AN EXERCISE IN DEVELOPING A SUPERVISED ANOMALY DETECTION MODEL WITH AUTOMATICALLY ENGINEERED FEATURES

Rahul Rameshbabu[1], Dushhyanth Rajaram[1], Olivia J. Pinon Fischer[1], Tejas G. Puranik[1] & Dimitri N. Mavris[1]

[1]Aerospace Systems Design Laboratory (ASDL), School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA, 30332, U.S.A

Abstract

With advancements in technology such as Industrial Internet of Things (IIoT), Machine Learning (ML), and cloud computing, data-driven approaches to the health monitoring and anomaly detection of systems are becoming increasingly pervasive. Anomaly detection techniques applied to bearings, in particular, have been widely discussed in the literature due to the prevalence of bearings in many industries. The large body of research on data-driven anomaly detection for bearings has largely consisted of benchmarking various model types and feature engineering methods. This work provides an end-to-end model construction and selection methodology that results in a model of appropriate complexity, and if necessary, automatically engineered features and tuned hyperparameters. While the methodology is demonstrated on a candidate dataset selected from a set of popular datasets used in the bearing anomaly detection literature, this work is expected to be of interest and value to any practitioners addressing anomaly detection problems.

Keywords: anomaly detection, machine learning, feature engineering, hyperparameter tuning

1. Introduction and Background

Anomaly detection and fault monitoring capabilities are critical to ensure the operational safety, timely maintenance, and overall reliability of engineering systems. Among the several potential candidate mechanical components for application of anomaly detection and fault monitoring, rolling contact bearings are arguably the most suitable due to their ubiquitous use. Rolling contact bearings are utilized in a wide array of aerospace systems [1]. Bearings are instrumental to critical components such as engines, gear-boxes, and transmissions. Therefore, a robust and scalable methodology for bearing wear/anomaly detection is desirable in order to minimize downtime and ensure maximum safety and efficiency for these systems.

Due to recent advances in Industrial Internet of Things (IIoT) capabilities, and data infrastructure and storage, large amounts of operational data can now be generated at various levels of detail. The abundance of data coupled with advances in artificial intelligence/machine learning has led to the proliferation of data-driven methods for bearing anomaly detection in addition to anomaly detection of other components [2–8]. To obtain good predictive accuracy, current data-driven efforts in this domain primarily rely on problem-specific feature engineering, which requires significant subject matter expertise and ad-hoc methods to select appropriate model architectures. This work addresses these issues by partially automating the feature engineering process and providing a methodology to construct a well-performing model independent of the components studied and measurements provided.

Anomaly detection for bearings and other components has been studied by many researchers. Given some measured data, this class of problems broadly involves classifying data as either anomalous (faulty bearing) or non-anomalous (nominal bearing). Some researchers have tested popular machine learning techniques such as random forest classifiers, K-means clustering, and support vector machines on this problem [3]. Others have utilized more traditional statistical methods relying on estimating distributions of measured data [4] as well as more advanced state-of-the-art deep learning techniques such as recurrent and convolutional architectures [2, 9].

When solving the problem of bearing anomaly detection, feeding raw sensor inputs directly into the prediction models is problematic due to the fact that raw sensor inputs can be noisy and can lead to inadequate results from several model types. Multiple feature engineering approaches have been implemented [5, 9, 10] to a number of applications, including for anomaly detection purposes. In [9], different sets of features are pre-selected, then the desired predictive accuracy metrics of the models are tested on the validation data and analyzed to select which features to retain. While such an approach produces good models, it may likely require a significant computational effort especially if many features are present in the training data. In [5], a different model is trained for each feature and the features are retained or discarded based on the performance of the model associated with the features. The issues with such approaches is that they do not take into account the interactions between features. Another relatively computationally inexpensive, but extremely popular feature engineering method is Principal Component Analysis (PCA) [10]. Unlike the previously mentioned methods, PCA requires pre-processing of the training data without regard for desired metrics such as validation accuracy. PCA is based on the assumption that features with a high variance will have a better split between classes. This is true for problems with a linear decision boundary; however, a linear decision boundary may not always be available depending on the problem. Another field where feature engineering is common is in the field of deep learning applied to computer vision where combinations of convolutions and max-pooling are used to reduce the dimensionality of the input data and increase computational efficiency [11]. While this approach works well in practice, convolutions and max-pooling operations on the inputs lack interpretability when utilized with a time-series input compared to more popular time-domain features. To address the gaps in the aforementioned feature engineering approaches, this work utilizes various time-domain features and a neural network architecture to learn the significance of these features while simultaneously optimizing weights to classify bearing anomalies on a candidate dataset.

While neural network parameters are trained by optimizing a loss function, the selection of hyperparameters is a problem that is challenging and often left to hand-tuning. The field of hyperparameter optimization deals with finding more efficient ways to optimize hyperparameters and neural networks. A popular hyperparameter tuning method used is Bayesian optimization [12]. Bayesian optimization efficiently trades exploration and exploitation of the hyperparameter space to quickly guide users towards the model configuration that maximizes some overall evaluation criterion like accuracy or likelihood of observing the training data. To improve the efficiency of Bayesian optimization, a separate scheduler is typically employed, such as the *hyperband* algorithm [13]. In the *hyperband* algorithm, a certain number of resources are allocated to each hyperparameter setting. As the settings are evaluated and hyperparameters are down-selected, the budget for each configuration is increased until the stopping criterion of the algorithm is satisfied. A modification to the *hyperband* algorithm is the Asynchronous Successive Halving Algorithm (ASHA). This algorithm improves upon the *hyperband* algorithm by promoting certain hyperparameter configurations earlier if they meet a certain performance criterion. In this work, the optimizer used will combine the benefits of Bayesian optimization and the ASHA algorithm. The Bayesian optimizer is used for efficient searching of the space, and the ASHA algorithm is used for efficient scheduling (to decide how long to evaluate and when to promote specific configurations) [14].

Hence, as discussed, many papers in the anomaly detection field focus on solving the problem of anomaly detection using a specific method [4, 5] or benchmarking multiple methods on various datasets [2, 3]. Unlike these other works, the goal of this work is to present and demonstrate an end-to-end model development methodology that can be followed for any candidate dataset and anomaly detection problem. Initially, some popular datasets are tested on relatively simple classification algorithms to select a candidate dataset for demonstrating the methodology. The candidate dataset selected is the one with the poorest performance. In the next step of the process, relatively complex model types, with and without engineered features, are trained on the candidate dataset. Finally, efficient hyper-parameter optimization is employed in a bid to improve the predictive accuracy. This methodology can be applied by practitioners interested in solving anomaly detection problems in diverse applications.

The structure of the paper is as follows. Section 2 presents an overview and description of the proposed methodology, summary of the data-sets considered for the demonstration, and summary of metrics of interest. Section 3 benchmarks multiple candidate datasets, selects a sufficiently challenging dataset from the pool, applies the proposed model selection methodology to this dataset, and presents the results. Fially, Section 4 summarizes the highlights of the capabilities developed as part of this research and discusses avenues for future work.

2. Methodology

This section presents a methodology that can be followed to select an appropriate model for a given dataset. The steps of this methodology include benchmarking relatively simple classifiers, traditional neural network architectures, and neural networks with automated feature engineering, and finally employing hyperparameter optimization. These steps are illustrated in Figure 1.

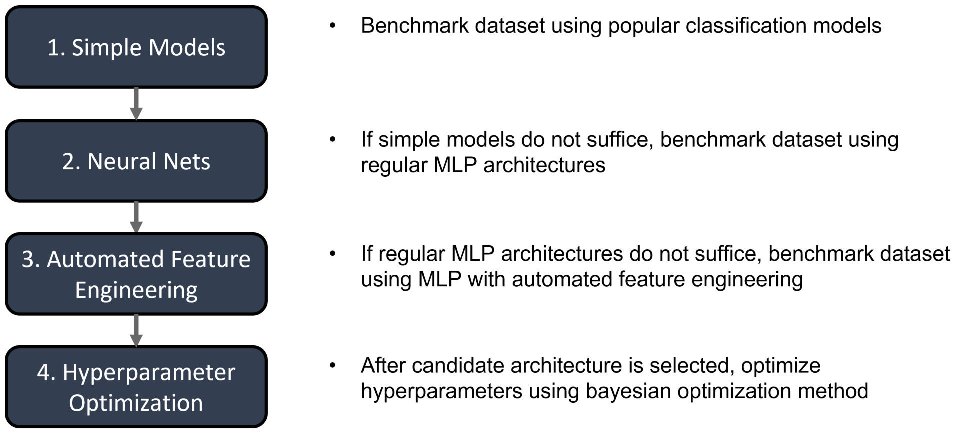
1. Simple Models
Benchmark dataset using popular classification models
2. Neural Nets
If simple models do not suffice, benchmark dataset using regular MLP architectures
3. Automated Feature Engineering
If regular MLP architectures do not suffice, benchmark dataset using MLP with automated feature engineering
4. Hyperparameter Optimization
After candidate architecture is selected, optimize hyperparameters using bayesian optimization method

Figure 1 – Model Selection Methodology

2.1 Datasets

First, a sufficiently challenging dataset was selected to demonstrate the proposed methodology. To accomplish this, multiple datasets were tested using various popular machine learning classifiers. The datasets considered include a number of datasets traditionally used in the bearing anomaly detection literature [2]. A summary of the datasets considered is presented in Table 1.

Dataset	number of features	Type of Fault	Component
CWRU [6]	3 (vibration)	artificial	bearing
MFPT [7]	1 (vibration)	artificial	bearing
PU [3]	3 (current and vibration)	both	bearing, gear-box, and motor
SEU [8]	8 (vibration)	artificial	bearing

Table 1 – Dataset Summary

All of the datasets in Table 1 are from test stands in which bearings (other components in some cases) are outfit with sensors and data is recorded for both anomalous and nominal cases. In the third row of Table 1, the artificial features refers to instances where faults are applied to specific locations of the components. For example, faults can be applied to the inner-raceway, outer race-way, or the ball/roller of a bearing. Alternatively, a combination of the previous faults could be applied as well.

Real faults refer to trials in which the test stand is run until a fault occurs naturally. Next, the datasets were benchmarked using some popular machine learning models.

2.2 Simple Models (Figure 1, Step 1)

As an initial step, simple machine learning models such as K-means clustering, Logistic Classifier, Random Forest Classifier, and a PCA Classifier [15] were used to train models on the aforementioned datasets. These were chosen to represent popular models that are vastly different in terms of their formulation. The models were benchmarked based on their performance (using a variety of metrics) on the binary classification problem of determining whether data is anomalous or nominal. The metrics were calculated using the results of the models in a 5-fold cross-validation experiment. In these experiments the datasets were *stratified* to ensure that the ratio of anomalous data to nominal data was consistent between training and test sets in each fold of the cross-validation.

2.3 Neural Nets (Figure 1, Step 2)

For many datasets, the relatively simple models specified in the previous section were expected to achieve inadequate classification performance. A popular method to solve complex classification problems is using Multilayer Perceptrons (MLP) or neural networks. This approach has been used in the anomaly detection literature for both bearings and other components [2, 9]. The structure of an MLP is shown in Figure 2.

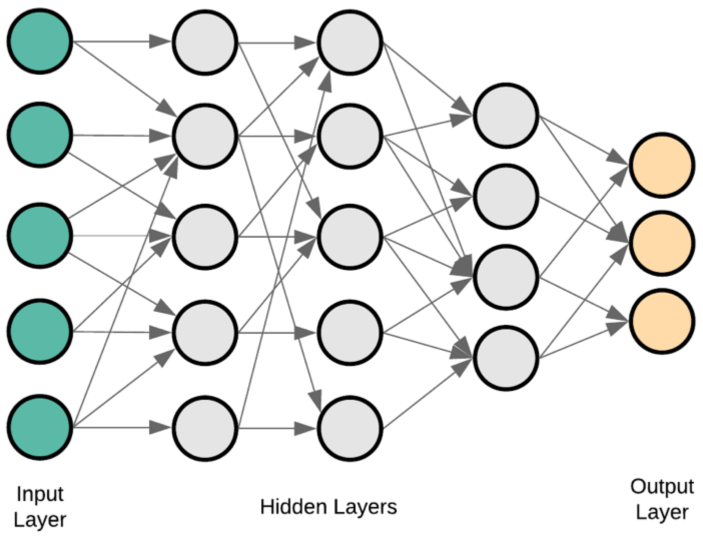

Figure 2 – MLP architecture

Another benefit of MLPs, which is also a large reason for their popularity, is the flexibility of the architecture. The hyperparameters that allow for flexible model definitions include number of layers, nodes per layer, optimizer, the type of loss function, among others. This large number of hyper-parameters allows for MLPs to be a candidate solution for many classification problems of varying levels of complexity. MLPs are used in this work as a relatively complex model type when compared to the previously mentioned simple models.

2.3.1 Time-Series Problem

If classifier performance trained using raw data is not satisfactory, it may be useful to consider the anomaly detection problem as a time series classification (TSC) exercise. A time-series is formally defined as an ordered set of real values with a length equal to the number of values in the set [16]. Considering a dataset $D = [(X_1, Y_1), (X_2, Y_2), \ldots, (X_N, Y_N)]$ as a collection of pairs (X_i, Y_i) where X_i is a time-series with Y_i as its corresponding label, the time series classification task consists on training a classifier to map the labels Y_i to their possible input X_i. Multiple deep learning architectures have been successfully trained on time-series data with results comparable or outperforming the current state-of-the-art models [16]. The benefit of formulating the anomaly detection problem as a time-series problem is that the relationship between sequential data-points in time may be of importance when determining whether that sequence corresponds to a damaged component or a nominal component. Figure 3 illustrates how raw data can be converted to a time-series classification problem. In this example, there are three measurements and each measurement is converted to a set of time-series, where each time-series is of length 3. The length of each time-series is a parameter that must be tuned. Also in this example, an anomalous point is 1 and a nominal point is 0.

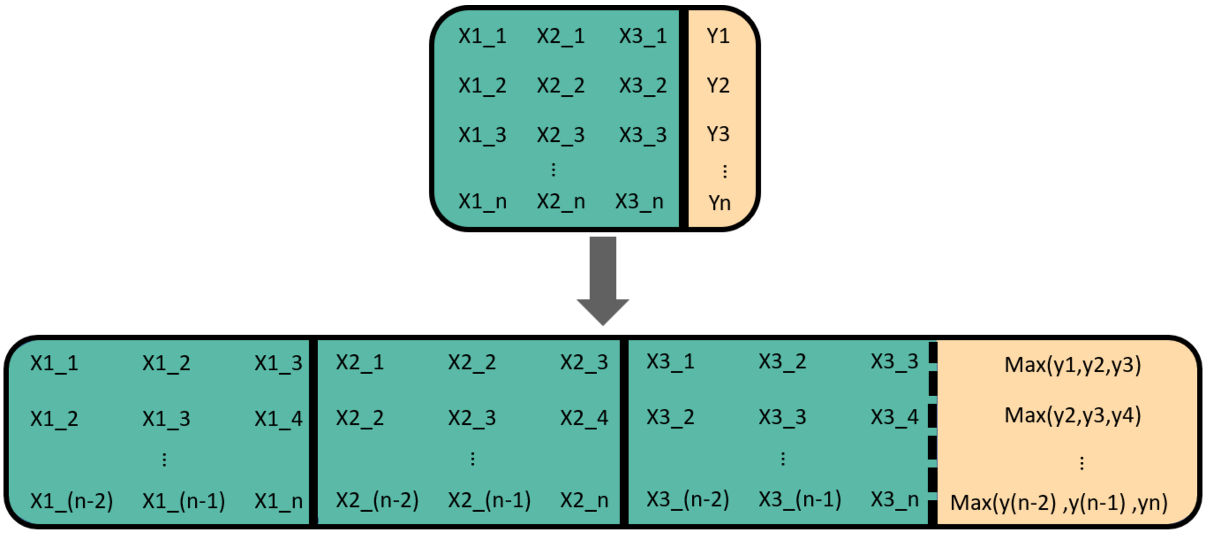
X1_1 X2_1 X3_1 Y1
X1_2 X2_2 X3_2 Y2
X1_3 X2_3 X3_3 Y3
X1_n X2_n X3_n Yn
X1_1 X1_2 X1_3 X2_1 X2_2 X2_3 X3_1 X3_2 X3_3 Max(y1,y2,y3)
X1_2 X1_3 X1_4 X2_2 X2_3 X2_4 X3_2 X3_3 X3_4 Max(y2,y3,y4)
X1_(n-2) X1_(n-1) X1_n X2_(n-2) X2_(n-1) X2_n X3_(n-2) X3_(n-1) X3_n Max(y(n-2) ,y(n-1) ,yn)

Figure 3 – Conversion from Raw Data to Time Series Data

2.3.2 Automated Feature Extraction (Figure 1, Step 3)

We can take advantage of the fact that the datasets considered in this work can be organically represented as time-series data to extract additional features and try improving classification accuracy. Reference [5] lists multiple time-domain features commonly extracted and used from time-series in a bearing anomaly detection context. For the proposed end-to-end process, a subset of these features is selected, as shown in Table 2.

Mean	Max Value	Root Mean Square	Variance	Crest Factor
$\frac{1}{n}\sum_{i=1}^{n} x_i$	$max(\lvert x_i \rvert)$	$\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$	$\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})$	$\frac{Max\ Value}{Root\ Mean\ Square}$
Kurtosis	**Clearance Factor**	**Impulse Factor**	**Shape Factor**	**Peak to Peak**
$\frac{1}{n}\sum_{i=1}^{n}\frac{(x_i-\bar{x})^4}{var^2} - 3$	$\frac{Max\ Value}{(\frac{1}{n}\sum_{i=1}^{n}\sqrt{\lvert x_i \rvert})^2}$	$\frac{Max\ Value}{\frac{1}{n}\sum_{i=1}^{n}\lvert x_i \rvert}$	$\frac{Root\ Mean\ Square}{\frac{1}{n}\sum_{i=1}^{n}\lvert x_i \rvert}$	$max(x_i) - min(x_i)$

Table 2 – Time Domain Features

However, all of these features will have varying levels of importance when it comes to classifying the data. In this work, we modify the standard MLP architecture to have the activation functions of the nodes of the first layer transform the time-series inputs into the features listed in Table 2. Every feature in Table 2 is calculated separately for each measurement in the selected dataset. For example, if a dataset has one current measurement and one vibration measurement, the neural network will calculate a separate mean for current and vibration. These features could be extracted beforehand and directly used as inputs to the neural network. However, the *Keras* library [17] can be leveraged to extract these features directly in the nodes of the MLP. This is useful because feature extraction and training occur seamlessly in the same function call. The described architecture is illustrated in

Figure 4.

In this work, the significance of a feature is defined as the sum of the weights of the connections of that feature node to the first hidden layer. To determine whether this approach is a valid feature engineering strategy for a specific dataset, the following hypothesis is tested: “Let model A be a model trained with the top 5 significant features removed (based on the aforementioned definition of significance). Let model B be a model with the bottom 5 significant features removed. Then, model B will outperform model A in terms of validation accuracy and training loss”.

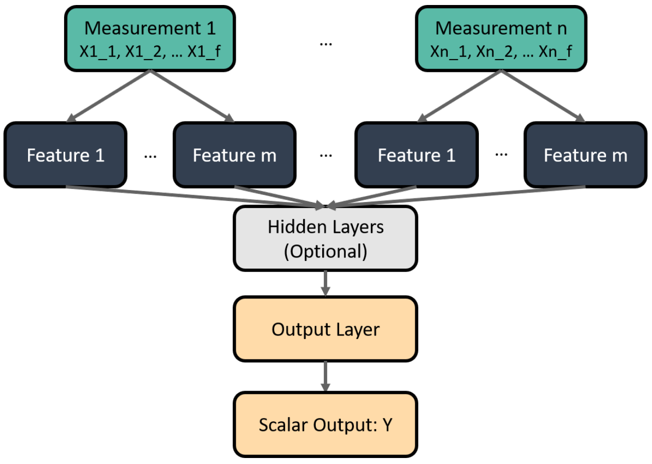

Figure 4 – MLP with automated feature engineering (architecture A)

For a fair assessment of the benefits/detriments, this approach must be compared to the one that solves the time-series classification problem without feature extraction. An MLP architecture for this problem is shown in Figure 5. These two methods will be compared to an MLP with raw features (shown in Figure 6) as input to assess the benefit of using time-series inputs for anomaly detection. For the remainder of the paper, the three MLP architectures described in Figures 4, 5, and 6 are referred to as architectures A, B, and C, respectively.

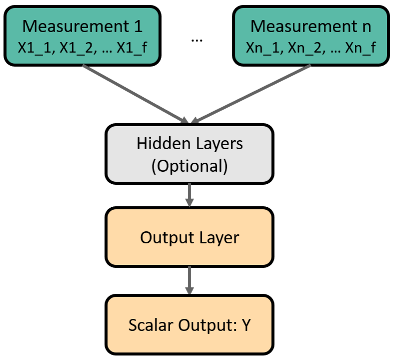

Figure 5 – MLP with time-series inputs (architecture B)

2.4 Metrics of Interest

The metrics used to evaluate these methods are common metrics for binary classification. The most commonly used metric is accuracy (Equation 1), where TP, TN, FP, and FN are used to denote true-positive, true-negative, false-positive, and false-negative, respectively. This is a good indication of classifier performance when the datasets are balanced, i.e., when there is a comparable number of nominal labels and anomalous labels. For imbalanced datasets a popular metric to use is balanced accuracy (Equation 2). Both Equations 1 and 2 indicate that the balanced accuracy provides a more

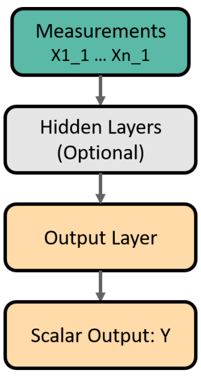

Figure 6 – MLP with raw inputs (architecture C)

robust metric of model behavior when the two classes are imbalanced.

$$Accuracy = \frac{TP+TN}{TP+FN+TN+FP} \tag{1}$$

$$Balanced\ Accuracy = \frac{\frac{TP}{TP+FN}+\frac{TN}{TN+FP}}{2} \tag{2}$$

2.5 Hyperparameter Optimization (Figure 1, Step 4)

As stated in the introduction, the optimizer used combines the benefits of Bayesian optimization and that of the ASHA algorithm [14]. The implementation of this optimizer is done utilizing the *Hyperopt* library, which allows for the specification of separate algorithms for search and scheduling [18]. To run the optimizer, first the architecture is defined using the number of layers and type of layer as the parameters. For a single neural network architecture, there exists both layer-specific parameters and general parameters that can be optimized. Table 3 shows the hyperparameters to be optimized as part of this work. This table is by no means exhaustive. Indeed, other parameters exist that are associated with other model types such as convolutional and recurrent architectures.

Non-layer specific	Dense Layers	Dropout Layers
Optimizer Type	Number of Nodes	Dropout Rate
Learning Rate	Activation Function	
Batch Size		
Optimizer Specific Parameters		

Table 3 – Search Space

3. Experiments and Results

First, the candidate datasets are benchmarked using popular classifiers. For the PU data-set only, the current measurements were utilized (vibration measurements were excluded) to try and replicate a real motor health detection scenario as much as possible. Current measurements are more realistic because it is generally easier to outfit standard motors with current sensors than to outfit the rotating components with accelerometers to measure vibration [3].

3.1 Simple models

From the results presented in Table 4, it is clear that the PU dataset consisting of naturally occurring anomalies is a difficult dataset for these classifiers. The difficulty in achieving good classification accuracy in addition to it being the only dataset with current measurements makes this dataset a good candidate for the more advanced MLP-based models and the rest of the methodology.

Data-set	Logistic Classifier	Random Forest Classifier	K-means Clustering	PCA Classifier
CWRU	0.67	0.74	0.5	0.67
MFPT	0.89	0.86	0.58	0.89
PU	**0.51**	**0.61**	**0.50**	**0.50**
SEU	0.99	0.99	0.3	0.5

Table 4 – Model Accuracy across the Datasets Considered

3.2 MLP Experiments

In these experiments, architectures A, B, and C are trained to solve the anomaly detection problem for the PU data-set. Every model is trained 5 times and the hyperparameters are roughly hand-tuned. The hyperparameters and their values used for the purpose of these experiments are shown in Table 5. Early stopping was employed for termination, i.e, the training stopped when the loss function did not change by a tolerance for a specified number of epochs. For architectures A and B, different models were trained for different time-series input lengths.
The metrics observed for these experiments were validation accuracy and binary cross-entropy loss. Binary cross-entropy loss is the standard loss function for training MLPs for binary classification [19]. Since the PU dataset happens to be balanced, we do not require the balanced accuracy metric as it will be approximately equal to regular accuracy. Augmenting this process for imbalanced datasets will be addressed in future work.

Parameter	Value
Nodes	50
Batch Size	100
Hidden Layer Activation Function	Relu
Output Layer Activation Function	Sigmoid
Loss Function	Binary Cross-entropy
Optimizer	Adam
Learning Rate	0.005
early stopping tolerance	0.1
early stopping epochs	5

Table 5 – Hand Tuned Hyper-parameters

3.2.1 MLP with Raw Features

For the first experiment, the performance of architecture C on classifying the PU dataset is tested. The resulting accuracy and loss of the experiment are both 0.63.
It is clear from these results that, while the results improve slightly from the best simple model (random forest) tested on the PU dataset, there is much room for improvement. The next step in the process is to test out the predictive performance of MLP architectures A and B.

3.2.2 MLP with Time Series Input

The accuracy and loss values for architectures A and B using different time-series input lengths are shown in Figures 7 and 8, respectively. The results are also tabulated in Table 6. For this particular dataset, architecture A outperforms architecture B in almost the whole range of time-series input length. The difference in performance increases as the lengths of input time-series are smaller. For almost all time-series input lengths tested, architectures A and B vastly outperform architecture C and the simple models with raw feature inputs and all other models previously tested. The only configuration that did not perform as well was architecture B with a time-series input length of 25 time-steps.

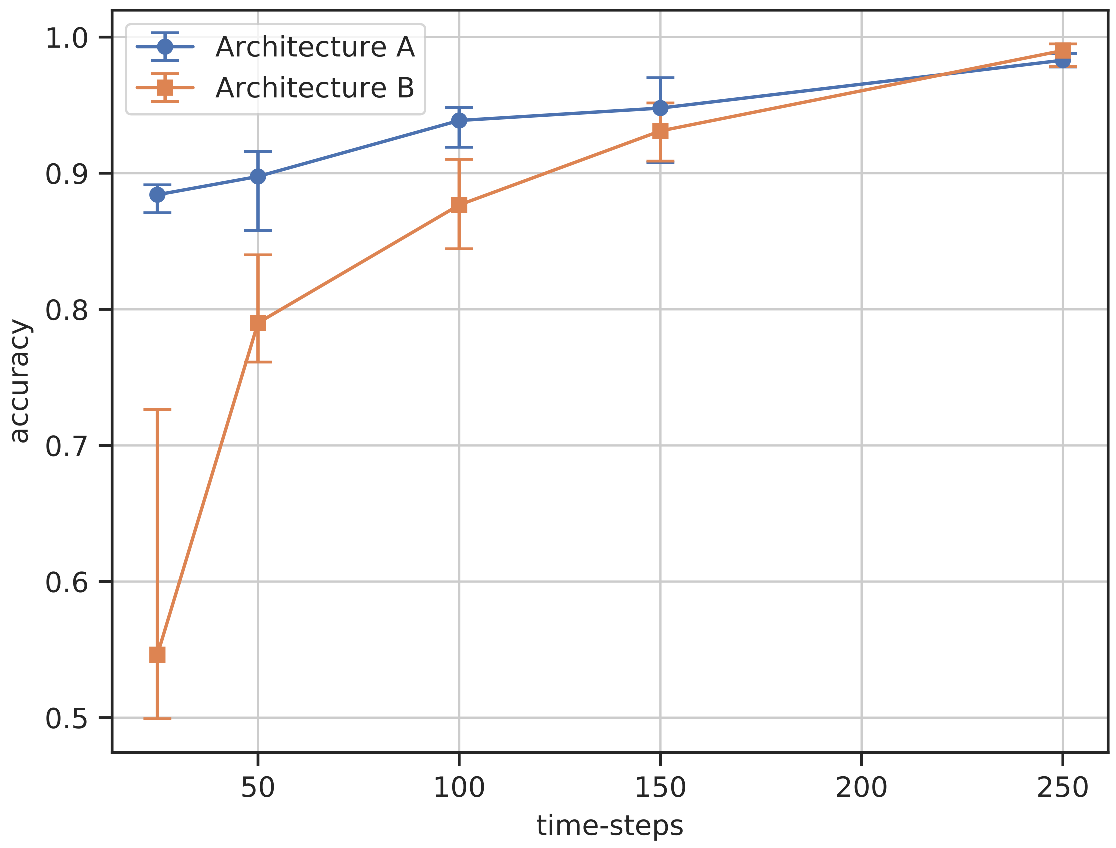

Figure 7 – Accuracy vs. Time-steps for Architectures A and B

time-steps	Architecture A Accuracy	Architecture B Accuracy	Architecture A Loss	Architecture B Loss
25	0.88	0.55	0.24	0.64
50	0.90	0.79	0.21	0.44
100	0.94	0.88	0.15	0.26
150	0.95	0.93	0.11	0.15
250	0.98	0.99	0.054	0.036

Table 6 – MLP with Time-Series Input

Now that we have some candidate models that have significantly higher accuracy than models previously tested, we shall look into feature engineering.

3.2.3 Significance of Features

The purpose of the following experiment is to test the hypothesis stated in Section 2.3.2. To that end, for time-series inputs of various lengths, multiple sub-architectures are constructed with various features omitted in each one. The accuracy and loss of these models are then compared. The accuracy and loss values are averaged over 5 trials for each model, with random initialization of the datasets. Note that the datasets are a stratified train-test split of 75% to 25%. Table 7 shows the significance of the features for the classifier trained with 100 time-steps as the input time-series dimension. These significance values, which were extracted from the experiments described in Section 3.2, are also averaged over 5 trials. From Table 7 it can be observed that the most significant feature for both measurements is the variance, and one of the least significant features for both measurements is the kurtosis. This may be an indication that variance of time-series data is the best feature to distinguish between nominal and anomalous data points, and kurtosis is one of the worst features to accomplish

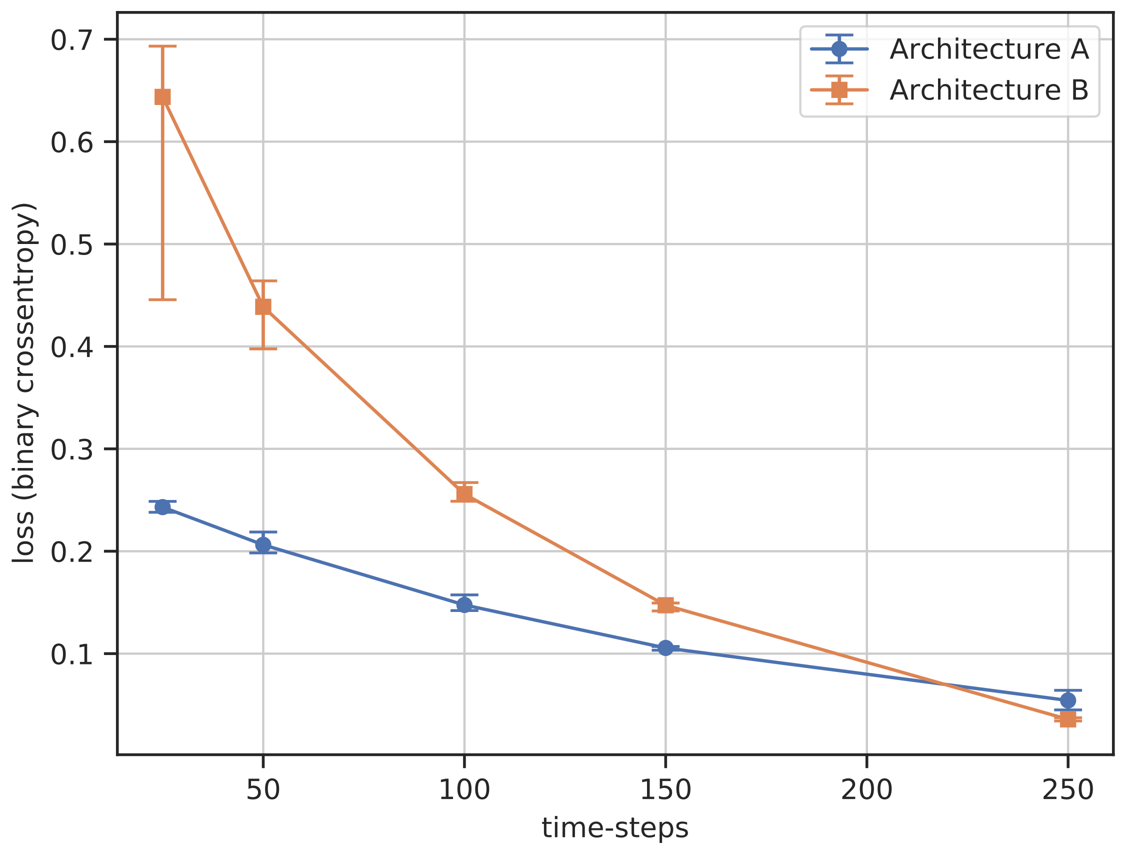

Figure 8 – Loss vs. Time-steps

the same task.

Feature	Mean	Max Value	Root Mean Square	Variance	Crest Factor
Current 1	8.43	4.59	5.69	32.56	10.63
Current 2	12.69	4.48	3.55	26.86	13.08
Feature	**Kurtosis**	**Clearance Factor**	**Impulse Factor**	**Shape Factor**	**Peak to Peak**
Current 1	4.15	7.04	9.55	19.18	3.31
Current 2	1.16	13.04	13.40	18.92	6.77

Table 7 – Significance of Extracted Features

For each time-series input dimension, two models are trained. First, a model with the bottom five significant features omitted, i.e., with the top 15 significant features retained. Second, a model with the top five significant features omitted, which means the bottom 15 significant features retained. For example, in Table 7, the top 5 features are variance of Current 1, variance of Current 2, shape factor of Current 1, shape factor of Current 2, and impulse factor of Current 1. The bottom 5 features are kurtosis of Current 2, peak to peak value of Current 1, root mean square of Current 2, kurtosis of Current 1, and max value of Current 2. Two separate models are trained with the aforementioned top 5 features and bottom 5 features omitted. The results of these experiments are shown in Figures 9 and 10.

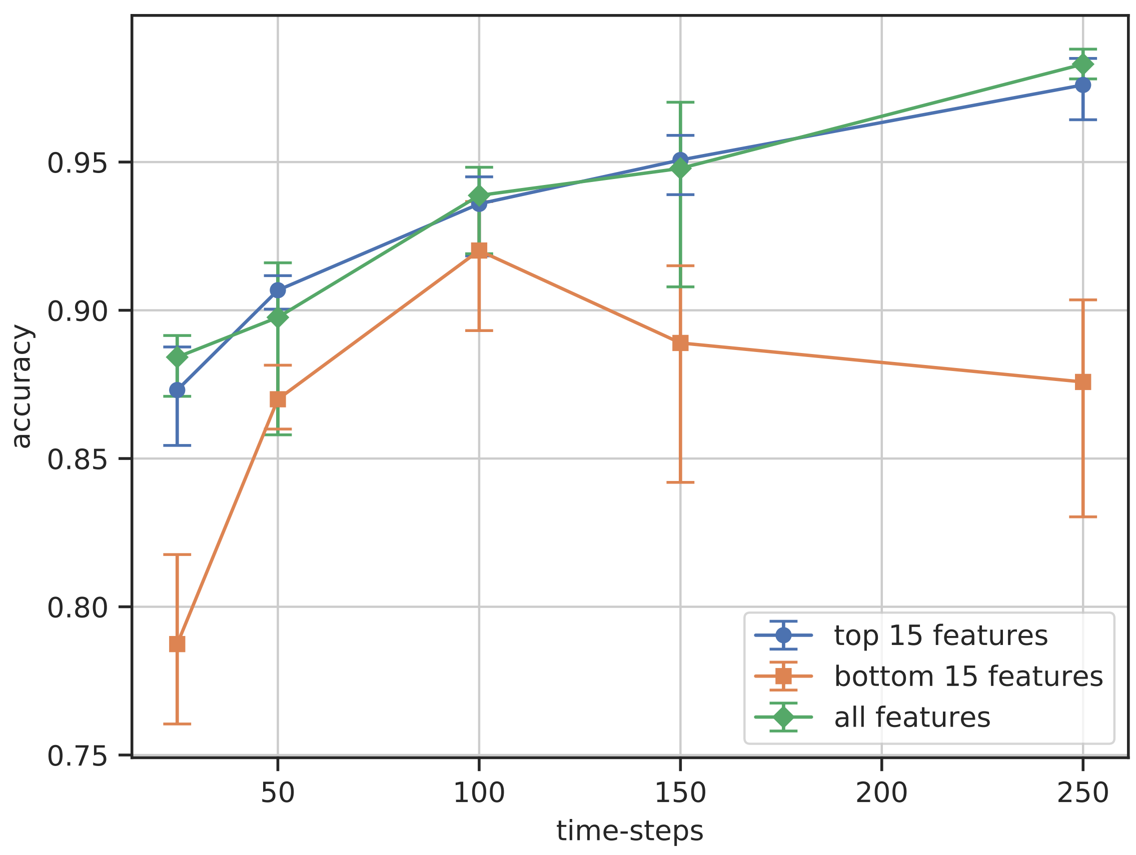

Figure 9 – Accuracy vs. Time-steps

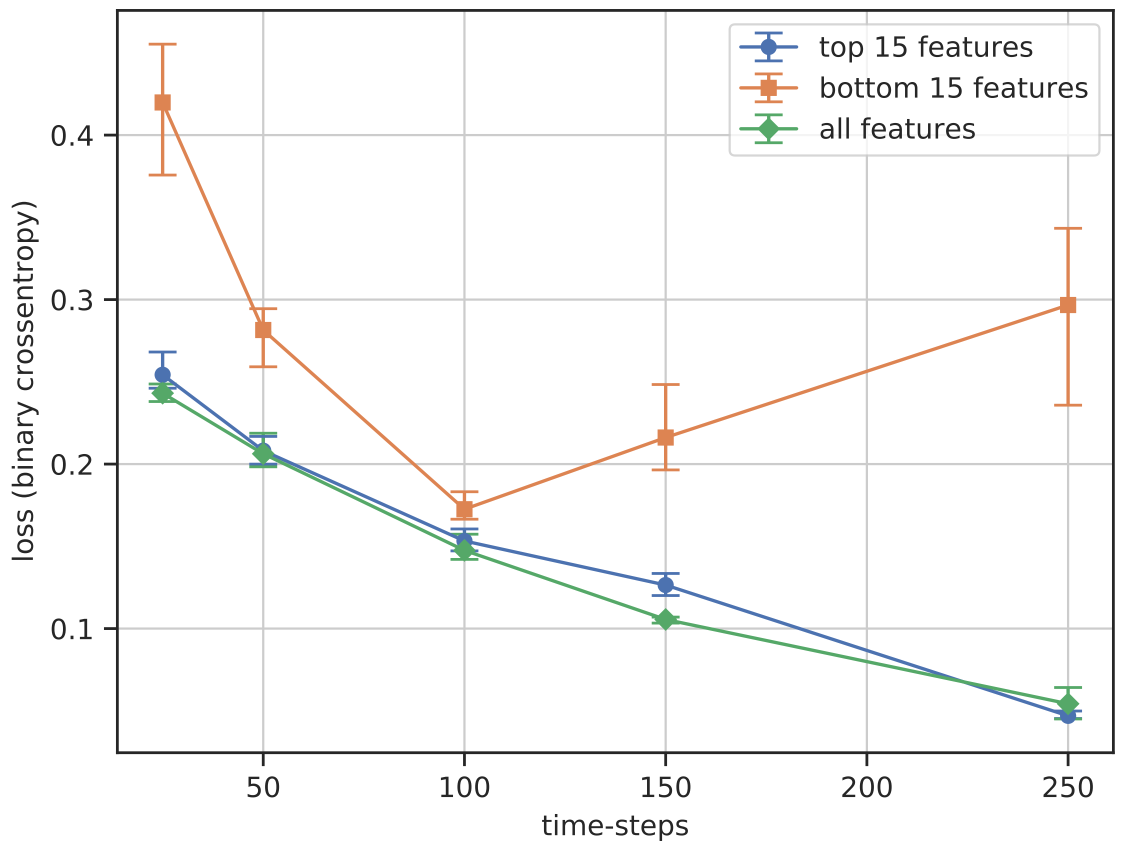

Figure 10 – Loss vs. Time-steps

These results show that across all time-step values, the model trained with the top 15 features outperforms the model with the bottom 15 features. Also, the difference in performance for the model trained with the top 15 features and the model trained with all the features is relatively small. These observations hold for both the validation accuracy and the training loss. These results show that the hypothesis mentioned in section 2.3.2 is valid for this specific dataset. This is important because it indicates that inference with such models for an anomaly detection application may work with data of a smaller dimension and yield similar results, thereby lowering the computational cost of performing inference. While the hypothesis validity holds true for this dataset, it is likely dataset specific. Consequently, similar experiments should be run for the practitioner's specific use-case before reducing the dimension of data during inference based on significance.

3.3 Hyperparameter tuning

To demonstrate the hyperparameter tuning step of the proposed end-to-end process, a candidate model was chosen from architecture A. The model was chosen with time-series input length of 25 time-steps, which had the poorest predictive accuracy, i.e., has the most room for improvement with the Bayesian optimizer. The model was optimized using the framework described in the Methodology section. The search space considered is shown in Tables 8 and 9. The results of the optimization are shown in Table 10.

Optimizer Type	Learning Rate	Batch Size (Relative to Input Size)	Momentum (SGD only)
Adam, SGD	$[\log(0.00025), \log(0.25)]$	$[0.00002, 0.0005]$	$[0.05, 0.25]$

Table 8 – Search Space for Experiments (Not Layer Specific)

$\frac{\text{Number of Nodes}}{\text{Number of Features}}$	Activation Function
$[0.1, 10]$	Relu, Sigmoid, Hyperbolic Tangent

Table 9 – Search Space for Experiments (Dense Layers)

Model	*accuracy*	*loss*
Original	0.88	0.24
Optimized to minimize training loss (500 evaluations)	0.89	0.23
Optimized to maximize validation accuracy (500 evaluations)	0.90	0.21

Table 10 – Optimized Model

Table 10 shows that the optimization improved upon the original model. The best performing model was optimized with the goal of maximizing the accuracy of the model on the validation set. This resulted in a 2% increase in validation accuracy and a 12.5% improvement in loss. The optimal parameters for the different models are shown in Table 11.

Model	*Optimizer Type*	*learning rate*	*batch size*	*activation function*	*number of nodes*
Original	Adam	0.0005	100	Relu	50
Optimized to minimize training loss (500 evaluations)	Adam	0.0042	84	Relu	101
Optimized to maximize validation accuracy (500 evaluations)	Adam	0.00084	80	Hyperbolic Tangent	127

Table 11 – Optimized Model Parameters

4. Conclusion

This work presented a methodology in which a suitable model for anomaly detection is constructed for a given dataset. In particular, this work provides a stencil for practitioners to follow for solving time-series classification problems for any application. The methodology involves starting with a variety of simple models before moving on to more complex models, which can then be further tuned through automated feature engineering and hyperparameter optimization. The methodology is applied on a candidate dataset that is challenging to model using conventional data-driven approaches to demonstrate that it can lead to an effective and parsimonious model with good predictive accuracy. This work also demonstrated an automated, interpretable feature engineering approach for MLPs. While this method of automated feature engineering was validated on a bearing anomaly detection problem, it is expected to provide modelers with the means to investigate performance and efficiency improvements for MLPs, for any application, in an interpretable manner.

As discussed, this methodology first explores simple models before moving on to more complex models, automated feature engineering, and finally hyperparameter optimization. Future work includes quantifying the computational complexity of the algorithms utilized, investigating techniques for working with imbalanced datasets, providing mathematical justification of the automated feature engineering approach, and investigating the significance of frequency domain features.

5. Contact Author Email Address

Olivia J. Pinon Fischer: olivia.pinon@asdl.gatech.edu

6. Copyright Statement

References

[1] Franz-Josef Ebert. An overview of performance characteristics, experiences and trends of aerospace engine bearings technologies. *Chinese Journal of Aeronautics*, 20(4):378–384, 2007.

[2] Zhibin Zhao, Tianfu Li, Jingyao Wu, Chuang Sun, Shibin Wang, Ruqiang Yan, and Xuefeng Chen. Deep learning algorithms for rotating machinery intelligent diagnosis: An open source benchmark study. *ISA Transactions*, 107:224–255, Dec 2020.

[3] Christian Lessmeier, James Kimotho, Detmar Zimmer, and Walter Sextro. Condition monitoring of bearing damage in electromechanical drive systems by using motor current signals of electric motors: A benchmark data set for data-driven classification. 07 2016.

[4] Yuanzhi Huang, Eamonn Ahearne, Szymon Baron, and Andrew Parnell. An evaluation of methods for real-time anomaly detection using force measurements from the turning process. *arXiv preprint arXiv:1812.09178*, 2018.

[5] J. Kimotho and W. Sextro. An approach for feature extraction and selection from non-trending data for machinery prognosis. 2014.

[6] Case western reserve university (CWRU) bearing data center. https://csegroups.case.edu/bearingdatacenter/pages/do
data-file/, 2019.

[7] Society for machinery failure prevention technology. https://www.mfpt.org/fault-data-sets/, 2019.

[8] SEU gearbox datasets. https://github.com/cathysiyu/mechanical-datasets, 2019.

[9] Tejas Puranik, Aroua Gharbi, Burak Bagdatli, Olivia Pinon Fischer, and Dimitri N Mavris. Benchmarking deep neural network architectures for machining tool anomaly detection. 2020.

[10] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1):37–52, 1987. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[12] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*, 2012.

[13] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[14] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pages 1437–1446. PMLR, 2018.

[15] Junwen Wu and Xuegong Zhang. A pca classifier and its application in vehicle detection. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, volume 1, pages 600–604 vol.1, 2001.

[16] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[17] Francois Chollet et al. Keras, 2015.

[18] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *25th annual conference on neural information processing systems (NIPS 2011)*, volume 24. Neural Information Processing Systems Foundation, 2011.

[19] Andreas Buja, Werner Stuetzle, and Yi Shen. Loss functions for binary class probability estimation and classification: Structure and applications. *Working draft, November*, 3, 2005.