**MAE 593: Applied Project Report**



**Optimization of Intentional Mistuning of Bladed Disks Using Machine Learning Techniques**

**By**

Rohit Venkataraman Iyer (1219868481)

**Mentor:**

Dr. Raghavendra Murthy

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| Index | Topic | Page |
| 1 | Introduction | 3 |
| 2 | Optimization using different Machine Learning models | 6 |
| 2.1 | Linear Regression (LR) | 6 |
| 2.2 | Neural Network (NN) – Fitrnet | 6 |
| 2.3 | Automatic Regression tuning - Fitrauto | 10 |
| 2.4 | Gaussian Process Regression (GP) – Fitrgp | 11 |
| 2.5 | Binary Tree (BT) – Fitrtree | 13 |
| 2.6 | Support Vector Machine (SVM) – Fitrsvm | 14 |
| 2.7 | Ensemble Methods (EM) – Fitrensemble | 15 |
| 2.8 | Gaussian Kernel (GK) – Fitrkernel | 16 |
| 3 | Observations | 18 |
| 4 | Conclusions | 20 |
| 5 | References | 21 |

1. **Introduction**

Over the past few decades, it has been well demonstrated that mistuning, which refers to random variation of blade-to-blade properties of a bladed disk as a result of manufacturing, in-service wear and tear, etc., can affect bladed disk performance dramatically, resulting in potential large increases in the forced response when compared to bladed disk performance that has been tuned. The large sensitivity of bladed disks has also been shown to be closely related to their cyclic symmetry[1], [2]. It has been proposed to design the bladed disks with nonidentical blades, i.e., to break the cyclic symmetry property of tuned bladed disks with a preset mistuning pattern to reduce sensitivity to the small, random mistuning described above. In most cases, such designed variations between blades are referred to as intentional mistuning. Thus, the desirable role of intentional mistuning is to interact with random mistuning to potentially mitigate its negative effects[3], [4].

At first, a detailed literature review was done to get insight into the current work done in intentional and random mistuning and about machine learning application in this area. There were three different kinds of optimization strategy involved in its optimization especially given the large number of local minimum amplification factors present and the high cost of function evaluation for its calculation[1]. The first optimization technique was created and extensively validated on a single-degree-of-freedom-per-blade-disk model as well as a reduced-order model of a bladed disk. This two-step strategy is based on the solution of an optimization problem over a subspace in which two successive blades exhibit one of two predefined groupings, either AA and BB or AB and BA. Because the subspace is much smaller than the full space, an exhaustive search is usually possible. Several of the best subspace patterns were then used as initial guesses in a local search in the entire space until convergence occurred. This strategy produced excellent outcomes in a wide range of circumstances.

The second optimization strategy was presented and verified, in which the initial guesses for a full-space local search with random mistuning are acquired from the optimization problem without performing random mistuning. This latter problem is substantially less computationally demanding because it does not require any Monte Carlo simulation. The validation of this second strategy indicated that it is also valuable and efficient, probably somewhat faster than the subspace approach. It was also observed that co-incidentally, in the machine learning models trained on just the intentional mistuning data provided the best results and took the least time to train and tune.

Finally, the third optimization approach was introduced and briefly validated which combined the two previous strategies, namely, using the subspace concept in conjunction with the intentional mistuning only problem to obtain the five best initial patterns for the local search conducted with random mistuning present. The results verified the dependability and efficiency of this third strategy.

It was noticed that there was not much research and work done to address the problems with it, using machine learning to predict the data instead. Among the few research papers that were found Genetic algorithms were the most popular and common techniques used to address the high cost of evaluation of the exhaustive methods discussed above[5]. Finite element methods (FEM) were another potential solution that was proposed for this optimization[6]. To begin the sensitivity analysis, the applicable reduction approach is presented first. This reduction method allowed the frequency response function (FRF) of a Finite Element Model (FEM) to be calculated in a reasonable amount of time. The sensitivity of the vibration amplitude to normally distributed input parameters is then evaluated using the Taylor series approximation, and the maximum amplitude can thus be estimated. The single frequency response function calculation yielded results which were in good agreement with the findings of over 1000 Monte-Carlo simulations[6]. Genetic optimization based on modelling the system based on an ant colony was technique that was employed which combined the benefits of ant colony algorithm and genetic algorithm[7]. Using this method, both the local and global search for the minima was possible and the disadvantage of using a single algorithm is overcome. Parallel computing was later employed with the addition of a GPU[8] which improved the computational speed and thus the efficiency of the model. This technique was also employed in this project for optimization of each machine learning model to speed up and explore more hyperparameter tuning options. Thus, using these optimization algorithms improved the local and global search capability while the incorporation of CUDA based parallel computing framework improves the computing speed[9].

Since this problem is essentially a combinatorial problem in nature, several techniques for combinatorial problems were also explored. Graph based neural networks or Graph Neural Networks (GNN)[10] was also a machine learning model that was found. A GNN takes a graph as an input unlike traditional neural network models and provides an optimized graph as an output. Given our input features, the 12 blade mistuning patterns are interdependent, and their collective arrangement is what provides us with the output, amplification factor, this theory made sense but was not tested due to its complexity. The family of models like GNNs were identified which address the combinatorial problems in which the message-passing algorithm is used are neural networks[11], recurrent relational networks[12], graph networks[13] but the pioneer model for this was the graph neural network[14]. To train such message computing and updating modules, MLPs and RNNs respectively, the Stochastic Gradient Descent algorithm was implemented via TensorFlow’s Adam optimizer. Relu was the most widely used activation function used in such applications and it also was the best activation function identified for our case after extensive tuning. But a common drawback that was noticed in these techniques was that there were a lot of integers to integer (binary) mapping algorithms available but none of them linked a binary value to float value, which was what we needed for our case.

For us to start with the optimization using machine learning techniques, we need to first understand the data generated from intentional and random mistuning as the dataset is the corner stone of any machine learning algorithm. For a bladed disk with N blades, the number of intentional mistuning patterns (with 2 different blade types) is 2N/N[1]. This results in a combinatorial explosion of intentional mistuning patterns as the number of blades increases. which results in an exponential increase of computational time and restrictions due to limitations depending on the hardware available for its computation thus resulting in exhaustive search to be very expensive. For this project, we shall consider a 12 bladed disk with two different types of blades chosen with their weights being 5% above and below the mean weight to form our A/B blade types. First, the 352 intentional mistuning patterns will be tested to find their corresponding amplification factor which will form our Int dataset. Later, random mistuning was introduced to enrich our dataset by performing random mistuning with the help of Monte Carlo simulations which provide us with 1000 random mistuned patterns for every intentional mistuning pattern. This results in the generation of the rand dataset which contains 352,000 different patterns which is a huge dataset for a machine learning application. To simplify things are reduce the computational burden the rand dataset introduced; two additional datasets were created from the rand dataset which involved choosing 1 random pattern out of every 1000 patterns to ensure we had one random mistuning pattern for each intentional mistuning pattern which gave us Randc1 dataset which now contains 352 different entries. The same was done again, this time choosing 5 random patterns for every 1000 patterns to generate the Randc5 dataset which results in a dataset with 1760 entries. These two datasets helped us in incorporating the random mistuning data without introducing the computational burden of it. Based on our literature survey and available data, models such as neural networks and other regression-based models were decided upon to be explored for this project.

Thus, the goal of this project is to use machine learning (ML) regression algorithms to address the combinatorial explosion of intentional mistuning patterns with increase in N and cut down the evaluation time by utilizing a small fraction of the total number of mistuning patterns (training data) and predict AF of the remaining patterns (testing data). To achieve this, we shall choose appropriate ML based regression models such as neural networks, gaussian process regression, etc., and tune the hyperparameters for each of these models to achieve the model and the best prediction possible.

**2. Optimization using different Machine Learning models**

**2.1 Linear regression (LR)**

Linear Regression is the supervised Machine Learning model in which the model finds the best fit linear line between the independent and dependent variable i.e., it finds the linear relationship between the dependent and independent variable.

We started out by trying linear regression first. Although we knew our dataset was highly nonlinear, it was done to understand the pattern and kind of results in order to see the effects of various datasets and when L1 (or L1 regularization, L1 norm or Lasso (in regression problems), to combat overfitting by shrinking the parameters towards 0) and L2 (or L2 regularization, L2 norm, or Ridge (in regression problems), to combat overfitting by forcing weights to be small, but not making them exactly 0) penalty schemes were used[15].

The L2 penalty improved the MSE by a small margin but the overall residual range and prediction accuracy for both local and global solution was very poor. Nonlinearity was later introduced to the data in the form of additional features by multiplying the blade responses, two (12C2) and three (12C3) at a time but no significant improvement was seen and thus it was decided to explore better regression models.

The results from these simulations for linear regression are tabulated below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | MSE | Residual % | % diff (pred) | % diff (global) |
| Linear regression | 0.014 | 2.37 | -15.34 | 24.74 |
| Linear regression + 12C2 | 0.0067 | 2.28 | -15.34 | 24.74 |
| Linear regression + 12C2 + 12C3 | 0.0067 | 2.15 | -15.32 | 24.69 |

Table 1: Linear Regression (LR) - Evaluation metrics

It is evident that although the value of MSE decreases with the addition of nonlinear features, the overall residuals remain the same and there is no significant decrease in the prediction accuracy for neither the local nor the global AF. Thus, we can conclude that linear regression is not the best machine learning algorithm for our case.

**2.2 Neural Network (NN) - Fitrnet**[16]

Neural networks are one of the many methods of machine learning for both regression and classification purposes that form the heart of deep learning algorithms. Both their name and structure are inspired by the human brain and mimic the biological method of communication which is employed by the neurons signal to one another. Node layers in neural networks consist of an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has a weight and threshold associated with it. If the output of any individual node exceeds the specified threshold value, that node is activated and begins sending data to the next layer of the network. Otherwise, no data is passed to the next network layer. To learn and improve their accuracy over time, neural networks rely on training data. However, once these learning algorithms have been fine-tuned for accuracy, they become powerful tools in artificial intelligence, allowing us to predict data at a high rate.

Fitrnet is a tool used to train a fully connected feedforward neural network for regression. The neural network's initial fully connected layer is connected to the network input (predictor data), and each succeeding layer is connected to the preceding layer. Each fully connected (FC) layer multiplies the input by a weight matrix, followed by the addition of a bias vector. Each fully connected layer, except for the last, is followed by an activation function. The network's output, specifically predicted response values, is produced by the final fully connected layer.

Relu

Output

FC

FC

Input

Neural network was decided to be used next for our modelling. We decided to use MATLABs built in toolbox for machine learning. Fitrnet was the built in function for neural network which took in inputs which included the dataset as a table, output variable and hyperparameters options. The optimizable hyperparameters and their description are as follows.

|  |  |
| --- | --- |
| Parameter | Description |
| Hidden layers | The MATLAB default is up to 3 but more can be used up to 5 in total |
| Neurons | The default max per layer is 300 but that can be increased with no limit |
| Activation function | The four kinds of activation functions are: |
| Relu – (Rectified linear unit) performs a threshold operation on each element of the input, where any value less than zero is set to zero |
| Tanh – (Hyperbolic tangent) applies the tanh function to each input element |
| Sigmoid — Performs the following operation on each input element:  f(x)=1/1+e−x |
| None - Returns each input element without performing any transformation |
| Standardization | The software centers and scales each numeric predictor variable by the corresponding column mean and standard deviation |
| Layer weights | There are two methods to initialize weights, which are: |
| Glorot - Initializes the weights for each layer by independently sampling from a uniform distribution with zero mean and variable 2/(I+O),  where “I” is the input size and O is the output size for the layer. |
| He - Initializes the weights for each layer by sampling from a normal distribution with zero mean and variance 2/I,  where “I” is the input size for the layer. |
| Lambda | It is the regularization term strength which is a nonnegative scalar. MATLAB determines the objective function for minimization from the MSE loss L2 penalty term. |

Table 2: Hyperparameters options for Fitrnet

At first, Fitrnet was used in a 40/40/20 split of training, validation, and test data along with hyperparameter tuning set to auto in to tune and optimize all tunable hyperparameters to understand the effects of these parameters on the results and narrow down our search parameters for the optimum model. Auto tuning did not provide good prediction, but it helped us pick out parameters which had large effects on the results, such as, number of hidden layers, activation function, standardization and range and arrangement of neurons in the hidden layers. K-Fold cross validation was later introduced to reduce the amount of data used for training and validation and 5-fold cross validation was concluded to be an ideal choice.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | MSE | Max residual | Actual AF of max residual | AF of smallest residual |
| fitrnet\_int\_2080\_param\_kfold | 0.0199 | 0.4469 | 1.6557 | 1.2086 |
| fitrnet\_int\_2080\_param\_kfold10 | 0.0189 | 0.5062 | 1.4643 | 1.3572 |
| fitrnet\_int\_3070\_param\_kfold | 0.0191 | -0.9496 | 0.9534 | 1.1096 |
| fitrnet\_randc1\_2080\_param\_kfold | 0.0113 | 0.4168 | 1.6164 | 1.2001 |
| fitrnet\_randc5\_2080\_param\_kfold | 0.0107 | 0.4183 | 1.6364 | 1.2275 |
| fitrnet\_randc5\_2080\_param\_kfold10 | 0.0107 | 0.4161 | 1.6364 | 1.2728 |

Table 3: Selection of train/test split and K-Fold value

From this study, we can conclude that a 20-80 split for training and testing data along with 5-fold cross validation provides optimum results. Although increasing the training data does improve the prediction by a small amount, the computational time increases much more for it to be beneficial. This setup will now form the base on which the parameters will be tuned for all models.

We then performed the auto hyperparameter tuning to get an idea about which hyperparameters to tune and the range/kind of values to be provided as its input. It was evident after performing said tuning that a single hidden layer with a large value of neurons, two hidden layers with a small-large arrangement and three hidden layers with a large-small-medium number of neurons gave good predictions. We can also conclude that Relu, Tanh and Sigmoid were the best activation functions for our case and standardizing the data generally gave good predictions. The following table shows us the result of automatic hyperparameter tuning for Fitrnet and the various architecture of the neural network as decided by the software. This was used to draw the conclusions made above. The values in red were deemed the best by the model and its architecture was given the most preference for further tuning.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Optimization | Number of layers | Min neuron | Max neuron | Distribution kind | Activation function | Standardize |
| Case 1 | 2 | 28 | 305 | 28|305 | none | TRUE |
|  | 2 | 59 | 496 | 59|496 | none | TRUE |
|  | 1 | 2497 | 2497 | 2497 | sigmoid | FALSE |
|  | 2 | 18 | 132 | 18|132 | relu | FALSE |
|  | 3 | 19 | 3882 | 3882|19|1332 | relu | TRUE |
| Case 2 | 2 | 28 | 305 | 28|305 | none | TRUE |
|  | 2 | 40 | 322 | 40|322 | none | TRUE |
|  | 2 | 22 | 44 | 22|44 | none | TRUE |
|  | 2 | 12 | 3234 | 12|3234 | none | TRUE |
|  | 1 | 2479 | 2479 | 2479 | sigmoid | FALSE |
|  | 2 | 56 | 99 | 56|99 | tanh | FALSE |
| Case 3 | 2 | 35 | 142 | 35|142 | tanh | TRUE |
|  | 4 | 30 | 1262 | 1262|30|109|199 | tanh | TRUE |
|  | 2 | 19 | 2949 | 19|2949 | sigmoid | TRUE |
|  | 4 | 34 | 1426 | 1426|34|60|216 | tanh | TRUE |
|  | 2 | 70 | 141 | 70|141 | tanh | FALSE |
|  | 3 | 72 | 2991 | 2991|612|72 | relu | TRUE |
|  | 2 | 1460 | 1756 | 1460|1756 | sigmoid | TRUE |

Table 4: Best models from auto tuning

Now, a fractional factorial design was formulated to experiment with all possible combination of these values to generate our finalized design. The best results of the factorial design are shown below. We can thus conclude that the Int dataset provides the best results in terms of MSE and prediction of AF with the small value of residual. This indicates that with further tuning, it can predict a local AF with good accuracy. Thus, it was chosen to be the dataset to train our neural network models on. The bests model of each dataset was then used to predict the AF values for the other two datasets, i.e., Randc1 and Randc5, and it was noticed that the model trained on the Int dataset predicts the AF for the other two well and has a relatively lower value of MSE and residual when the models trained on Randc1 and Ranc5 are used to predict the remaining datasets. This again indicates that models trained on Int perform better.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Dataset | Case | MSE | Max residual | Actual AF of max residual | AF of smallest residual | Smallest AF predicted with smallest residual |
| Int | 30 | 0.0152 | 0.4672 | 1.4643 | 1.2552 | 1.2547 |
| rc1 | 0.1678 | -1.3241 | 0.8747 | 1.2057 | 1.208 |
| rc5 | 0.1602 | -1.4488 | 1.1283 | 1.1839 | 1.1831 |
| Randc1 | 1 | 0.0187 | -0.5826 | 1.142 | 1.1517 | 1.1509 |
| int | 0.0251 | -0.5379 | 0.9583 | 1.2893 | 1.2894 |
| rc5 | 0.0185 | -0.712 | 1.1681 | 1.2612 | 1.2612 |
| Randc5 | 10 | 0.0102 | 0.3984 | 1.6364 | 1.1637 | 1.1639 |
| int | 0.019 | 0.4148 | 1.6696 | 1.246 | 1.2472 |
| rc1 | 0.0104 | 0.3951 | 1.6164 | 1.233 | 1.2336 |

Table 5: Results of fractional factorial design

Upon training our model on Int, we predicted the best local AF and its percentage difference from the global minimum AF value for each dataset to generate the following values. This metric is then used henceforth to evaluate the performance of each regression model.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | MSE | Avg residual % | % diff (pred) | % diff (global) |
| Neural network (Int) | 0.0152 | -0.37 | 1.21 | 5.71 |
| Neural network (Randc1) | 0.1678 | -25.00 | -6.74 | 11.18 |
| Neural network (Randc5) | 0.1602 | -123.00 | -20.17 | -4.83 |

Table 6: Neural Network (NN) - Evaluation metrics

We can see that neural network model, which was trained on the Int dataset predicts a local minimum AF with an accuracy of 1.21% which is 5.71% off from the global minimum value. When we use the same model to predict the AF values of Randc1 and Randc5 datasets, they predict the global optimum within 11.18% and 4.83% of difference to the global value. This speaks well to the model’s scalability when using different datasets. One important observation that was noted was the increase in the residual percentage which indicates that a lot of values are not being predicted accurately but since the global minimum is predicted reasonably, this result was accepted but this is a problem which needs to be addressed in future work. The architecture of the best model (the first one int the table above) is shown in the table below.

|  |  |
| --- | --- |
| Parameter | Value |
| Hidden layers | [99,1250,3882] |
| Activation function | Relu |
| Standardization | True |
| Kfold | 5 |
| cvloss | 0.014 |
| L2 penalty (lambda) | 0.0035 |

Table 7: Architecture of best NN model

Three hidden layers were used in this model with their corresponding number of neurons being ordered from the smallest to largest as this arrangement was found to be the best from our fractional factorial design. A layer activation function of Relu was added between each hidden layer. The data was standardized, and a 5-fold cross validation was performed to find the lambda of the model with the smallest cross validation loss which are mentioned in the table above. This setup was then tuned on 20% of the total Int dataset to predict the AF for all three datasets.

It was then decided to explore more regression-based models and identify their important tuning hyperparameters. For this purpose, Fitrauto, another function of MATLABs machine learning toolbox was used as detailed below.

**2.3 Automatic Regression tuning - Fitrauto**[17]

Its main purpose is to try out different regression models available on MATLAB to help in deciding the regression models to pursue further for detailed evaluation. It attempts a variety of regression model types with different hyperparameter values given predictor and response data. The function selects models and hyperparameter values using Bayesian optimization and computes the following for each model: log(1 + valLoss), where valLoss is the cross-validation mean squared error (MSE). Fitrauto returns the model, trained on the full data set, that is expected to best predict the answers for new data after the optimization is complete. For the returned models, “predict” and “loss” functions can be used to predict the test data and compute its MSE, accordingly. When this was used on the entire rand dataset, we used an asynchronous successive halving algorithm (ASHA) instead of Bayesian optimization as ASHA optimization often finds good solutions faster than Bayesian optimization for data sets with over 100,000 observations.

The optimizable hyperparameters and their description are as follows.

|  |  |
| --- | --- |
| Parameter | Description |
| Learners | The regression models available in this function are ensemble, gp, kernel, linear, net, svm, tree, net which are described and explored in this report. |
| Standardization | The software centers and scales each numeric predictor variable by the corresponding column mean and standard deviation |
| Optimizer | There are two methods to optimize are: |
| “bayesopt” – Uses Bayesian optimization. |
| “asha” – Asynchronous successive halving algorithm (ASHA)   * It randomly chooses several models with different hyperparameter values and trains them on a small subset of the training data. If the performance of a particular model is promising, the model is promoted and trained on a larger amount of the training data. This process repeats, and successful models are trained on progressively larger amounts of data. * It is preferred to be used when you have a large dataset (over 100,000) for faster optimization compared to Bayesian optimization. |

Table 8: Hyperparameters options for Fitrauto

Using this function, Neural Networks (NN), Gaussian Process Regression (GP), Binary decision Tree (BT), Support Vector Machine for regression (SVM), Ensemble Models (EM) and Gaussian Kernel (GK) were explored. NN and GP were chosen as the best models based on MSE and prediction accuracy for both local and global optimum values. Detailed tuning was performed for both these models as described in the report. Best parameters were also found out and analyzed further for each other model to observe its MSE and prediction accuracies.

**2.4 Gaussian Process Regression (GP) - Fitrgp**[18]

Gaussian process regression (GP) is a nonparametric, Bayesian regression approach that is making waves in the field of machine learning which is based on kernels. It has several advantages, including the ability to work well on small datasets and provide uncertainty measurements on predictions. Unlike many popular supervised machine learning algorithms, the Bayesian approach infers a probability distribution over all possible values.

The probability distribution over all admissible functions that fit the data is computed by GP. However, we specify a prior (on the function space), calculate the posterior using the training data, and compute the predictive posterior distribution on our points of interest, just like in Bayesian optimization.

The most promising optimizable hyperparameters and their description are as follows.

|  |  |
| --- | --- |
| Parameter | Description |
| Fit method | The methods to estimate parameters of the gp model are: |
| 'none' - No estimation |
| 'exact' – Uses exact Gaussian process regression |
| 'sd' – To be used for more than 2000 observations in dataset |
| 'sr' - Subset of regressors approximation |
| 'fic' - Fully independent conditional approximation |
| Standardization | The software centers and scales each numeric predictor variable by the corresponding column mean and standard deviation |
| Sigma | It is the initial value for the noise standard deviation of the Gaussian process model |
| Computation method | Specifies the method for computing the log likelihood and gradient for parameter estimation. |
| 'qr' — For better accuracy |
| 'v' - For faster computation of log likelihood gradients. |
| Kernel (Covariance) function | Specifies the covariance function for kernel. Their formula is linked in reference at the end[19] |
| 'exponential' |
| 'squaredexponential' |
| 'matern32' |
| 'matern52' |
| 'squaredexponential' |

Table 9: Hyperparameters options for Fitrgp

It was the second-best model based on the simulation run using Fitrauto. Using the same process as before, the best models and their prediction are detailed below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dataset | MSE | Residual % | % diff (pred) | % diff (global) |
| Int | 0.0136 | -0.20 | -3.15 | 13.07 |
| Randc1 | 0.0106 | -0.31 | 0.2 | 27.91 |
| Randc5 | 0.0106 | -0.39 | 0.3 | 19.41 |

Table 10: Gaussian Process Regression (GP) - Evaluation metrics

In this case, although the models trained on Randc1 and Randc5 predict the local minimum AF within 0.2% and 0.3% of accuracy, their prediction of the global minimum AF is off by a large factor. Thus, it can be concluded that the model trained on Int dataset performs better with a local minimum AF prediction accuracy of 3.15% and the global AF within ~13%. The hyperparameters of its best model are listed in the table below.

|  |  |
| --- | --- |
| Parameter | Value |
| Sigma | 0.3785 |
| Kernel Function | exponential |
| Standardization | True |
| Kfold | 5 |
| Fit method | none |
| optimizer | fminunc |

Table 11: Architecture of best GP model

It was noted that the values of sigma played a very small role in its optimization and there were limitations on its range of value set by MATLAB due to which it was not properly explored. We shall now look at the other regression models which were tested and evaluate its performance based on the same metrics.

**2.5 Binary Tree (BT) - Fitrtree**[20]

All regression techniques include a single output (response) variable and one or more input (predictor) variables. The output is a numerical variable. A general regression tree has a mixture of continuous and categorical variables as input variables. A decision tree is formed when each decision node in the tree contains a test on the value of some input variable. The predicted output variable values are located at the tree's terminal nodes.

A regression tree is generated by means of binary recursive partitioning, which involves an iterative process that divides the data into partitions or branches and then breaks down each partition into smaller groups as the method advances up each branch. All values in the training set are initially grouped into the same partition. The algorithm then begins dividing the data into the first two partitions aka, branches, employing every possible binary split on each field. The split that minimizes the sum of the squared deviations from the mean in the two separate partitions is chosen by the algorithm. This rule is then applied to each of the new branches. This process is repeated until each node reaches the minimum node size specified by the user and becomes a terminal node. (If the sum of a node's squared deviations from the mean is zero, the node is considered as a terminal node even if it has not reached the minimum size.)

Fitrtree generates a regression tree using the input variables (blade pattern) and output amplification factor (AF) in the dataset. The resulting tree is a binary tree, with each branching node split based on the values of the columns in the dataset. The most promising optimizable hyperparameters and their description are as follows.

|  |  |
| --- | --- |
| Parameter | Description |
| Standardization | The software centers and scales each numeric predictor variable by the corresponding column mean and standard deviation. |
| Max depth | Can provide a positive value for returning a tree that has fewer levels and requires fewer passes. The function doesn’t set any value for this by default. |
| Min Parent Size | Minimum number of branch node observations. |
| Num Bins | Number of bins for numeric predictors. This value is empty by default. |
| If specified, it bins every numeric predictor into at most ‘numBins’ equiprobable bins, and then grows trees on the bin indices instead of the original data. |
| Max Num splits | Specifies maximum number of decision splits or branch nodes. Set to 1 by default. |
| Min Leaf Size | Specifies minimum number of leaf node observations. |

Table 12: Hyperparameters options for Fitrtree

The best models which it generates are listed below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dataset | MSE | Residual % | % diff (pred) | % diff (global) |
| Int | 0.0236 | 0.68 | 3.8 | 27 |
| Randc1 | 0.0116 | 1.08 | 12.6 | 26.5 |
| Randc5 | 0.0111 | 1.02 | 11.5 | 29.79 |

Table 13: Binary Tree (BT) - Evaluation metrics

The Int dataset provides the best result locally but its off from the global minimum AF by 27%. The other datasets do not provide good results and it was noticed that most of the predictions were the same value for different patterns.

**2.6 Support Vector Machine (SVM) for regression - Fitrsvm**[21]

SVM analysis is a widely used machine learning tool for classification and regression. Because it is based on kernel functions, SVM regression is considered a nonparametric technique. The straight line required to fit the data is referred to as a hyperplane. The algorithm's goal is to find a hyperplane in an n-dimensional space that clearly classifies the data points. Support Vectors are the data points closest to the hyperplane on either side of the hyperplane. These affect the position and orientation of the hyperplane and thus aid in the construction of the SVM model.

Fitrsvm trains or cross-validates a support vector machine (SVM) regression model on a low- through moderate-dimensional predictor data set. It supports mapping the predictor data using kernel functions, and supports Sequential Minimal Optimization (SMO), Iterative Single Data Algorithm (ISDA), or L1 soft-margin minimization via quadratic programming (L1QP) for objective-function minimization. The most promising optimizable hyperparameters and their description are as follows.

|  |  |
| --- | --- |
| Parameter | Description |
| Standardization | The software centers and scales each numeric predictor variable by the corresponding column mean and standard deviation. |
| Optimizer | The options for optimization are: |
| 'bayesopt' — Use Bayesian optimization. |
| 'gridsearch' — searches in a random order, using uniform sampling without replacement from the grid. |
| Box Constraint | The function searches among positive values, by default log-scaled in the range [1e-3,1e3]. |
| Kernel Scale | The function searches among positive values, by default log-scaled in the range [1e-3,1e3]. |
| Kernel Function | The function searches among 'gaussian', 'linear', and 'polynomial'. |
| Polynomial Order | The function utilized the value of integers in the range [2,4]. |

Table 14: Hyperparameters options for Fitrsvm

The best models generated after optimization are listed below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dataset | MSE | Residual % | % diff (pred) | % diff (global) |
| Int | 0.0144 | -1.18 | 0.5 | 19.58 |
| Randc1 | 0.0115 | -3.21 | 6.8 | 29.47 |
| Randc5 | 0.0106 | -2.14 | 1.0 | 24.28 |

Table 15: Support Vector Machine (SVM) - Evaluation metrics

Fitrsvm predicts the local minimum AF very accurately but its off from the global minimum by more than 20% and hence the results are not acceptable. It also has the highest consistent value of residual % amongst all the models for each dataset. Int dataset again performed the best for this model.

**2.7 Ensemble Methods (EM) - Fitrensemble**[22]

Ensemble methods aim to improve generalizability / robustness over a single estimator by combining the predictions of several base estimators built with a given learning algorithm.

There are two kinds of ensemble methods:

* The driving principle behind averaging methods is to build several estimators independently and then average their predictions. Because its variance is reduced, the combined estimator is usually better than any of the single base estimators. Its example are the bagging methods.
* On the contrary, in boosting methods, base estimators are built sequentially, and the bias of the combined estimator is reduced. The objective is to combine several weak models into a powerful ensemble. Their examples include AdaBoost and Gradient Tree Boosting**.**

Fitrensemble returns the trained regression ensemble model object that contains the results of boosting 100 regression trees using LSBoost and the predictor and response data in the dataset. The most promising optimizable hyperparameters and their description are as follows.

|  |  |
| --- | --- |
| Parameter | Description |
| Standardization | The software centers and scales each numeric predictor variable by the corresponding column mean and standard deviation. |
| Method | The options for ensemble aggregation methods are: |
| 'LSBoost' – (Least-squares boosting) |
| ‘Bag’ — (Bootstrap aggregation) uses bagging with random predictor selections at each split (random forest) by default. |
| Num Bins | Number of bins for numeric predictors. This value is empty by default. |
| Num Learning Cycles | To specify the number of ensemble learning cycles. The software trains one weak learner for every template object in “Learners” which is tree by default. |

Table 16: Hyperparameters options for Fitrensemble

The best models generated are listed below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dataset | MSE | Residual % | % diff (pred) | % diff(global) |
| Int | 0.0158 | 0.45 | -4.3 | 14.82 |
| Randc1 | 0.0115 | -1.22 | -0.5 | 23.12 |
| Randc5 | 0.0105 | 2.31 | 0.6 | 25.54 |

Table 17: Ensemble Methods (EM) - Evaluation metrics

Just like Fitrsvm, it predicts the local minimum AF well but severely overpredicts the global minimum AF value. Int dataset does not have the best local prediction, but the global prediction is relatively better compared to the other datasets. It also has one of the lowest residuals % amongst the different regression models.

**2.8 Gaussian Kernel (GK) - Fitrkernel**[23]

Gaussian Kernel Regression (GK) is an interesting regression technique that does not require iterative learning. It is also a nonlinear regression technique. Gaussian kernel function is used to calculate kernels for the data points. Fitrkernel is a nonlinear regression algorithm that trains or cross-validates a Gaussian kernel regression model. Fitrkernel is more useful for big data applications with large training sets, but it can also be used with smaller data sets that fit in memory. Fitrkernel converts data from a low-dimensional space to a high-dimensional space and then fits a linear model in the high-dimensional space by minimizing the regularized objective function. To obtain the linear model in high-dimensional space, apply the Gaussian kernel to the model in low-dimensional space. Regularized support vector machine (SVM) and least-squares regression models are among the linear regression models available. The most promising optimizable hyperparameters and their description are as follows.

|  |  |
| --- | --- |
| Parameter | Description |
| Standardization | The software centers and scales each numeric predictor variable by the corresponding column mean and standard deviation. |
| Epsilon | The function searches among positive values, by default log-scaled in the range [1e-3,1e2]\*iqr(Y)/1.349. |
| Kernel Scale | The function searches among positive values, by default log-scaled in the range [1e-3,1e3]. |
| Lambda | The function searches among positive values, by default log-scaled in the range [1e-3,1e3]/n, where n is the number of observations. |
| Learners | Learner — The function searches among 'svm' and 'leastsquares' to use as specified. |

Table 18: Hyperparameters options for Fitrkernel

The best models which it generates are tabulated below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dataset | MSE | Residual % | % diff (pred) | % diff(global) |
| Int | 0.0168 | -0.67 | -2.4 | 10.31 |
| Randc1 | 0.0114 | -2.37 | -11.0 | 29.96 |
| Randc5 | 0.011 | 2.17 | 10.7 | 31.39 |

Table 19: Gaussian Kernel (GK) - Evaluation metrics

The Int dataset has the lowest residual % predicts both the local and global minimum AF with the most accuracy compared to the other models. Since its global prediction is off by 10.31%, it is not chosen for our case or explored further as neural network performs much better when compared to it.

**3. Observations**

The following table compiles the performance for all the best models trained on the Int dataset and its performance.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | MSE | Avg residual % | % diff (pred) | % diff (global) |
| Linear Regression (LR) | 0.014 | 2.37 | -15.34 | 24.74 |
| Neural Network (NN) | 0.0152 | -0.37 | 1.21 | 5.71 |
| Gaussian Process (GP) | 0.0136 | -0.20 | -3.15 | 13.07 |
| Binary Tree (BT) | 0.0236 | 0.68 | 3.76 | 27.00 |
| Support Vector Machine (SVM) | 0.0144 | -1.18 | 0.51 | 19.58 |
| Ensemble Methods (EM) | 0.0158 | 0.45 | -4.33 | 14.82 |
| Gaussian Kernel (GK) | 0.0168 | -0.67 | -2.42 | 10.31 |

Table 20: Comparison of ML models

We notice that the neural network model has the lowest residual % and prediction inaccuracy for the global minimum AF. Although MSE is a good evaluation metric for regression models, it did not prove very useful to compare the best models. Neural network is the only regression model that could predict the global minimum AF within ~5% as compared to the rest which are very off.

To summarize our work, in this project, several ML models as detailed in this report were explored to model the AF for the set of intentional mistuning patterns. LR with L2 penalty was explored first but did not provide good prediction which was anticipated. Combinations of input parameters were added to model nonlinearity without much success. Exploration of various nonlinear ML models was then conducted with automatic hyperparameter optimization to identify the best models. NN and GP models were chosen as the best and a set of hyperparameters and a range of values for them was identified. A fractional factorial study was later carried out to find the best set of hyperparameters for both NN and GP models. Random mistuning was later added to try and enrich the training data but did not improve prediction accuracy. The entirety of the random mistuning data was not used due to hardware restrictions as the computational time was exponentially increased with its addition. It was then decided to make two datasets from the random mistuning data to incorporate and check its effect. For this purpose, Randc1 dataset was created which was generated by taking 1 randomly chosen mistuned pattern out of every 1000 for each intentional mistuning pattern. The Randc5 dataset was generated similarly by choosing 5 random mistuned patterns out of every 1000 for each intentional mistuning pattern. Although these additional datasets do not provide any improvements, but instead serve as new data to verify and test the prediction of our best models on. The performance of our models on different datasets is tabulated below.

|  |  |  |  |
| --- | --- | --- | --- |
| Model \ Dataset | Int | RandC1 | RandC5 |
| NN | 1.2 | -7.8 | 0.6 |
| GP | -3.2 | 0.2 | 0.3 |
| BT | 3.8 | 12.6 | 11.5 |
| SVM | 0.5 | 6.8 | 1.0 |
| EM | -4.3 | -0.5 | 0.6 |
| GK | -2.4 | -11.0 | 10.7 |

Table 21: Local AF prediction % difference of models on different datasets

This table shows the % deviation of local predicted AF from its actual value and the table bellow shows the % deviation of the same prediction from the global minimum AF for that dataset.

|  |  |  |  |
| --- | --- | --- | --- |
| Model \ Dataset | Int | RandC1 | RandC5 |
| NN | 5.71 | 10.66 | 14.45 |
| GP | 13.07 | 27.91 | 19.41 |
| BT | 27.00 | 26.50 | 29.79 |
| SVM | 19.58 | 29.47 | 24.28 |
| EM | 14.82 | 23.12 | 25.54 |
| GK | 10.31 | 29.96 | 31.39 |

Table 22: Global AF prediction % difference of models on different datasets

It can be observed that all the models perform better when trained on the Int dataset. The local predictions are generally very good for all the models expect BT and GK, but the global predictions are poor for almost all of them. Even the NN models trained of Randc1 and Randc5 do not predict the global minimum within 10% of its value.

**4. Conclusions**

The NN model identifies a few local minimum values of AF which are close to the global optimum. The best NN model predicts a local optimum within 5% of the global optimum using only a fraction of the data (~20%). This ensures a large savings in computational time as exhaustive calculations are very computationally intensive.

As detailed in the discussions before, it was observed that ordering the neurons in the hidden layers from small to large with three hidden layers had the best results. The table provides the value of the predicted and actual AF and its deviation percentage as error with the global minimum.

|  |  |  |  |
| --- | --- | --- | --- |
| Best Model | AF actual | AF pred | Error |
| Best pattern (Full data) | 0.92 | 1.15 | -20.4 |
| Best pattern (Pred data) | 0.96 | 0.97 | -1.2 |
| Best overall | 0.92 | 0.97 | -5.4 |

Table 23: Comparison of predictions of AF for NN model

The minimum value of AF in the dataset was 0.92 and the model predicted the value for this pattern as 1.15 which is 20.4% off. The best predicted AF by the model is 0.97 and the actual value of AF for this pattern was 0.96 which is off by 1.2%. Thus, the best overall when comparing the minimum value of both patterns, the minimum predicted AF is 5.4% off from the true minimum. There is scope for improvement but for the sake of this project, this value was deemed acceptable.

For future work, Graph Neural Network (GNN) and Genetic Algorithms (GA) can be explored. Based on literature review, GNN seems to be a good model to represent our data since the features in our dataset are not mutually exclusive and do that interconnective meaning as they represent the pattern on each blade. The dataset can be looked at for improvement in representation of patterns and their effect (in this case, AF was decided as the best output). Different sampling algorithms can be explored such as Latin Hypercube Sampling (LHS) to determine the testing and training data. Also, utilization of a larger random mistuning data can be attempted for training to check if it improves the result.

**5. References**

[1] Y. Han, R. Murthy, M. P. Mignolet, and J. Lentz, ‘Optimization of intentional mistuning patterns for the mitigation of the effectsof random mistuning’, *J Eng Gas Turbine Power*, vol. 136, no. 6, Jun. 2014, doi: 10.1115/1.4026141.

[2] V. Píštěk, P. Kučera, O. Fomin, A. Lovska, and A. Prokop, ‘Acoustic Identification of Turbocharger Impeller Mistuning—A New Tool for Low Emission Engine Development’, *Applied Sciences 2020, Vol. 10, Page 6394*, vol. 10, no. 18, p. 6394, Sep. 2020, doi: 10.3390/APP10186394.

[3] S.-T. Wei, G. Research, and A. C. Pierre, ‘Localization Phenomena in Mistimed Assemblies with Cyclic Symmetry Part I: Free Vibrations’, 1988. [Online]. Available: http://vibrationacoustics.asmedigitalcollection.asme.org/

[4] S.-T. Wei, G. Research, and A. C. Pierre, ‘Localization Phenomena in Mistimed Assemblies with Cyclic Symmetry Part II: Forced Vibrations’, 1988. [Online]. Available: http://vibrationacoustics.asmedigitalcollection.asme.org/

[5] B.-K. Choi, H.-J. Kim, H.-E. Jeong, D.-S. Gu, and W.-C. Kim, ‘Mistuning Intensity Effect to Optimization of Mistuning Pattern for Bladed Disk’, *Journal of Fluid Machinery*, vol. 10, no. 1, pp. 14–19, Feb. 2007, doi: 10.5293/kfma.2007.10.1.014.

[6] L. Pohle, S. Tatzko, L. P. Von Scheidt, and J. Wallaschek, ‘High order sensitivity analysis of a mistuned blisk including intentional mistuning’, *Journal of Theoretical and Applied Mechanics (Poland)*, vol. 55, no. 1, pp. 353–368, 2017, doi: 10.15632/jtam-pl.55.1.353.

[7] T. Zhao, H. Li, and H. Sun, ‘Parallel intelligent algorithm analysis of optimization arrangement on mistuned blades based on compute unified device architecture’, *Proc Inst Mech Eng G J Aerosp Eng*, vol. 233, no. 6, pp. 2207–2218, May 2019, doi: 10.1177/0954410018778778.

[8] I. Kiss, Z. Badics, S. Gyimothy, and J. Pavo, ‘High locality and increased intra-node parallelism for solving finite element models on GPUs by novel element-by-element implementation’, *2012 IEEE Conference on High Performance Extreme Computing*, 2012, doi: 10.1109/HPEC.2012.6408659.

[9] T. Zhao, H. Yuan, W. Yang, and H. Sun, ‘Genetic particle swarm parallel algorithm analysis of optimization arrangement on mistuned blades’, *Engineering Optimization*, vol. 49, no. 12, pp. 2095–2116, Dec. 2017, doi: 10.1080/0305215X.2017.1292422.

[10] H. Lemos, M. Prates, P. Avelar, and L. Lamb, ‘Graph Colouring Meets Deep Learning: Effective Graph Neural Network Models for Combinatorial Problems’, Mar. 2019, [Online]. Available: http://arxiv.org/abs/1903.04598

[11] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, ‘Neural Message Passing for Quantum Chemistry’, Apr. 2017, Accessed: Apr. 30, 2023. [Online]. Available: http://arxiv.org/abs/1704.01212

[12] K. Cho *et al.*, ‘Learning phrase representations using RNN encoder-decoder for statistical machine translation’, *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 1724–1734, 2014, doi: 10.3115/V1/D14-1179.

[13] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, ‘The graph neural network model’, *IEEE Trans Neural Netw*, vol. 20, no. 1, pp. 61–80, Jan. 2009, doi: 10.1109/TNN.2008.2005605.

[14] F. Geyer, ‘DeepComNet’, *Performance Evaluation*, vol. 130, pp. 1–16, Apr. 2019, doi: 10.1016/J.PEVA.2018.12.003.

[15] ‘About Linear Regression | IBM’. https://www.ibm.com/topics/linear-regression (accessed Apr. 30, 2023).

[16] ‘Train neural network regression model - MATLAB fitrnet’. https://www.mathworks.com/help/stats/fitrnet.html (accessed Apr. 30, 2023).

[17] ‘Automatically select regression model with optimized hyperparameters - MATLAB fitrauto’. https://www.mathworks.com/help/stats/fitrauto.html (accessed Apr. 30, 2023).

[18] ‘Fit a Gaussian process regression (GPR) model - MATLAB fitrgp’. https://www.mathworks.com/help/stats/fitrgp.html#d124e426265 (accessed Apr. 30, 2023).

[19] ‘Kernel (Covariance) Function Options - MATLAB & Simulink’. https://www.mathworks.com/help/stats/kernel-covariance-function-options.html (accessed Apr. 30, 2023).

[20] ‘Fit binary decision tree for regression - MATLAB fitrtree’. https://www.mathworks.com/help/stats/fitrtree.html (accessed Apr. 30, 2023).

[21] ‘Fit a support vector machine regression model - MATLAB fitrsvm’. https://www.mathworks.com/help/stats/fitrsvm.html (accessed Apr. 30, 2023).

[22] ‘Fit ensemble of learners for regression - MATLAB fitrensemble’. https://www.mathworks.com/help/stats/fitrensemble.html (accessed Apr. 30, 2023).

[23] ‘Fit Gaussian kernel regression model using random feature expansion - MATLAB fitrkernel’. https://www.mathworks.com/help/stats/fitrkernel.html (accessed Apr. 30, 2023).