Machine learning aided identification of train weights from railway sleeper vibration

Kaewunruen, Sakdirat; Sresakoolchai, Jessada; Thamba, Arol

DOI:
10.1784/insi.2021.63.3.151

Document Version
Peer reviewed version

Citation for published version (Harvard):
Kaewunruen, S, Sresakoolchai, J & Thamba, A 2021, 'Machine learning aided identification of train weights from railway sleeper vibration', *Insight - Non-Destructive Testing and Condition Monitoring*, vol. 63, no. 3, pp. 151-159. https://doi.org/10.1784/insi.2021.63.3.151

Machine learning aided identification of train weights from railway sleeper vibration

Sakdirat Kaewunruen [1,*], Jessada Sresakoolchai [2], Arol Thamba [3]

[1] School of Engineering, University of Birmingham, Birmingham B15 2TT, UK; s.kaewunruen@bham.ac.uk
[2] School of Engineering, University of Birmingham, Birmingham B15 2TT, UK; jss814@student.bham.ac.uk
[3] School of Engineering, University of Birmingham, Birmingham B15 2TT, UK; axt945@student.bham.ac.uk
* Correspondence: s.kaewunruen@bham.ac.uk; Tel.: +44 (0) 121 414 2670 (S.K.)

Abstract: The weight of a train is a critical factor in determining how much a train operator has to pay to use the railway network within the UK and many other countries. In this study, the different machine learning algorithms are explored to be used to predict a trains weight based on the vibration signals recorded in the sleeper of a rail track. The vibration signals are split into two groups, time domain and frequency domain signals. Then, different algorithms are developed for each group to see which domain the best representation of a vibration signal is to use within models to find out target values. From the study, machine learning has the potential to predict trains' weight effectively. This insight can lead to the use of mobile sensors in practice, such as the application of wireless accelerometers connected with a smartphone that will help engineers audit and assure the train access integrity.

Keywords: Weight Detection; Machine Learning; Deep Learning

1. Introduction

During the mid-1990s, there was a major change within the railway system in the UK. The railway system was privatised and broken down into several different private companies [1]. These companies can be classified into two groups, Train Operating Companies (TOCs) and Freight Operating Companies (FOCs). A Train Operating Company is a franchised passenger operator offering services to passengers, where as a Freight Operating Company is a freight operator which transports goods such as cereals, coal and biomass along the railway system.

The weight of a train plays an important role in determining the track access fee a train has to pay to use the railway network. In the UK, the majority of the railway network infrastructure is owned and managed by Network Rail. The track access fee is a billion-pound money generating source of income for Network Rail. In 2017-18, Network Rail received an income of £1.7 billion just from track access fees.

Currently train operators are required to fill out documentation, such as the control period 6 (CP6), provided by Network Rail and from these documentations their access fee is calculated [2]. Freight operators' access fee charges are differentiated by the weight of the train. The heavier the train, the higher charge the access fee is. This is due to the fact that the heavier the train is the more stress and

damages the train causes onto the railway network. The revenue generated from the access fee is partially used in the maintenance of the railway network.

For freight trains, the weight of the train is the critical factor in calculating the track access fee train operating companies have to pay. Due to this, freight operators could falsify their documentations and claim lower train weights to decrease their track access fee costs. Presently there is no method for Network Rail to survey moving trains within the railway network and as a result they have to assume that the freight operators are being truthful with the weights they are presenting.

Being able to predict the weight of a train from the vibration signals measured on the train track sleeper will be a major game changer as it would empower Network Rail, allowing them to be able to survey operating trains and compare the weight results obtained in the field with the weights given by the train operating companies. The proposed method in this study will empower the use of low-cost smartphone technology (in comparison with more expensive wayside train monitoring systems, WTMS) where any track inspector can use the smartphone to gauge and judge not only defect information but also estimate the actions and the weight of service trains. This will enable the improvement in assurance process of train-track interface management. This technology can be applied to ensure the system integration and supplement WTMS when malfunctioned. In addition, the goal of this development is to help engineers carry out the assurance function under extreme conditions such as floods, extreme heat, and so on. To some extent, existing sensors can no longer work in such extreme conditions. This new method will unleash the supplementary capability for rail engineers and scientists.

This study will focus on freight trains as they have a greater difference in weights compared to passenger trains. In this study, a dataset consisting of different train weights and their vibration signals will be trained into different Machine Learning algorithms, with the aim of being able to predict the train weight using the sleeper vibration signal. The study will cover different algorithms that can be used to reach our desired aim and evaluate each of these algorithms.

2. Literature Review

There are several new raising technologies which centre around detecting weights of objects whilst they are in motion. Bridge weigh-in-motion (B-WIM) is a system which focuses on detecting the weight of moving trains using railway bridges. B-WIM uses existing bridges as scales to weigh the train as they traverse through the bridge structure. Sensors are placed around the bridge and these sensors are used to record the measurement of the bridge displacement due to the axle weight of the train and also used to record the travelling speed of the train. The basic principle of a B-WIM algorithm is to calculate the axle loads by minimising the difference between the measured response and the theoretical fitted response [3].

The difference between B-WIM and the system that will be covered in this study is that the B-WIM system uses bridges as a medium to measure the train weight as a result this system can only be used on the bridges found in the rail track network.

However, the system that will be developed in this study can be used in any part of the rail track network as the sleeper part of the rail track will be the medium to find out the displacement caused by a passing train. The system covered in this study will remove the limitations of having to need a bridge in order to find the train weight. One of the disadvantages of the developed system compared to the B-WIM system is that the system has not been designed to find out the speed of the passing train and an assumption that the train will pass the sensors in the sleeper are a constant speed of 60 mph has been made.

The vibration signals data can be calculated theoretically using the Beam on Elastic Foundation (BOEF) model [4] which gives the rail displacement at a given time caused by forces against the rail at a constant velocity. This displacement can be calculated using Equation 1.

$$w(0,t) = \sum_{n=1}^{N} \frac{F_n}{2k_{system}L} e^{\frac{vt-d_n}{L}\left(\cos\left(\frac{vt-d_n}{L}\right)+\sin\left(\frac{vt-d_n}{L}\right)\right)} \qquad \text{Equation 1}$$

Where t is time, d_n is the distance between each load, v is the velocity, w(x,t) is vertical displacement (m), $k_{systems}$ is the support system modulus (N/m^2), El is the bending stiffness of the rail (Nm^2) and L is the characteristic length. The vibration signal can be found by differentiating the displacement equation which in turn will give the rate of displacement which is the vibration signal. The downside of using this method to calculate vibration signals is that the signal will not be realistic as they will not contain any noise and fluctuation that would represent real-life data.

However, there is a computer model developed by David Steffons [5] called D-Track, which can be used for the analysis of railway track dynamic behaviour. The model allows real-life simulation of railway tracks, where properties such as the sleeper vibrations caused by the train weight can be obtained. The difference between using this model and using a theoretical model such as BOEF is that the model factors in other dynamic characteristics of the rail into the calculation thus producing data which are close to being real as data you will obtain in the field. The model will allow for the generation of data without having to worry of human errors that can arise from doing calculations and may other factors can be considered using the model such as the material of the sleeper and also the spacing of the sleeper, each of these factors plays a role in the type of vibration the sleeper produces.

3. Background

This section provides an overview of the algorithms used in this study in order to help with understanding the upcoming sections.

3.1. Machine learning

The science of programming computers, so they can learn from data and act like humans do, is known as Machine Learning. The learning rate of the program is improved overtime in a self-governing fashion through the use of data obtained from observations and real-world interactions.

There are many types of Machine Learning algorithms which can be classified into broad categories [6] such as Supervised Learning, Unsupervised Learning, and Reinforcement Learning.

This study will be based around Supervised Learning, where the task is to predict a target numeric value, train weight, given a set of features, sleeper vibration. This sort of task is known as a regression. The regression models that will be covered in this study are Linear Regression, Decision Tree, K-Nearest Neighbors (KNN), Support Vector Regression (SVR), Random Forest, and Multilayer Perceptron.

3.2. Hyperparameter

Many machine learning algorithms' performance depends on their hyperparameters. Hyperparameters are variables which control the algorithm structure and also the variables which control the learning process of algorithm. Hyperparameters can be tuned to allow an algorithm to reach optimal performance. Search strategies such as Random Search and Grid Search can be used to find which hyperparameter setting gives the optimal performance for a given algorithm. Different machine learning algorithms have different types of hyperparameters to tune.

According to Probst et al. [7], there was a huge significant between the performance of an algorithm which used default hyperparameters with one which had its hyperparameter tuned. For this study, all the baseline machine learning algorithms will use the default hyperparameter values. Once the best performing algorithms have been found, they will undergo hyperparameter tuning to further increase their performance before being evaluated using the test dataset.

3.3. Grid Search and Random Search

Grid Search is an example of a process which can be used to configure optimal hyperparameters for a given model. The user is required to specify a finite set of hyperparameter values and the Grid Search program iterates through every possible combination of the hyperparameter values and stores a model for each combination. The model which performs the best can be considered the model with the best hyperparameters for the given algorithm. Grid Search can be applied across all machine learning algorithms. It will be used in this study in tuning the hyperparameters of the best baseline model produced in this study.

Random Search is an alternative to Grid Search. It can also be used to configure optimal hyperparameters for a given model. Random search randomly searches samples configurations until a certain budget for the search is exhausted. Random Search can be used instead of Grid Search when some of the hyperparameters to be tuned are more important than others [8].

3.4. K-Fold Cross Validation

A commonly used technique for evaluating Machine Learning models with small dataset is cross-validation. Cross-validation is a powerful preventative measure against withholding data from the training set and also against overfitting.

Cross-validation works by splitting the available input data into partitions consisting of a training subset and a complementary testing subset. The Machine Learning model is trained using the training subset and is validated using the testing subset.

In this study, K-fold cross-validation is used which works by splitting the input data into k subsets of data and training the Machine Learning model on all but one of the subsets and then evaluating the model using the unused subset. This process is repeated k times using the same procedure reserving a different k subset for evaluation each time.

An average performance score can then be obtained using all the different process repeats and then used to compare the performance of different Machine Learning models.

3.5. Performance measure

In machine learning, the performance measure is used to compare the trained model predictions with the testing data and the actual observed data. In other words, performance measures are used to estimate the accuracy of a model on future data. For regression type models, there are three main performance measures [9] which are Mean Absolute Error, Mean Squared Error, Root Mean Squared Error (RMSE), and R Squared (R^2).

This study will use the RMSE and R^2 to evaluate the performance scores of all the machine learning algorithms that will be trained. The RMSE is chosen as it is a commonly used performance measure for regression problems which gives a sense of how much error the model typically makes in its prediction and R^2 is chosen as it indicates how close the predicted values are to the actual values. The Mean Squared Error is not suitable for dataset as it does not work well with data which contain a lot of noise and Mean Absolute Error is not very sensitive to outliers compared to the RMSE since it does not assign higher weights to larger errors [10].

3.6. Tools used

There is a numerous amount of programming languages that can be used to build Machine Learning models such as Java and Python. Java has the WEKA toolbox which can be used to generate a general Machine Learning algorithm, whereas Python has a numerous number of frameworks, such as Scikit Learn, Keras and Tensor flow, to build Machine Learning models. In this study, Python is used because it is flexible, stable and includes predefined libraries and frameworks to simplify the development process. Python fits these criteria, as it is one of the most popular languages used for Machine Learning with numerous libraries and frameworks built around machine learning to make the process as smooth, easy and time-efficient as possible. Python is used as the programming language with Scikit Learn and Keras as the machine learning frameworks.

3.7. Wilcoxon Signed-Rank Test

Wilcoxon Signed-Rank Test is a non-parametric statistical hypothesis test used to compare two related samples such as the same algorithm evaluated on different datasets or different algorithms evaluated on the exact training data. Wilcoxon p-value is calculated using a built-in class in Python using the following hypothesis for all the tests carried out.

- Null Hypothesis (H0): Sample distributions are equal
- Hypothesis One, Reject H0 (H1): Sample distributions are not equal

The results of this test will interpret whether or not that the samples are drawn from different distributions. If the p-value is less than 0.05 then the null hypothesis can be rejected and H1 can be accepted.

4. Methodology

This section explains the procedure that will be carried out in this research in order to reach the aim of producing Machine Learning models to predict the weight of a train from sleeper vibrations.

First, training set data is generated by using D-Track. The output is the raw data, which is already in the time domain. Then, the frequency domain data is generated using the Fourier Transform. Data is split into training set, validation set, through the use of K-Fold validation, and a test set.

The time-domain data will not undergo anymore data processing and can be fed directly into the machine learning algorithms. However, the frequency domain data will undergo feature scaling before being passed onto the machine learning algorithms.

A benchmark model, baseline zero, is created for both the time and frequency domain dataset. This model will serve as a baseline and its performance will be used to compare the effectiveness of the other machine learning models. Then, two models from each category are selected for further development and hyperparameter tuning to ensure it fits best with the problem in hand. Last, a final performance measure is then carried out using the test set on the two new improved machine learning models.

5. Data Collection and Pre-processing

This section covers how the data used in the machine learning algorithms is acquired and pre-processed.

5.1. Dataset

The dataset used to train the machine learning algorithms will be based on trains of different weights and the vibration signals they induced onto the train track sleeper. The vibration signals can be classified into two categories, Time Domain and Frequency Domain.

The time-domain dataset consists of the raw data of 616 train weights with their corresponding vibration signals. As the name suggest, the parameters of these data are of time against acceleration/amplitude. The time-domain vibration signal

consists of 2,418 timesteps within a given time length of 1.21 seconds, with each timestep being equivalent to 0.0005 seconds. This dataset is collected using D-Track.

The frequency-domain dataset consists of the manipulated time-domain dataset. As such, there is also a total of 616 train samples, each with their own individual frequency against power spectral density data. This dataset is collected through the use of mathematical calculations, converting time into frequency using fast Fourie transform.

5.2. Data Collection

The raw time-domain dataset is generated using D-track. D-Track is a program which can be used to generate dynamic behaviour railway track data, in our case we will be using it to generate the vibration at the rail track sleeper for a given weight. A simple methodology is used to generate the data using a basic understanding of freight trains.

On average freight train consists of over 100 wagons with 3 or 4 locomotives running the train. To generate the dataset, the train is assumed to consist of 1 locomotive and a maximum of 22 wagons. The train weight data is generated using the combination of the nominal tare mass (the weight of the locomotive plus the weight of the empty or completely filled wagons) and the wagon carries mass, as the total train weight.

A systematic approach is used to calculate the total train weight, the initial total train weight is assumed to consist solely of the mass of the locomotive and the mass of one empty wagon, and the vibration signal at the sleeper is generated and stored for this weight. Then the carry mass of the wagon is increased by a set amount of 5 ton or 0.5 ton and the vibration signal for this new weight is generated and collected.

According to Barkan [11], the weight of a modern freight locomotive is 196 tons and the weight of an empty railway car is 33 tons with a maximum carry weight of 110 tons. This led to the dataset consisting of trains weighing from 229 tons to 3,342 tons with a total of 616 weight samples. In Figure 1, it shows the vibration signal produced using D-Track for a given train weight of 229.

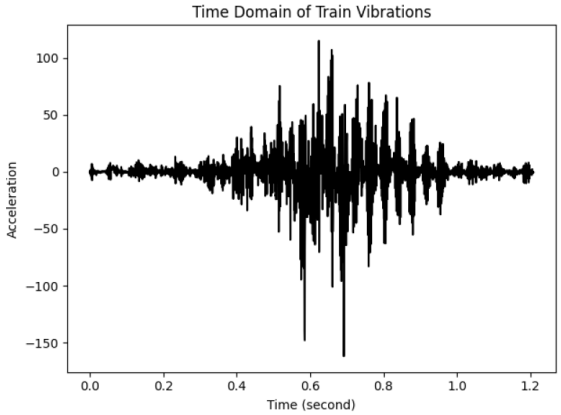

Figure 1. Time-domain train vibration

5.3. Pre-processing

The noise within the vibration data can be removed which may lead to an improvement of the performance of the machine learning algorithm. With the reduction of the noise, the data will not contain unwanted information within the data. The noise within the data could lead to overfitting due to the algorithm training itself around the noise pattern produced or lead to a huge model runtime due to the large size of the data. However, noise reduction can lead to important information within the vibration signal to be lost thus leading to poor quality data, if not done correctly. To prevent the loss of vital information from the dataset, the machine learning algorithms are trained using the unprocessed raw time-domain data.

The time-domain data is represented using an array, where each weight is represented by two arrays. One array contained the mass of the train whilst the other contained the acceleration/amplitude points, starting from 0 seconds to 1.21 seconds. This leads to the array representing the acceleration/amplitude points to be of length 2418, where the index of the array represented the timestep. Therefore, each index of the array represents a timestep increase of 0.0005 seconds resulting with the time domain data having a huge matrix dimension as such, no visual representation of the regression data can be created. Representing the data into an array makes it easier for the information to be passed onto the machine learning models.

Using a similar procedure as Liu and Yang [12], the time domain data is used to create frequency domain data. The time-domain data is converted to the frequency domain to produce machine learning model based on the characteristics of the signal that are not easy to see whilst looking at the signal in the time domain. Machine learning algorithms will be created for the two types of domain and a comparison will be made. The difference between the two datasets is that frequency domain data shows how much of the signal lies within each given frequency whereas the time domain data shows how a signal changes over time.

The frequency-domain data is calculated using a Fast Fourier transform. Fast Fourier transform is used to represent a signal from its original domain, such as time, to the frequency domain as Equation 2.

$$X_k = \sum_{n=0}^{p-1} x_n W^{nk}, W = e^{-j2\pi/p} \qquad \text{Equation 2}$$

Where x_n is the input signal data, p is the number of signal samples and X_k is the consecutive samples in the frequency domain [13]. Once the frequency domain is found, the power spectral density is also calculated. For a given signal, the power spectral density describes the distribution of power from the frequency components. Figure 2 shows a frequency domain data.

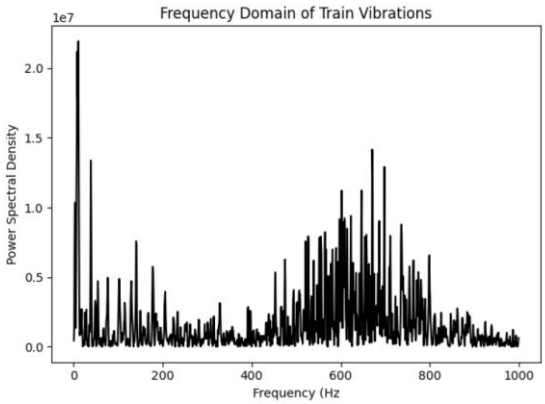

Figure 2. Frequency Domain train vibration

From Figure 2, it can be seen that there is a huge difference between the scaling of the power spectral density values at different frequencies. Machine Learning algorithms do not perform well when there is a very different scale between the attributes. To get the values of the power spectral density within the same scale, the data will need to undergo feature scaling. Standardization will be used to feature scale our frequency domain data. Standardization works by subtracting each value with the mean value and then dividing it by the standard deviation resulting in uniformity within the data.

The two datasets, time and frequency domain, are separated into 3 groups to be used for training, validation and testing in the machine learning algorithms. Ten percent of the total data is used for testing, whilst another ten is used for validation and the remaining data is used for training the algorithms.

6. Optimised algorithm design

This section will cover the design of the machine learning algorithms which are selected to have their hyperparameters tuned in order for them to reach optimal performance. The two algorithms that are selected to be tuned are the Multilayer Perceptron and the Random Forest.

6.1. Multilayer Perceptron Design

Multilayer Perceptron (MLP) consists of three layers. The first layer is the input layer, where the feature vector is passed through. The second layer is the hidden layer, consisting of one or more threshold logic unit layers. It is within this layer where the weighted sums of the inputs are calculated, and a step function is applied to it before being sent off as an output through the use of the Rectified Linear (ReLU) activation function which provides non-linearity in the network. Finally, the output layer which produces the output variables.

6.1.1. Baseline MLP Model Design

A 2-layer MLP model is used as a baseline to see how well dataset performed within a neural network algorithm. A 2-layer MLP consists of an input layer, one hidden layer and one output layer which is represented by the notation of 2418/100/1 for the time domain dataset. Since the network consisted of 2418 variables in the

input layer, one hidden dense layer with 100 nodes and an output layer with one node. The frequency-domain dataset is represented with the notation 1208/100/1 as it has a different about input layer variables.

The default Keras hyperparameter values are used for the baseline MLP models, which consists of using the default values of the efficient Adam optimization algorithm and mean squared error loss function. However, there is no default value for the number of neurons within the hidden layers so a random value of 100 neurons is chosen and for the output layer, only one neuron is required with no activation function as the model is targeted towards a regression problem.

The neural network design of the baseline MLP model for both the time domain and frequency domain is the same, with the sole expectation of the number of input variables. The design of the network for the time domain dataset can be seen in Figure 3 and the frequency domain can be seen in Figure 4.

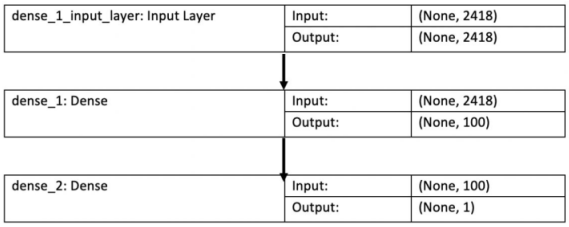

Figure 3. Neural Network Design of Baseline Time Domain MLP model

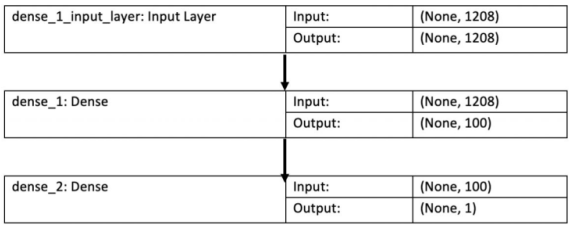

Figure 4. Neural Network Design of Baseline Frequency Domain MLP model

6.1.2. Optimized MLP Model

To ensure that the MLP model runs on optimal performance, some of the hyperparameters of the MLP neural network are tuned. The two main hyperparameters that control the architecture of the neural network are the Number of layers and Number of nodes in each hidden layer.

Aside from these two hyperparameters, the Adam learning rate is also tuned as it also plays a role in determining the training error of the network. A large learning rate may result in gradient descent increasing the training error rather than decreasing it and a small learning rate results with slow training time.

Random Search is used to find the best hyperparameter values for both the individual time and frequency domain MLP final models. A range of values are given for each of the hyperparameters and the Random Search algorithm finds out the best combination between these values of the number of layers, the number of nodes and the values of Adam learning rate which produced the least training error thus best performance score. The Random Search algorithm returns the best values to use for the hyperparameters and using these values, a final MLP model is created for both the time and frequency domain. The final network design for the time

domain can be seen in Figure 5 and that for the frequency domain can be seen in Figure 6.

The use of Random Search to tune the hyperparameters saves time and also prioritises producing a model complexity which is centred on the performance of the algorithm rather than the algorithm runtime. If Random Search is not available then the hyperparameters will be chosen manually, varying each hyperparameter one by one in different combinations with the other hyperparameters and comparing the changes in the performance till the best combination is found. This will have consumed a huge amount of time.

The Adam learning rate for the time-domain model is found to be 5e-5 and for the frequency-domain, it is 0.0001. These values are found to work well with the number of layers and number of neurons assigned to each of the respective MLP model.

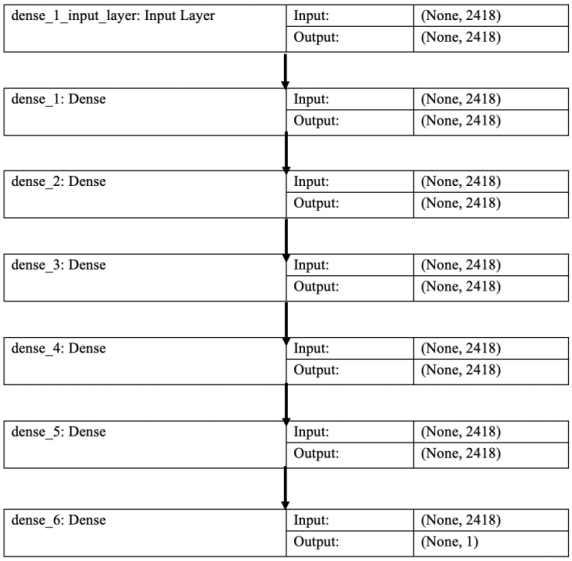

Figure 5. Final MLP Model for Time-Domain dataset

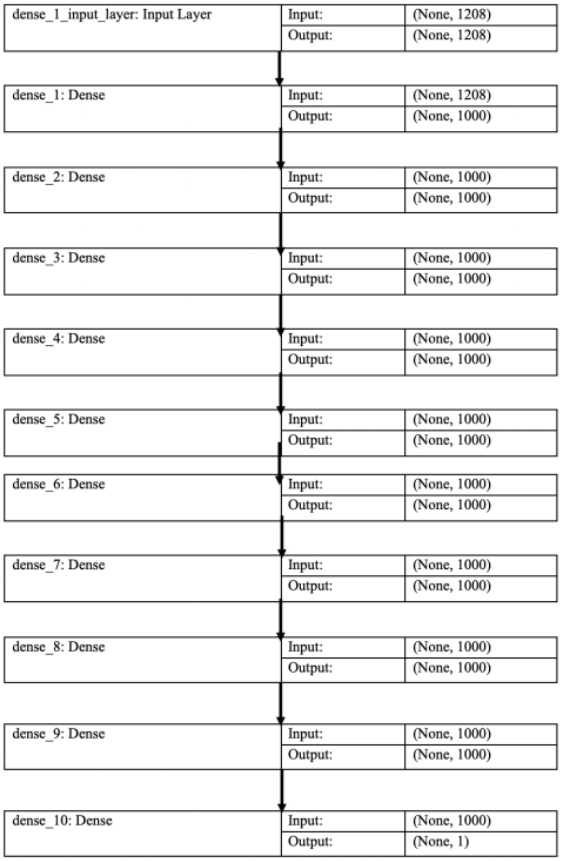

Figure 6. Final Frequency-Domain MLP Model

6.2. Random Forest Design

The Random Forest algorithm has several hyperparameters which can be tuned in the effort to try improving the performance of the algorithm. To avoid a huge training time using the search algorithm or Grid Search, it will be focusing on three hyperparameters to tune which are Bootstrap, N_estimators, and Max_features.

Bootstrap decides whether bootstrap samples are used when building the trees or when set to fault the whole dataset is used to build each tree. N_estimators decides the number of trees that will be built in the random forest and max_features decides the number of features to consider when looking for the best split.

The default Random Forest used in the baseline models has bootstrp set to true, n_estimators set to 100, and max_features set to 10. After using Grid Search for both the time and frequency domains, it is found that the following hyperparameters are ideal for the final model for each type:

- Time domain: bootstrap = false, max_features = 20 and n_estimators = 73
- Frequency domain: bootstrap = false, max_features = 20 and n_estimators = 200

7. Results and Discussion

In this section, different algorithms are compared and evaluated. The training set consisted of 80% of all the data, the validation set consisted of 10% whilst the test set consisted of the remaining 10%.

7.1. Time Domain Baseline Models Result

The performance of models can be shown as Table 1.

Table 1. Time Domain Baseline model results

Model	Training Set		Validation Set	
			RMSE	
	R^2	Average RMSE	Average	Standard Deviation
Baseline Zero	-	-	912.33	912.33
Linear Regression	1.0	1.13	68.40	5.70
Support Vector Regression (SVR)	1.0	0.01	68.40	5.57
Decision Tree	1.0	0.00	22.51	3.26
K Nearest Neighbor (KNN)	0.084	872.40	1101.84	62.15
Random Forest	0.999	2.98	7.75	1.84
Multilayer Perceptron (MLP)	0.998	41.73	92.23	2.0

The Baseline Zero model is a sanity check. It is used as a predictor to compare how relatively well the other models do. The Baseline Zero is a prediction model that will be used if there is no machine learning available, in other words, it predicts the weight of the train without using the time/frequency domain data. For the Baseline Zero, the weight of the train is predicted to be equal to the mean train weight of all the samples used in the validation set. If a model has an RMSE score that is higher than the Baseline Zero's RMSE then it means that the model performed worse than a prediction thus would be no better than just using the mean train weight for all the predicted train weight values.

From Table 1, it can be seen that only the KNN model performed worse than the Baseline Zero model in the validation set. One of the reasons for this is that the accuracy of a KNN model tends to decrease with an increase in the number of k neighbours around a given point in the model. The dataset used in training the model consists of over 2,400 points for a given single train weight resulting in a huge k neighbour value thus leading to poor accuracy for the model. The dataset used is clearly not suitable for the KNN model especially with its hyperparameters being untuned.

The R^2 score provides a "goodness of fit" measure of the predicted line of best fit to the actual value. KNN has an extremely low R^2 compared to the other models and the effects of this can be seen with the average RMSE score in the validation set. The prediction does not fit in well with the training set data. From the boxplots in Figure 7, it can be seen that the boxplot of the KNN results can be considered as an outlier compared to the other model due to its underfitting of the training data. Underfitting can be improved by either increasing the number of samples used in the training data, tuning the hyperparameters to produce predictions that better fit the data, or simply using another algorithm.

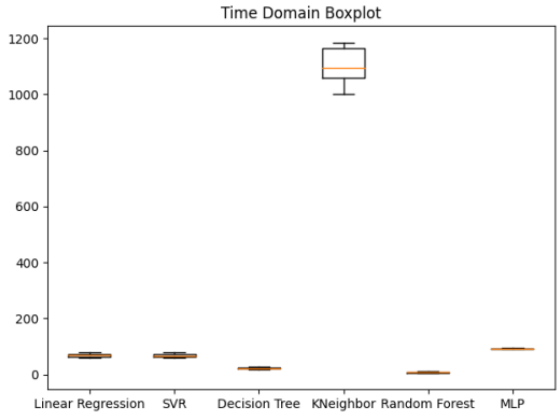

Figure 7. Time-domain results shown in a boxplot diagram

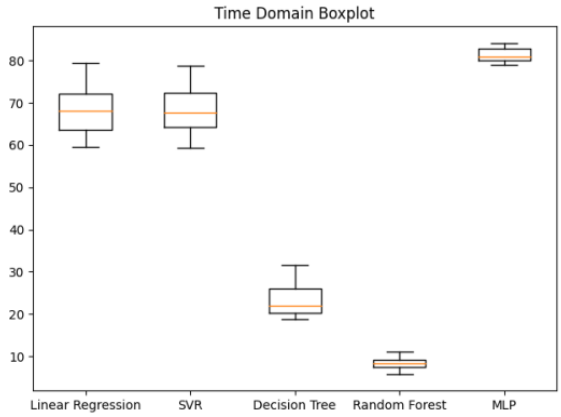

Figure 8. Time-domain results excluding the KNN model

Accepting that the KNN model is a poor choice for the given dataset and removing it from the boxplot diagrams it can be seen a better relationship between the rest of the algorithms created, as shown in Figure 8.

Excluding KNN, the other models have great R^2 scores with values close to one thus meaning that the prediction fit of the training set is very close to the observation. It can be clearly seen that as the R^2 score for the training set increase the mean RMSE score for the training set got closer to zero. For RMSE, the closer the score is to zero the better the accuracy as there is less error between the predictions and the observations. The best performing algorithm is the Random Forest with a training set RMSE score of 2.98 and a validation RMSE score of 7.75; whereas MLP, excluding the KNN, performed the poorest with a training set score of 41.73 and validation score of 92.23. Compared to the remaining algorithms, excluding KNN, the Decision Tree, MLP and Random Forest algorithms seem to perfectly fit the training data without any overfitting and thus generalise well with new data. This can be seen from the relatively low difference between the RMSE values obtained training and validation set. Linear Regression and SVR both seem to overfit the training data, and this can be seen from the large difference between the training and validation RMSE scores. When a model is too complex relative to the amount and noisiness of the training data, overfitting occurs. It can be resolved by removing the noise in the training data, simplifying the model via reducing the number of attributes in the training data or gathering more data. The training data set which used can be considered to contain a lot of noise and also have a lot of attributes and

this may be the reason why overfitting is observed in some algorithms. The amount of training data used is not also ideal quantity wise in producing high-quality models and is thus also another factor which lead to the overfitting of algorithms.

From the results, it can be clearly seen that the Random Forest algorithm performed the best even though it uses default hyperparameter values. Based on its performance, it is chosen to be selected to undergo hyperparameter tuning to see if its performance can be further enhanced. MLP is also chosen to be tuned even though it performed relatively poorly compared to the other algorithms, but it has a small standard deviation between the average RMSE value and there are not any signs of overfitting thus the algorithm generalises well when it comes to new data. The reason for its poor performance can be due to the simplicity of the network architect used in the baseline model and also due to the small training data sample size. The MLP algorithm has a huge potential when it comes to enhancing its performance via hyperparameter tuning.

7.2. Frequency Domain Baseline Models Result

The performance of models can be shown as Table 2.

Table 2. Frequency Domain Baseline model results

Model	**Training Set**		**Validation Set**	
			RMSE	
	R^2	Average RMSE	Average	Standard Deviation
Baseline Zero	-	-	912.33	912.33
Linear Regression	1.0	2.28	604.65	138.26
Support Vector Regression (SVR)	-1.379	1406.33	2349.66	376.90
Decision Tree	1.0	0.00	133.56	12.91
K Nearest Neighbor (KNN)	0.935	232.19	291.37	19.71
Random Forest	0.998	31.72	85.81	7.43
Multilayer Perceptron (MLP)	0.128	987.47	959.31	78.58

From the results of the frequency domain algorithms, it can be clearly seen that the performance of the algorithms is generally worst compared to their respective algorithms in the time domain. Two algorithms, SVR and MLP, performs worse than the Baseline Zero algorithm, which is the terrible predictor. These two algorithms underfit the training data thus meaning that either there is not enough training data to effectively train the algorithms or that the algorithms are not suitable at all for the frequency domain dataset. The MLP has an extremely low R^2 score and its effect can be seen on the RMSE score as the training error measured in the RMSE is huge due to the predictions not being a good fit for the observations. The SVR has a negative R^2 score which means that the algorithm is doing worse than the mean value and this can be seen from comparing it to the Baseline Zero RMSE score, where the SVR RMSE score is more than twice bigger than the Baseline Zero RMSE score in the validation set.

The Linear Regression algorithm performs well in the training set but poorly in the validation set. The RMSE score in the validation set is slightly lower than the Baseline Zero and there is a huge difference between the training set RMSE, and the validation set. RMSE clearly shows that there is overfitting in the algorithm.

Overfitting also occurs in the Decision Tree algorithm and the Random Forest algorithm, which perform the best in comparison to the other algorithms. The Decision Tree has the same performance in the train set as it did for the time domain dataset. However, there is a huge difference between the RMSE values obtained in the validation set. This shows that an algorithm can perform well in the training set but may not generalise as well when it comes to new data presented in the validation or even test set.

The KNN algorithm performs much better with the frequency domain dataset compared to the time domain dataset and this can be explained due to the fact that there are fewer attributes used in the frequency domain data set thus reducing the number of k neighbours in the algorithm. In KNN, the higher the number of k neighbours, the lower the accuracy of the algorithm. A clear comparison of the frequency domain algorithms can be seen in Figure 9, from which it can be seen that the SVR algorithm not only has the worst mean RMSE score but also a huge standard deviation between the scores recorded for it.

The same two algorithms are used as the time domain dataset to undergo hyperparameter tuning. The Random Forest performs the best in both the time and frequency domain datasets, so it makes sense choosing it to further undergo tuning. However as stated before, even though the MLP does not perform as well as the Random Forest or even the Decision Tree, there is still a huge potential in improving the performance score once it undergoes tuning.

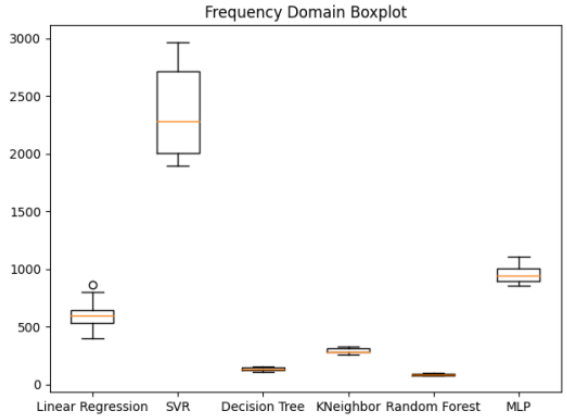

Figure 9. Frequency domain results shown in a boxplot diagram

7.3. Final Tuned Models Result

The performance of models can be shown in Table 3.

Table 3. Tuned Time Domain Models Results

Model	**Training Set**		**Validation Set**		**Test Set**	
			RMSE		**RMSE**	
	R^2	Average RMSE	Average	Standard Deviation	Average	Standard Deviation
Random Forest	1.0	0.0	85.61	1.51	73.49	4.19
Multilayer Perceptron (MLP)	0.999	13.60	55.80	6.25	57.17	5.40

The results for both the tuned algorithms for each dataset can be found in tables 3 and 4. For the time domain dataset, it can be seen that the Random Forest algorithm

performs worse compared to the untune Random Forest algorithm. This simply means that the default hyperparameter values suited the dataset more. As for the MLP algorithm, the tuned model performs better in both the training set and validation set compared to the untuned model. For the training set, there is a decrease in the average RMSE which resulted in a decrease in the average validation set RMSE. This indicates that the new hyperparameters obtained using random search are able to optimise the performance of the model. The test set average RMSE results for both models are not too far off from the validation RMSE score, however, there is clear evidence of overfitting occurring within these models since there is a huge difference between the test RMSE scores with the training set RMSE score. As previously mentioned, one of the main reasons why models keep showing signs of overfitting is most likely due to the size of the used dataset. Dataset size can be considered small and not of the ideal size to produce precise and reliable machine learning models.

Table 4. Tuned Frequency Domain Results

Model	**Training Set**		**Validation Set**		**Test Set**	
			RMSE		**RMSE**	
	R^2	Average RMSE	Average	Standard Deviation	Average	Standard Deviation
Random Forest	1.0	0.0	77.61	0.62	59.66	1.73
Multilayer Perceptron (MLP)	0.999	57.40	231.58	20.40	250.03	11.95

For the frequency domain tuned algorithms, it can be seen that both algorithms perform better than the untune models thus meaning that the optimisation carried out using grid search and random search is a success. The RMSE score in the validation set for the MLP is decreased by nearly a factor of four and this can evidently be seen from the increase of the R^2 score which goes from being close to zero to being close to one. The changes in the hyperparameter for the MLP model allows for the predictions of the training set to fit much better with the real data observation. However, there is still overfitting occurring in both algorithms. If there are more training samples available, it should result in a greater improvement in the RMSE scores.

7.4. Wilcoxon Test

Several statistical hypothesis tests are carried out to see if there is any clear distinction between the distribution of the final two models. The following tests are carried out:

1) Test 1 – Test to see if there is the same distribution between the tuned MLP model and the Random Forest model for the time domain dataset
2) Test 2- Test to see if there is the same distribution between the time domain dataset untune MLP model and the tune MLP model
3) Test 3 - Test to see if there is the same distribution between the frequency domain untune Random Forest model and the tuned Random Forest Model

4) Test 4 - Test to see if there is the same distribution between the same MLP algorithm but for the different dataset
5) Test 5 - Test to see if there is the same distribution between the same Random Forest algorithm but for the different dataset

The Wilcoxon Signed-Rank Test is used for all the five tests and the p-value calculated for each test is 0.00008857, which is lower that the needed 0.05 p-value to indicate that the distribution is the same. This small p-value shows that the idea that the difference for each test is due to chance can be rejected and that each model tested in the different tests has a clear possible distribution. From the results, it can be seen clearly that the time domain dataset performed better than the frequency domain dataset.

8. Conclusion

8.1. Achievements

The aim of this study is to produce machine learning algorithms capable of predicting the train weight using the vibration signals measured in the sure. Using D-Track, dataset of vibration signals is generated and measured at the sleeper to use for our machine learning algorithms. The data generated are real-life representations of the type of signals which gain from rail track. Using this data, the vibration signals are processed into the time domain signal and the frequency domain signals which are then used to train machine learning algorithms.

Six machine learning algorithms are developed for each type of domain, which are all successfully able to predict the train weight. Some algorithms perform better than others and the best algorithm, the Random Forest algorithm, is selected for further tuning in hopes of improving the performance. The MLP algorithm is also chosen, as it is a neural network algorithm which has a huge potential of performance improvement from having its hyperparameters tune. The performance of the MLP algorithm for both datasets can be improved through hyperparameter tuning but the performance improvement for the Random Forest algorithm varies for each dataset. This insight can lead to the use of mobile and wireless sensors such as accelerometers, laser droppers, laser systems to help engineers quantify the train weights in the field as part of systems audit and assurance.

8.2. Future Work

Some overfitting can be observed in results. One of the ways to overcome overfitting is by increasing the data size, so in the future if this study is to more dataset from the fields. The architect of the MLP algorithm and the Random Forest algorithm can be improved, by using a much more detailed search algorithm which takes in more hyperparameters into consideration and also covers a greater range of possible hyperparameter values. Due to the higher training time, only a small narrowed hyperparameter test is carried out and the effects of this can be seen in the Random Forest for the time domain results, for example, are the optimal hyperparameter values does not perform as well as the default hyperparameter

algorithm. The time-domain data can of underwent more pre-processing such as signal denoising before being used to obtain the frequency domain dataset. This will have removed the unwanted frequency bands within the frequency data, giving a frequency dataset which contains nothing but the important features which are distinct for each weight without the noise pattern which are identical in each given data.

ACKNOWLEDGMENT

The authors also wish to thank the European Commission for the financial sponsorship of the H2020-RISE Project no.691135 "RISEN: Rail Infrastructure Systems Engineering Network", which enables a global research network that addresses the grand challenge of railway infrastructure resilience and advanced sensing in extreme environments (www.risen2rail.eu).

References

1. Reform, R., *Charges for the Use of Infrastructure. ECMT*. 2005, Published by OECD Publishing.
2. Office of Rail and Road, *UK rail industry financial information 2017-18*. 2019.
3. Žnidarič, A., et al., *Railway bridge weigh-in-motion system.* Transportation Research Procedia, 2016. **14**: p. 4010-4019.
4. Powrie, W., *A Guide to Track Stiffness: August 2016*. 2016: University of Southampton Department of Civil & Enviromental Engineering.
5. Steffens, D.M., *Identification and development of a model of railway track dynamic behaviour*. 2005, Queensland University of Technology.
6. Géron, A., *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. 2019: O'Reilly Media.
7. Probst, P., A.-L. Boulesteix, and B. Bischl, *Tunability: Importance of Hyperparameters of Machine Learning Algorithms.* J. Mach. Learn. Res., 2019. **20**(53): p. 1-32.
8. Feurer, M. and F. Hutter, *Hyperparameter optimization*, in *Automated Machine Learning*. 2019, Springer, Cham. p. 3-33.
9. Bowles, M., *Machine Learning with Spark and Python: Essential Techniques for Predictive Analytics*. 2019: John Wiley & Sons.
10. Satyan, S. *What is Mean Squared Error, Mean Absolute Error, Root Mean Squared Error and R Squared?* 2019; Available from: https://www.studytonight.com/post/what-is-mean-squared-error-mean-absolute-error-root-mean-squared-error-and-r-squared.
11. Barkan, C. *Introduction to Rail Transportation*. 2012. Railroad Engineering Education Symposium (REES).
12. Liu, J. and X. Yang, *Learning to see the vibration: a neural network for vibration frequency prediction*. Sensors, 2018. **18**(8): p. 2530.
13. Nussbaumer, H.J., *The fast Fourier transform*, in *Fast Fourier Transform and Convolution Algorithms*. 1981, Springer. p. 80-111.