY = "bf82cde58a82e4a848217245f50f59e3578492895838dee4"
BUCKET = "bearingdata1-donotdelete-pr-4wsaodz5ooiqpk" # bucket used by pro
ject ex. myproject-donotdelete-pr-XXX. To find this, go to your project, c
lick Manage > General and find your associated COS bucket

!pip install s3fs

import s3fs
import json


connection = {
    "endpoint_url": COS_ENDPOINT_URL,
    "access_key_id": COS_ACCESS_KEY_ID,
    "secret_access_key": COS_SECRET_ACCESS_KEY
}

fs = s3fs.S3FileSystem(
    anon=False,
    key=connection["access_key_id"],
    secret=connection["secret_access_key"],
    client_kwargs={'endpoint_url': connection["endpoint_url"]}
)

f = fs.open(BUCKET + "/" + training_id + "/assets/" + training_id + "/reso
urces/wml_model/request.json")
req = json.loads(f.read())
f.close()

req["name"] = "Trained Bearing Whole Data"

model_save_payload = json.dumps(req)

print ("Model save payload: %s" % model_save_payload)

model_save_resp = requests.post(WML_SERVICES_URL + "/ml/v4/models",
                                params={"version": API_VERSION,
                                        "project_id": PROJECT_ID,
                                        "content_format": "native"},
                                headers={"Content-Type": "application/json
",
                                         "Authorization": token},
                                data=model_save_payload,
                                verify=False)

print(model_save_resp)
status_json = json.loads(model_save_resp.content.decode("utf-8"))
print("Save model response : "+ json.dumps(status_json, indent=4))

model_id = json.loads(model_save_resp.content.decode("utf-8"))["metadata"]
["id"]
print("Saved model id: %s" % model_id)
```

Full party script for Federated XGBoost

```
# @hidden_cell
# The project token is an authorization token that is used to access proje
ct resources like data sources,
#connections, and used by platform APIs.

from project_lib import Project
project = Project(project_id='51864300-0251-4e7b-8c4e-c7f6e75f73d4', proje
ct_access_token='p-5dc55297565421704fbddf40089c6003ce9aa935')

############################# copy from bearing admin ####################
##############
WML_SERVICES_HOST = 'us-south.ml.cloud.ibm.com'
PROJECT_ID = '51864300-0251-4e7b-8c4e-c7f6e75f73d4'
IAM_APIKEY = 'KjO_qP6r7I-s1cm3NUkRs2smTgjqsnUXn9rQF9tG2Ezs'
RTS_ID = 'd74b89d6-0cd5-41a3-9579-59e4dfd29c89'
TRAINING_ID = '69141a68-e665-41b9-9165-cd3b3d3fdff2'



########################### download data ################################
####################

import requests

dataset_resp = requests.get("https://raw.githubusercontent.com/leonelvc/be
aringFL/main/RMS%20local.csv",
                            allow_redirects=True)

f = open('RMS local.csv', 'wb')
f.write(dataset_resp.content)
f.close()
```

```
!ls -lh

import sys
!{sys.executable} -m pip install --upgrade 'ibm-watson-machine-learning[fl
-rt22.1]'

from ibm_watson_machine_learning import APIClient

wml_credentials = {
    "apikey": IAM_APIKEY,
    "url": "https://" + WML_SERVICES_HOST
}

wml_client = APIClient(wml_credentials)
wml_client.set.default_project(PROJECT_ID)

################################################# DATA HANDLER ###########
##############################

datahandler_resp = requests.get("https://raw.githubusercontent.com/leonelv
c/bearingFL/main/custom_data_handler3.py",
                                allow_redirects=True)

f = open('custom_data_handler3.py', 'wb')
f.write(datahandler_resp.content)
f.close()



!ls -lh

import json

from pathlib import Path
working_dir = !pwd
pwd = working_dir[0]

party_config = {
    wml_client.remote_training_systems.ConfigurationMetaNames.DATA_HANDLER
: {
    "info": {
            "csv_file": "./RMS local.csv"
    },
    "name": "CustomDataHandler",
    "path": "./custom_data_handler3.py"
  }
}
```

```
print(json.dumps(party_config, indent=4))

party = wml_client.remote_training_systems.create_party(RTS_ID, party_conf
ig)
party.monitor_logs()

party.run(aggregator_id=TRAINING_ID, asynchronous=False)
```

BIOGRAPHICAL SKETCH

Leonel Villafranca was born in Matamoros, Mexico on June 7th, 1997. He attended Saint-Joseph's Academy and graduated in 2015. He pursued his educational career as an undergraduate student at the University of Texas Rio Grande Valley, where he graduated Summa Cum Laude with a bachelor's degree in Mechanical Engineering in May 2020. During his undergraduate studies he completed three internships, namely with ATD de Mexico and CEDRO de Mexico in Matamoros and with Cummins Inc. as a Manufacturing Engineering Intern in North Carolina. Additionally, he worked as a tutor for the Materials Engineering course, and was involved in volunteering activities and student organizations such as SHPE. He also served as an Undergraduate Research assistant at the University Transportation Center for Railway Safety for a year. Leonel continued his education right after obtaining his bachelor's degree and completed his Master of Science in Engineering degree in Mechanical Engineering in July 2022. Leonel can be reached by e-mail at leonel.villafranca01@gmail.com