A Comprehensive Approach to Fault Classification of Helicopter Engines with Adaboost Ensemble Model

Peeyush Pankaj[1], Sammit Jain[2], and Shyam Joshi[3]

[1,2]*MathWorks India, Trillium Building, Blocks I & J, Embassy Tech Village, Bangalore, India 560103*
ppankaj@mathworks.com
sammitj@mathworks.com

[3]*MathWorks USA, 5810 Tennyson Parkway Suite 425, Plano, TX, USA 75024*
shyamj@mathworks.com

ABSTRACT

This work is based on the PHM North America 2024 Conference Data Challenge's datasets of Helicopter turbine engine performance measurements. These datasets were large and moderately imbalanced. This submission produces compelling results using MATLAB for all the necessary visualizations, feature engineering, model exploration, explainability, and confidence margin estimation. All these tools will be generally applicable to data-driven AI/ML modeling and predictions.

The MathWorks team score on Testing Data was 0.9686 at the close of competition. This was further improved to 0.9867. The Validation Data submission scores were also improved. Our approach demonstrates the effectiveness of combining strategic data processing, feature engineering, and model optimization. High prediction metrics and explainability were demonstrated.

1. INTRODUCTION

The health monitoring of helicopter turbine engines is a critical component in ensuring operational safety and efficiency. With the increasing availability of large-scale operational datasets, there is a growing opportunity to leverage advanced data analytics and machine learning techniques to enhance predictive maintenance strategies. Such predictive maintenance strategies traditionally rely on prognostics data trends over time and comparing these against performance characteristic curves informed from domain expertise (Bechhhoefer & Hajimohammadali, 2023). However, this study addresses the PHM North America 2024 Conference Data Challenge, which involves predicting the health of helicopter turbine engines by estimating torque margins and classifying engine health status, where none of the time-dependent information in the dataset was available. The approach we used is described in Figure 1 below.

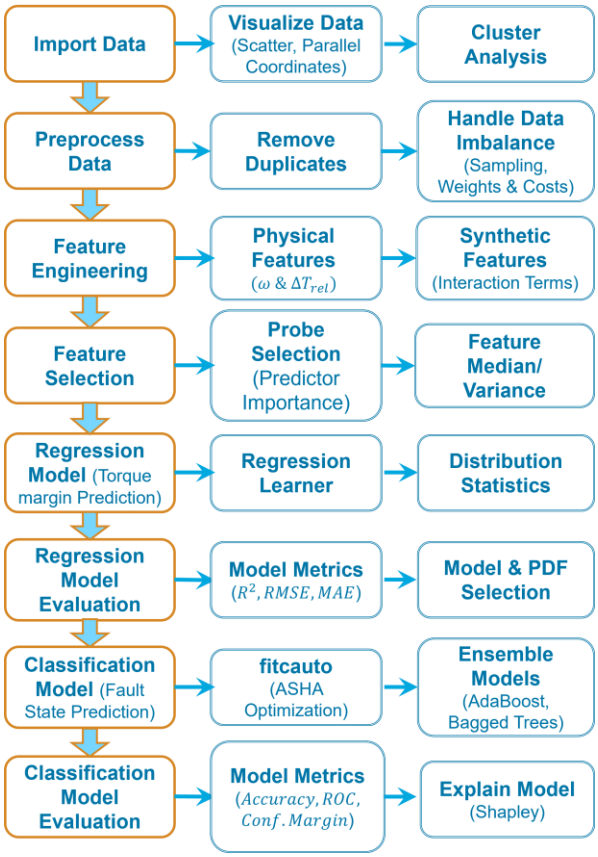


Figure 1. Block diagram representing workflow adopted for failure prediction modeling on the helicopter engine dataset.

This was structured around a systematic data-driven workflow that breaks down the analysis into three key stages: data access & exploration, data preparation, and AI modeling.

2. DESCRIPTION OF THE DATASET

The dataset provided for the 2024 PHM Society Data Challenge consists of 742,625 observations in the training set. Additionally, there are 21,436 observations present in both the test and validation sets each. The training set data comes from 4 different helicopter turbine engines, but the same make. Each engine is instrumented to capture the outside air temperature (oat), mean gas temperature (mgt), pressure altitude (pa), indicated airspeed (ias), net power (np), and compressor speed (ng). For these operational conditions, there is a design (target) torque. The real output torque is also measured. Engine health is assessed by comparing the output torque to the target torque. More specifically, the torque margin (calculated as per Eq. 1 below) is a key indicator of engine health:

$$\boldsymbol{Torque\ Margin}\ (\%) = \mathbf{100} \times \frac{(Torque_{measured} - Torque_{target})}{Torque_{target}} \quad (1)$$

The participating teams of the data challenge are tasked to submit 2 types of predictions, a. torque margin estimation, and b. fault labels (0 for healthy and 1 for faulty). The fault labels are only provided in the training set.

The observations have though been shuffled and asset ids have been removed. The test and validation sets represent data from 3 different helicopter engines, which are not part of the training set.

3. DATA ACCESS AND EXPLORATION

Initial exploration of the training dataset revealed that the training dataset is labeled with a roughly 60-40 split between healthy and faulty engine statuses, and largely distributed into two major operational clusters as represented in Figure 2 below.

This clustering is a common characteristic in fleet analytics, where variations in operational conditions or engine configurations can lead to distinct groupings. The presence of these clusters suggests two potential modeling strategies: the development of global models that utilize the entire dataset and the creation of local models tailored to each operational cluster, potentially offering more precise predictions by leveraging the unique characteristics within each group. This dual modeling strategy allows for flexibility in prediction, as new data received from the field can be evaluated using both the global model and the appropriate local model, depending on its proximity to the cluster centroids.

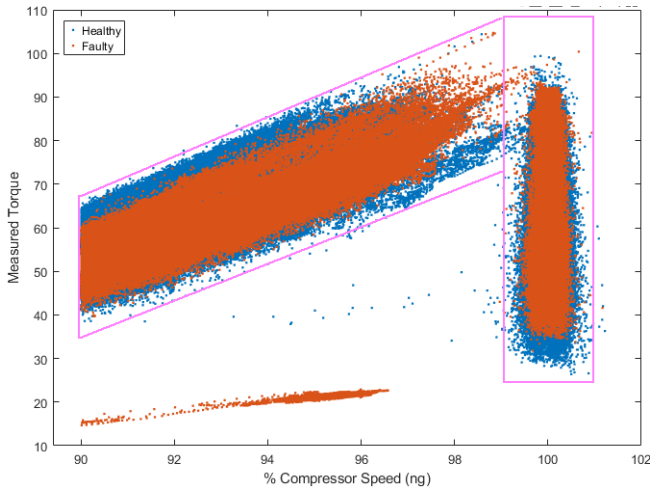


Figure 2. Scatter plot of compressor speed (ng) vs $Torque_{measured}$ reveals 2 major clusters of operation**.** There's also a decent number of observations outside of these 2 clusters, but those are largely faulty observations.

4. DATA PRE-PROCESSING AND FEATURE ENGINEERING

4.1. Removing Duplicates

In the initial stage of data pre-processing, we identified and addressed the presence of duplicate observations within the training dataset. Specifically, 59,600 rows were found to be exact duplicates. The elimination of these duplicates resulted in a shift in the distribution of labels, altering the healthy-faulty split from an approximate 60-40 ratio to a more imbalanced 67-33 ratio.

4.2. Data Balancing with Upsample Downweight

To address the class imbalance in the training dataset, we employed an "Upsample Downweight" strategy on the minority class by padding the dataset with additional instances of the faulty class. This process ensured that the dataset achieved an even 50-50 split between healthy and faulty observations, thereby mitigating the risk of model bias towards the majority class during training. However, simply upsampling the minority class could lead to overfitting, as the model might learn redundant patterns from the duplicated data. To counteract this, we adjusted the weights of the faulty class observations, reducing them in comparison to those of the healthy class. These weights were then incorporated as a parameter during the training of decision tree ensembles. There are more strategies for classification of imbalanced data (MathWorks Documentation: Handling Imbalanced Data, 2024), for example, using Misclassification Costs (Zhou & Liu, 2010) or RUSBoost (Seiffert, Khoshgoftaar, Hulse & Napolitano, 2008), which were explored later at the AI Modeling stage.

4.3. Feature Engineering

Feature engineering is a critical step in enhancing the predictive power of machine learning models by incorporating domain knowledge (FAA, 2019) into the dataset. In this study, we introduced two new features derived from domain understanding of the operational parameters highlighted in Eq. 2 & Eq. 3:

$$\Delta T_{relative} = (mgt - oat)/mgt \quad (2)$$

$$\omega = np/Torque_{measured} \quad (3)$$

As the engines age, one would expect the turbine to generate lesser work, and monitoring the relative temperature drop can be an indicator for the same. Whereas angular velocity computation will be useful compared to rotor speed 'ng' which is denoted in percentage terms.

Recognizing the prevalence of second-order terms in physics-driven equations, we extended this principle to our feature set by introducing quadratic terms for all seven original parameters and the two newly engineered features. This resulted in a focused set of 18 features that were particularly relevant for training regression models aimed at predicting design torque.

Feature modeling and machine learning/deep learning model training are inherently iterative processes. Through multiple iterations, we identified and incorporated numerous linear interaction terms among these 18 features. This iterative refinement expanded our feature set to a comprehensive 242 features. This extensive feature set was crucial for capturing complex relationships and interactions within the data, thereby enhancing the models' ability to make accurate predictions.

4.4. Dimensionality Reduction with Random Noise Probing

To effectively reduce the dimensionality of our extensive feature set, we employed a strategy involving the introduction of random noise and subsequent feature importance analysis (Stoppiglia, Dreyfus, Dubois & Oussar, 2003). We began by selecting a subset of 100,000 observations from the full training dataset to facilitate efficient computation. A baseline fault classification decision tree model was then trained on this sample dataset, achieving an initial accuracy of 98% before any feature reduction was applied.

Next, we introduced three random features to the dataset, each representing random noise generated from different statistical distributions: normal, binary, and uniform. The feature values for each of these distributions is shown in Figure 3. The decision tree model was retrained with these additional noise features, maintaining the same accuracy of 98%. This step was crucial for utilizing MATLAB's 'predictorImportance' function(MathWorks Documentation: Predictor Importance, 2024), which ranks features based on their contribution to the classification decision tree. It does so by summing changes in risk due to splits on every predictor and dividing the sum by the number of branch nodes.

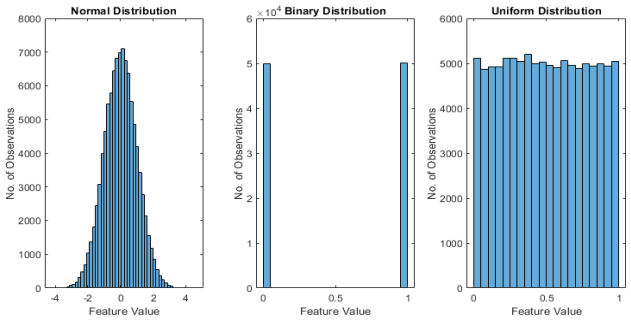


Figure 3. Introduction of 3 Random Features based on normal, binary & uniform distributions

The random noise features, by design, should exhibit minimal importance scores. Consequently, any features in the original set that showed lower importance scores than the mean importance of the three random noise features were considered insignificant and removed from the dataset. This process resulted in a reduced set of 58 features. Finally, the decision tree model was retrained using this reduced set of 58 features. The model's accuracy remained at 98%, confirming that the features removed had an insignificant impact on the model's performance.

4.5. Parallel Coordinates Plot

To further refine our feature set, we utilized a parallel coordinates plot, as shown in Figure 4. This technique allows for simultaneous comparison of multiple features, providing insights to retain features that exhibit a significant normalized variance in their coordinate values. Such features help in discriminating different classes, making them valuable for model training. Conversely, features that demonstrated minimal variance were deemed less informative and were consequently removed from the feature set. This further reduced the feature set from 58 to 35 features.

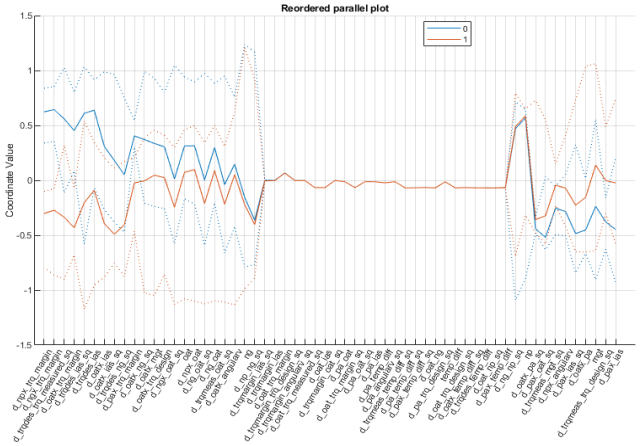


Figure 4. Parallel coordinates plot showing the normalized variance in each feature for failure labels, i.e. healthy (0) and faulty (1)

5. AI MODELING

Rather than employing a traditional holdout validation approach, which could potentially limit the evaluation of model performance to a single subset of the data, we utilized 5-fold cross-validation. This cross-validation technique involves partitioning the dataset into five distinct subsets or "folds." During the training process, the model is iteratively trained on four folds while being validated on the remaining fold. This process is repeated five times, with each fold serving as the validation set exactly once. The results from each iteration are then averaged to produce a comprehensive assessment of the model's performance.

5.1. Regression Model for Torque Margin Estimation

In our effort to accurately estimate the torque margin, we trained multiple regression models using the dataset with the refined set of 18 features, as outlined in the feature engineering section. These features were used to predict the design torque effectively. The torque margin was subsequently calculated using an arithmetic equation detailed in Eq. 1. Figure 5 Below depicts the summary of the linear regression models trained on one of the local clusters in the training dataset as highlighted in the Data Exploration section.

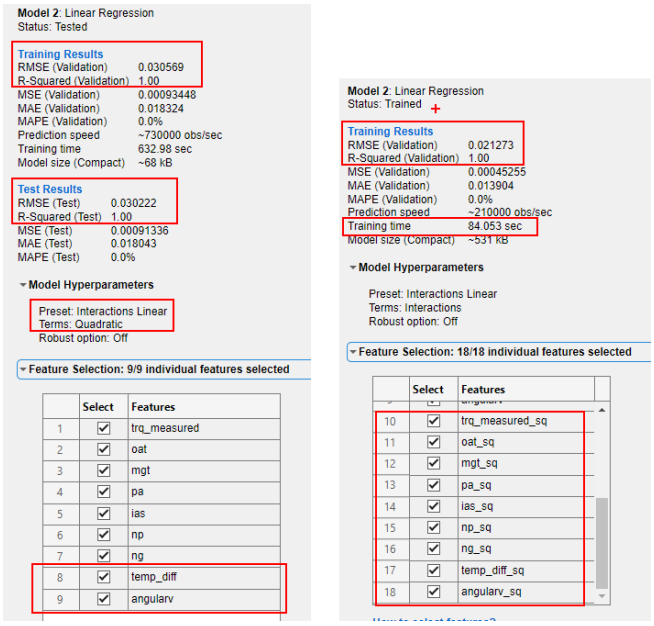


Figure 5. Including quadratic terms for each predictors (right) significantly reduced the RMSE score. MATLAB's regression learner app is used here for model(s) training in parallel over multiple cores

The linear regression model with interaction terms seems to be performing better than high fidelity feed forward neural networks, so we chose linear regression with interaction terms. We used this insight, and further trained a sequential linear regression model on the same data. Sequential LM choses the right terms to be included in the model based on their impact score with each iteration. Following is the resulting linear regression equation on one of the data clusters. Note that the R^2 of most of the trained models are 1.0, which is only part of measuring the accuracy of the regression models. To measure how closely these points fit to the ground truth, we should be looking to optimize the RMSE (Root Mean Squared Error) or MAE (Mean Absolute Error) scores. The coefficients and the terms of the linear regression model are tabulated in Table **1**, while Figure 6 describes the torque margin residuals for the training data. All the residuals lie within a consolidated range of +/-0.5% error, and 99% of the residuals lie within the range of +/-0.1% error.

Table 1. Estimated Coefficients for a Linear Regression Model from a sequential LM approach

Term	Estimate	SE	tStat	pValue
(Intercept)	-179.65	0.69	-259.03	0.00
trq_measured	0.02	0	34.42	0.00
oat	-1.1	0	-413.58	0.00
mgt	0.45	0	408.21	0.00
pa	-0.01	0	-223.26	0.00
ias	-0.01	0	-51.83	0.00
ng	0.98	0.02	44.67	0.00
oat_sq	-0.01	0	-33.12	0.00
ng_sq	-0.01	0	-38.09	0.00

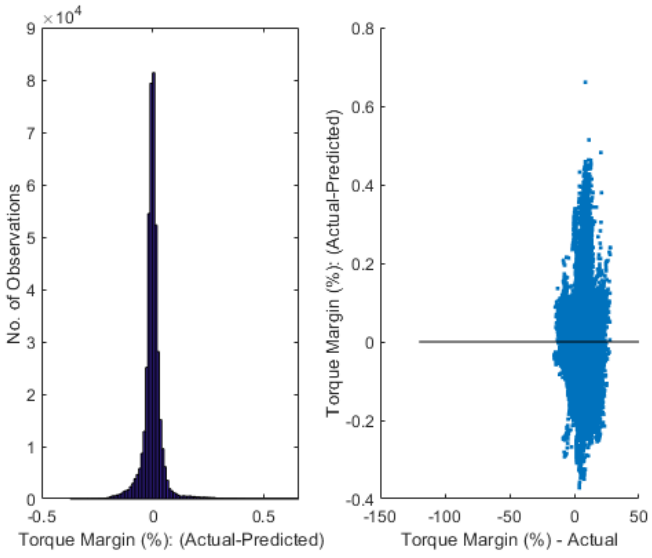


Figure 6 Histogram(left) and scatter plot (right) of Torque Margin Residuals on the training data show the distribution between lower bound of -0.4% and upper bound of +0.5%, with more than 90% of the observations being predicted with +/-0.1% of the ground truth.

5.2. Selection of Probability Density Function

In this year's data challenge, the goodness of fit is critical to getting higher score on the torque margin predictions. However, if we use any probability density function which has a bell-shaped curve (e.g. normal, cauchy or logistic distributions), one will lose some marginal points even when the torque margin predictions are few decimal precisions away from the ground truth. Thus, in order to get the perfect score on the torque margin regression predictions we chose "uniform" distribution function with flat-top after ensuring that the histogram for the torque margin errors is within a bandwidth of 1.0%. Figure 7 and Figure 8 demonstrate the drop in regression scores for different probability density functions if the ground truth is away from the predicted value, apart from the uniformly distributed probability density function.

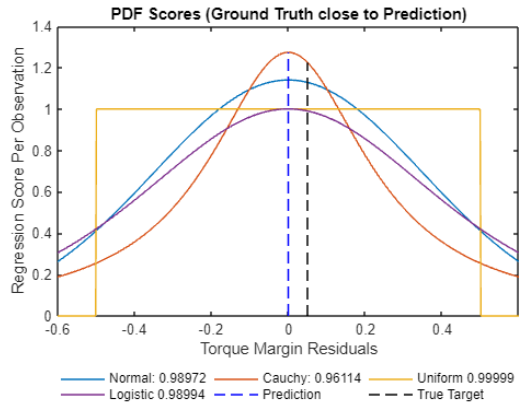


Figure 7. Comparison of different probability density functions when the true target is near the predicted outcome. In such scenario, there can be multiple pdf functions which will fetch high scores

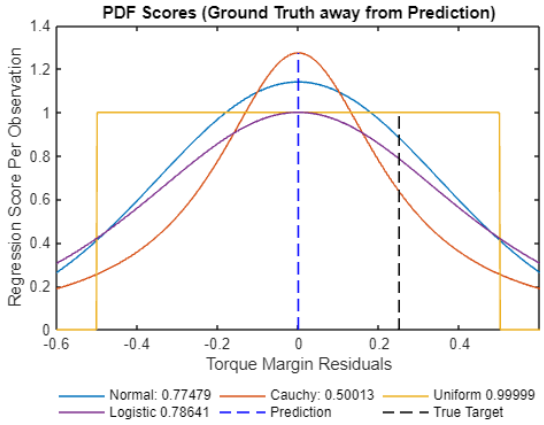


Figure 8. Comparison of different probability density functions when the true target is away from the predicted outcome. If the torque margin residual (prediction - actual) is less than 1%, then uniform distribution with a flat top will give the perfect score on regression predictions unlike any other pdf with a bell-curve.

The test data submission score was also validated as 0.5 by just submitting the regression model predictions (assigning 0 confidence to all classification predictions). This means a perfect score of 1.0 on the torque margin predictions.

5.3. Fault Class Predictions

5.3.1. Model selection with Automated Machine Learning and ASHA Optimization

Traditional exploration and hyperparameter optimization of Classification models involves exhaustive experimentation with different model families and parameter sets – which was found to be computationally intensive. Given the time-sensitive nature of the data challenge and the large number of observations in the training dataset, pursuing a hit-and-trial approach of model selection or using a Bayesian optimizer for automated model selection would consume a lot of time. Thus, an automated classification model training ("fitcauto" function in MATLAB (MathWorks documentation for Automated Classifier Selection, 2024) with ASHA (asynchronous successive halving algorithm) optimization was employed to determine the best suite of classification models for this dataset (Li, Liam, Jamieson, Rostamizadeh, Gonina, Hardt, Recht, & Talwalkar., 2020). The function randomly chooses several models with different hyperparameter values and trains them on a small subset of the training data. If the cross-validation classification error (Validation Loss) of a particular model is promising, the model is promoted and trained on a larger amount of the training data as shown in Figure 9. This process repeats, and successful models are trained on progressively larger amounts of data. The outcome of this process is to use decision tree ensembles as highlighted at the end of Figure 10

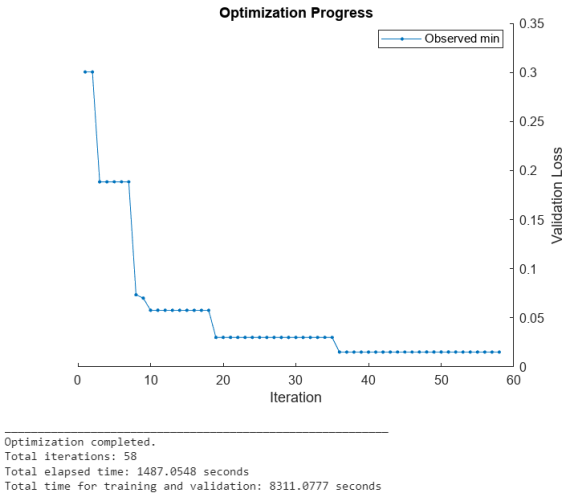


Figure 9. fitcauto optimizing for validation loss with different types of learners

Iter	Active workers	Eval result	Validation loss	Time for training & validation (sec)	Observed min validation loss	Training set size	Learner	Hyperparameter:	Value
1	24	Best	0.3004	29.179	0.3004	2321	net	Activations:	none
								Standardize:	false
								Lambda:	0.040751
								LayerSizes:	[10 10]
2	24	Accept	0.54316	32.618	0.3004	2321	net	Activations:	none
								Standardize:	false
								Lambda:	1.4094e-05
								LayerSizes:	[7 2 4]
3	24	Best	0.18854	48.1	0.18854	2321	nb	DistributionNames:	normal
								Width:	NaN
								Standardize:	-
12	24	Accept	0.1393	52.658	0.05771	2321	tree	MinLeafSize:	438
13	24	Accept	0.13402	35.484	0.05771	2321	tree	MinLeafSize:	620
14	24	Accept	0.40319	40.453	0.05771	2321	ensemble	MinLeafSize:	6490
15	24	Accept	0.24889	96.249	0.05771	2321	net	Activations:	sigmoid
								Standardize:	false
								Lambda:	0.00027284
								LayerSizes:	22
34	24	Accept	0.40319	62.541	0.030076	2321	ensemble	MinLeafSize:	32082
35	24	Accept	0.032307	264.88	0.030076	2321	svm	BoxConstraint:	2.0527
48	24	Accept	0.40319	9.227	0.015189	2321	tree	MinLeafSize:	2.1217e+05
49	24	Accept	0.029203	341.24	0.015189	9283	net	Activations:	sigmoid
58	24	Accept	0.10353	56.255	0.015189	2321	ensemble	MinLeafSize:	199

Figure 10. Iterations done with fitcauto function in MATLAB with ASHA optimization indicate use of ensemble models

5.3.2. Experimenting With Different Classifier Models and Approach for Down-Selecting the Best Classifier

For modeling a fault classifier, several models were trained. Firstly, we set aside 20% of the training data as holdout set and trained multiple models, e.g. decision trees, support vector machines, neural networks using classification learner app in MATLAB. Some of these models, e.g. wide neural networks returned with a 99.6% accuracy on the holdout validation set and was also submitted for the data challenge in initial rounds. Though we learnt that by holding out a portion of the training data, the model is missing out on some information and is unable to generalize well over the other engines in the test data. This made us to use the full training data for model training.

Having over 700,000 observations in the dataset is a challenge and requires a higher compute time if the models are trained on single core. For this reason, we loaned cloud compute and trained deep learning neural network models using GPUs. The wide neural network approach faded away as the confidence margins for the failure labels were inconsistent with the prediction errors on the test data, i.e. the neural network model was making highly confident errors.

Next up, we explored the automated classifier modeling with ASHA optimization (detailed in previous section) and understood that the ensemble decision trees will perform better on this dataset. So, we experimented between different types of decision trees with accelerating the model training with distributed computing on cloud resource.

5.3.3. Ensemble Decision Trees Modeling

After experimenting with the different hyper parameters of the bagged tree models, we trained ensemble bagged decision trees (Breiman, Friedman, Olshen & Stone, 1984) with minimum leaf size as 30 and 150 decision trees. This model was in fact used for predicting fault classes at the close of the competition. We got a combined score of 0.9686 on the test data with this model.

However, the bagged trees ensemble model was later revised to AdaBoost (adaptive boosting) decision trees ensemble, which improved the score on test data to 0.9867. Bagged tree model (Bootstrap Aggregation) randomly selects examples with replacements. Up to 38% examples are left out of bag (not used in training). Hence, cross-validation or holdout validation are not needed, and out-of-bag loss (oobloss) provides the estimate of generalization error. However, it trains on duplicate examples due to bagging and this may cause overfitting. Though using the AdaBoost approach increases the model training time significantly due to adaptive learning as compared to bagged trees where the trees formation can be distributed to multiple cores with parallel computing. We also tried binning the numerical predictors for faster compute time, but this approach loses some information. Training an AdaBoost model in MATLAB allows for resuming the model training, which allowed us to grow the number of trees in batches. The dynamically reducing misclassification rate on the training data is shown in Figure 12. Here we stopped the training early such that the model doesn't overfit on the training set, and the misclassification rate is also within allowable limits (less than 2%). Adaboost increases the weight of examples that were predicted wrongly in previous iterations. This increases the penalties for making those mistakes. Each tree in the ensemble minimizes loss (weighted cost of wrong predictions). All the examples from the training set are used in fitting the ensemble model.

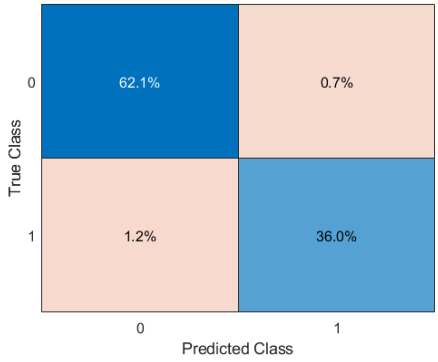


Figure 11 Confusion Matrix on the Training dataset for the AdaBoost model

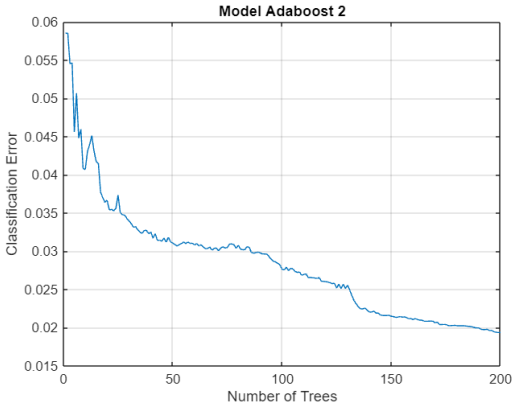


Figure 12. Misclassification rate plotted against the number of trees

The ROC and precision-recall performance curves on the training data are shown in Figure 13 and Figure 14 respectively. The Area Under the Curve (AUC) denotes high accuracy, and high values of both precision and recall for different thresholds. High precision is achieved by having few false positives in the returned results, and high recall is achieved by having few false negatives in the relevant results.

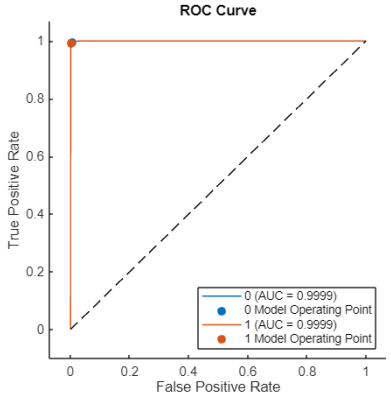


Figure 13. ROC (Receiver Operating Characteristic) curve for the training data

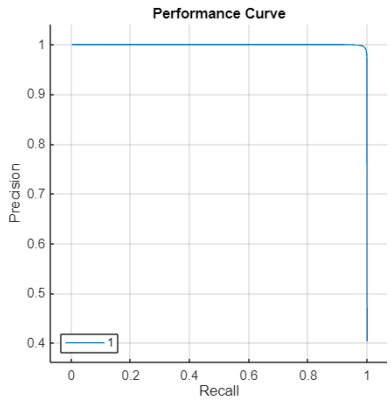


Figure 14. Recall-Precision curve for the training data

Though on the training set, the AdaBoost model's performance shows ~2% misclassification, we had another metric in the form of confidence margins on the test predictions as reported by different learners. The Figure 15 below shows the confidence margin histogram for all the predictions on test data being heavily skewed towards high nineties. This was not the case when other types of learners were overfitting on the training set.

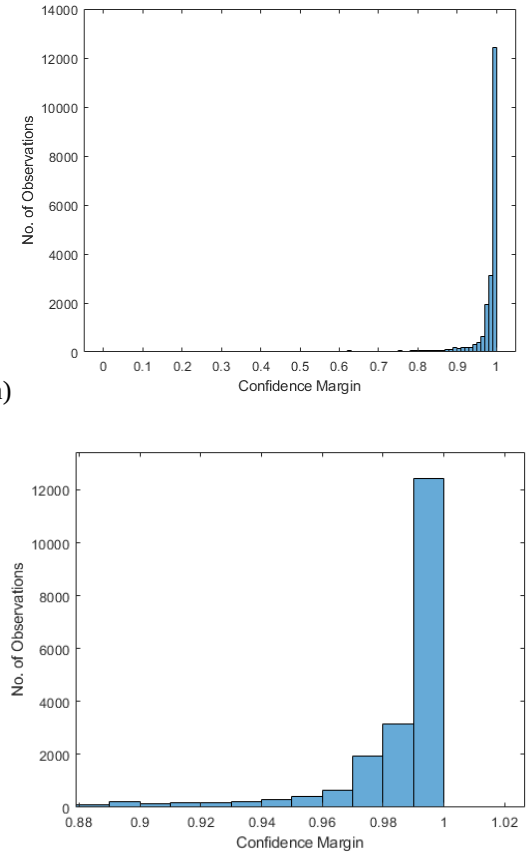


Figure 15 (a) Confidence Margins reported by the AdaBoost ensemble model with 200 trees on the test set predictions, (b) Magnified view of 0.9-1 Confidence Margin

5.3.4. Sanity Check for Test Data Predictions

Along with the confidence margin histogram on the test data, we had another sanity check in place based on data visualization to ensure that the models are not making highly confident false predictions. We visualized the data from training and test set on a grouped scatter plot as described in Figure 16 below.

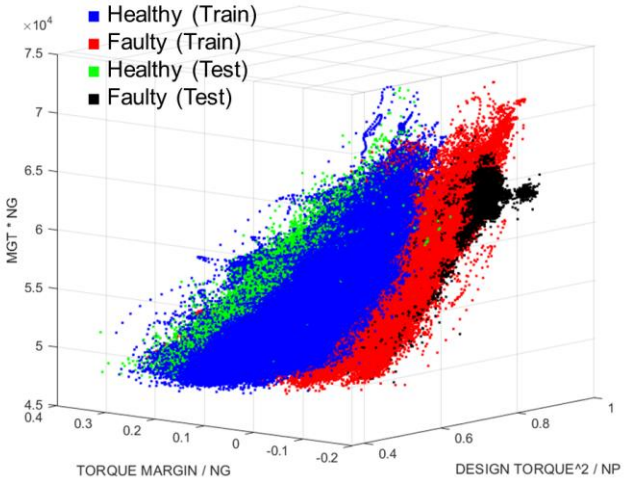
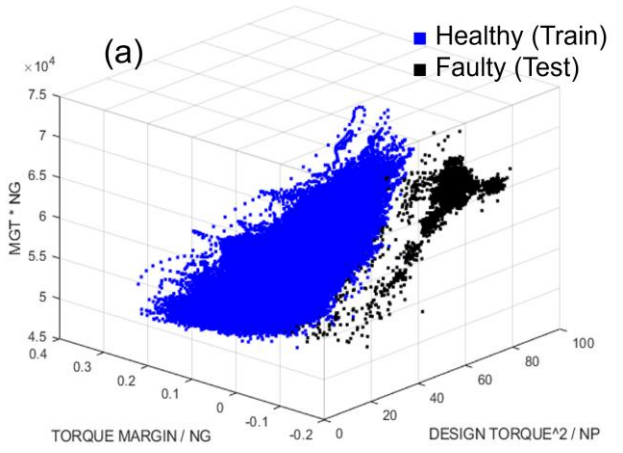


Figure 16 Grouped scatter plot on training and test set. Grouping is done based on the fault labels.

Model will score better if there's minimal overlap between the test-faulty and train-healthy data points with respect to important predictors from the feature ranking section as shown in Figure 17 a & b below.



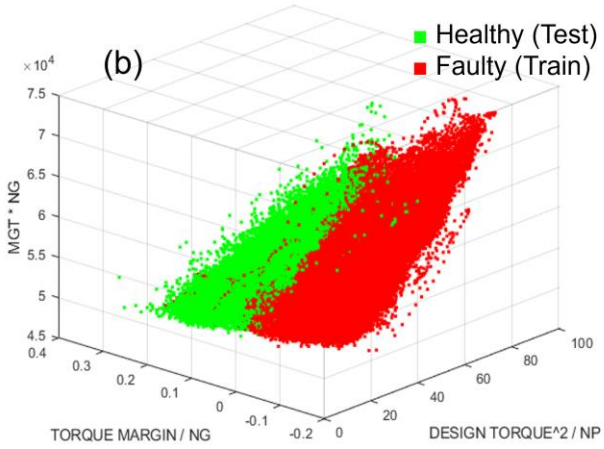


Figure 17. Scatter plots showing separation of healthy and faulty data between training and test sets

Further, Shapley (Lundberg, Scott, & Lee, 2017) plot was used to study the distribution of impact of important features on the model output as shown with Shapley boxplots in Figure 18. The median Shapley values in the box plot to the left or right of 0 and in some cases the locations of the outliers (indicated by circles) help in explaining the faulty predictions made by the model

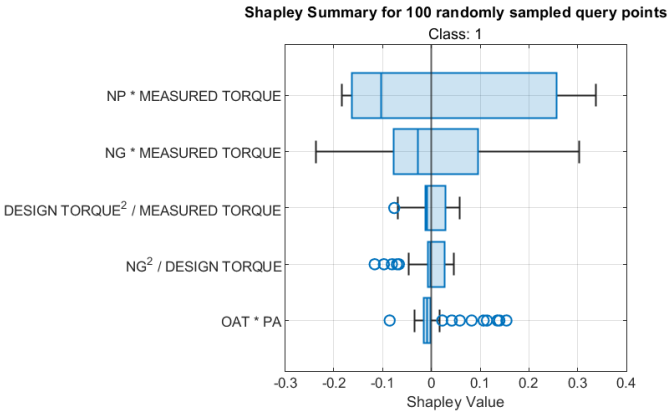


Figure 18. Shapley summary plot for Faulty predictions on 100 randomly sampled query points from the test set

6. CONCLUSION

In this work, we addressed the challenges of large, moderately imbalanced datasets by employing a comprehensive suite of MATLAB tools, including feature engineering, data augmentation, model exploration, and explainability techniques. Our approach demonstrated the successful development of predictive models for helicopter turbine engine health monitoring, with a focus on data integrity, feature selection, and model confidence estimation.

These methodologies not only improved accuracy but are broadly applicable to real-world engine health prediction and data-driven modeling challenges.

As next steps, we recognize the potential for further enhancing our models through hyperparameter optimization, particularly for neural networks and ensemble models. This process, albeit computationally intensive, promises to refine model performance and unlock additional insights. Given the constraints of the competition, we deferred this step, but it remains on the wish list for future work.

7. ACKNOWLEDGEMENT

The authors acknowledge support and encouragement from MathWorks for participating in PHM Data Challenge 2024. Extensive technical computing resources, software tools, IT infrastructure and the collaborative support of fellow MathWorkers are thankfully acknowledged.

8. REFERENCES

Bechhoefer, E., & Hajimohammadali, M. . (2023). Process for Turboshaft Engine Performance Trending. *Annual Conference of the PHM Society*, *15*(1). https://doi.org/10.36001/phmconf.2023.v15i1.3490

MathWorks, "Handle Imbalanced Data or Unequal Misclassification Costs in Classification Ensembles", MathWorks Documentation. [Online]. Available: https://in.mathworks.com/help/stats/classification-with-unequal-misclassification-costs.html

Zhou, Z.-H., and X.-Y. Liu. "On Multi-Class Cost-Sensitive Learning." Computational Intelligence. Vol. 26, Issue 3, 2010, pp. 232–257 CiteSeerX.

Seiffert, C., T. Khoshgoftaar, J. Hulse, and A. Napolitano. "RUSBoost: Improving classification performance when training data is skewed." 19th International Conference on Pattern Recognition, 2008, pp. 1–4.

Federal Aviation Administration, *Helicopter Flying Handbook*, FAA-H-8083-21B, U.S. Department of Transportation, 2019. [Online]. Available: https://www.faa.gov/sites/faa.gov/files/regulations_policies/handbooks_manuals/aviation/faa-h-8083-21.pdf. [Accessed: 20-Sep-2024]

Stoppiglia, Hervé, Gérard Dreyfus, Rémi Dubois, and Yacine Oussar. "Ranking a random feature for variable and feature selection." The Journal of Machine Learning Research 3 (2003): 1399-1414.

MathWorks, "Predictor importance for ensemble models," MathWorks Documentation. [Online]. Available: https://in.mathworks.com/help/stats/classreg.learning.classif.compactclassificationensemble.predictorimportance.html

MathWorks, "Automated classifier selection with Bayesian and ASHA optimization," MathWorks Documentation [Online]. Available: https://www.mathworks.com/help/releases/R2024b/stats/automated-classifier-selection-with-bayesian-optimization.html

Li, Liam, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. "A System for Massively Parallel Hyperparameter Tuning." *ArXiv:1810.05934v5 [Cs]*, March 16, 2020. https://arxiv.org/abs/1810.05934v5.

Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and Regression Trees. Boca Raton, FL: Chapman & Hall, 1984.

Lundberg, Scott M., and S. Lee. "A Unified Approach to Interpreting Model Predictions." Advances in Neural Information Processing Systems 30 (2017): 4765–774.