Predictive Analysis of Wind Turbine Damage Through Machine Learning Models Utilizing Aerodynamic Pressure and Time Series Data

Zhengxu Li, Lucien Walewski

December 2023

Contents

Abstract

This paper investigates the use of machine learning models for understanding the severity of damage in wind turbine blades. Various feature extraction methods are explored and a range of classifiers are employed to optimize accuracy and recall. This research finds that the combination of Minirocket with the CatBosst classifier achieves the best performance. This research demonstrates the efficacy of these systems in improving the efficiency and reducing the cost of wind turbine maintenance.

1 Introduction

As the field of wind energy advances, the scale of wind turbines have significantly increased, leading to greater energy capture and cost efficiency in wind farm operations [20]. This evolution, however, introduces complex aeroelastic challenges and higher aerodynamic loading on rotor blades, demanding a deeper understanding of aerodynamics and their interplay with wind turbine structure. Contemporary research emphasizes the need for extensive data to comprehend these dynamics, particularly concerning wind speed and the angle of attack on blades [3]. This data, crucial for design and maintenance, remains limitedly accessible to the broader research community.

Addressing this gap, an ETH spin-off RTDT has developed the AeroSense system, an innovative technology that is a wireless, non-intrusive measurement system designed for large-scale, operational wind turbine blades. Featuring a variety of sensors, it provides crucial data for creating a digital twin of monitored turbines.

With rotor components forming a significant portion of wind turbine capital expenses and being prone to various failures, early detection of structural damage is paramount [17]. Structural health monitoring (SHM) methods typically include acoustic emissions, thermographic measurements, and other techniques [8], but the potential of utilizing aerodynamic pressure distribution for identifying structural damage remains largely unexplored [2].

This study aims to bridge this gap by investigating if structural damage, typically in the form of cracks, can be detected through sectional aerodynamic pressure data. We conducted wind tunnel experiments using the AeroSense system to record aerodynamic pressures across different conditions and structural states on a heaving airfoil. This research proposes a pipeline that leverages signal processing and machine learning, to identify and classify the severity of structural damage.

2 Background

RTDT previously collected data by conducting wind tunnel experiments in an EPFL laboratory. Figure 1 displays the experimental setup. Experiments consisted of placing the blade with the Aerosense node in the wind tunnel and

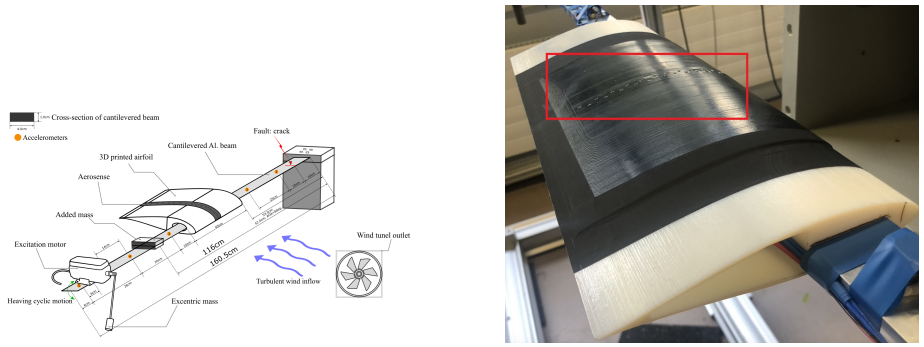

Figure 1: Experimental setup. Figures taken from [11]

taking measurements through the barometers on the node. There are forty barometers on the node, although two malfunctioned during the experiments so experiments contain readings from thirty-eight sensors. Readings are sampled at 100Hz with experiments lasting on average 150 seconds. A motor positioned at the end of the blade is used to excite the blade. The conditions that are varied in the experiments include: wind speed, angle of attack and the heaving frequency of the motor. This is summarized in table 1. For each experiment, the first 20 seconds are discarded to remove the effects of the initial conditions. As the system operates in a steady-state for time scales of the order of a few seconds, we subdivide the time series data obtained into overlapping windows of 20 seconds with overlaps of 10 seconds in order to produce 11 windows per experiment. There are three experiments per test series.

Test series	Wind speed (m/s)	Angle of attack (°)	Heaving frequency (Hz)
1	10	0	1.0
2	10	0	1.9
3	10	8	1.0
4	10	8	1.9
5	20	0	1.0
6	20	0	1.9
7	20	8	1.0
8	20	8	1.9

Table 1: Overview of experimental series

The data given to us was mostly very clean. Therefore, few preprocessing steps were required. Two of the forty sensors failed during the experiments so the data collected from these two sensors was not usable and dropped. Otherwise, there was no major issues with inconsistent data, duplicates or other significant errors.

3 Methodology

In this study, we crafted our methodology into two critical phases: feature extraction and model implementation. Initially, we concentrated on the first phase, which involves the extraction of relevant and representative features from the raw time series data. This step is crucial as it lays the groundwork for the subsequent phase. Following this, the second phase integrates machine learning classifiers, which are applied to the previously extracted features. This systematic approach ensures a cohesive and targeted analysis, leveraging the strengths of both feature extraction techniques and advanced classification methods to yield insightful results.

3.1 Feature Extraction

For feature extraction from raw time series data, we implemented an independent feature extractor for each sensor in every split window. This method ensures a thorough analysis by treating each sensor's data individually. The features extracted are then organized into different categories, as we will describe in the following subsections. To further refine these features, each category underwent a post-processing stage where a feature selector and scaler were applied. This step is essential to enhance the utility and accuracy of the features for the subsequent machine learning models.

3.1.1 Statistical Features and Signal Processing Features

In each segmented window of our analysis, we computed a comprehensive set of traditional statistical features, encompassing mean, maximum, minimum, range, variance, skewness, kurtosis, and percentile values. This foundational statistical analysis provides a robust overview of the data's characteristics. Alongside these, we also employed advanced signal processing techniques to transform the aerodynamic pressure data from the time domain to the frequency domain. This transformation allowed us to extract crucial frequency-related features. Key implementations in this regard include the Fast Fourier Transform (FFT) and the Discrete Wavelet Transform (DWT). Additionally, we calculated various other metrics such as power, frequency band power, Shannon entropy, and Signal-to-Noise Ratio (SNR), further enriching our feature set and providing a deeper understanding of the underlying data patterns.

3.1.2 MINIROCKET

MiniRocket, an acronym for Minimally Random Convolutional Kernel Transform, is an evolution of the earlier Rocket algorithm for time series classification, as noted in [9]. It mirrors Rocket's state-of-the-art accuracy but excels in processing speed, particularly for large datasets. The fundamental difference between MiniRocket and its predecessor lies in its kernel usage. While Rocket relies on random convolutional kernels, MiniRocket utilizes a smaller, predefined set of kernels, lending a near-deterministic quality to its operations. This

strategic modification significantly slashes computational complexity without compromising on accuracy. The main attributes of MiniRocket — its efficiency and predictability — make it an ideal choice for analyzing large-scale time series data, especially in scenarios constrained by time or computational resources. In our study, we explored MiniRocket's efficacy by experimenting with varying numbers of kernels for feature extraction, paired with a simple, fixed classifier. Our experimentation led us to identify 1600 as the optimal number of kernels for our specific project requirements.

3.1.3 DNN features

Recent studies have highlighted the efficacy of Deep Neural Networks (DNNs) in handling predictive tasks with time-series data [6]. Capitalizing on the complex feature representations generated by DNNs, we adapted these models into deep feature extractors. This was achieved by removing the output layers of the DNNs and harnessing the values from the last hidden layers as features, a technique applicable to various DNN architectures. In our research, we specifically implemented three types of DNN variants, each well-regarded for their performance in time-series prediction: Convolutional Neural Networks (CNNs), Long Short-Term Memory networks (LSTMs), and Variational Autoencoders (VAEs).

3.1.4 CNN

CNNs excel in feature detection, both in image processing, where they identify edges or shapes, and in time series data [17], where they detect spikes, drops, or recurring patterns. Their ability to capture local dependencies is vital in time series analysis, offering insights into the immediate context of data points. Additionally, CNNs possess a degree of scale invariance, making them effective for time series data that varies in scale. This is a significant advantage over traditional time series models that often require extensive manual feature engineering. CNNs have the distinct capability to autonomously learn and identify the most pertinent features directly from the data.

Regarding our CNN implementation for this study, the architecture begins with a series of convolutional layers (64, 128, 256 filters) combined with MaxPooling and BatchNormalization layers, designed to capture local dependencies and patterns in the data. These are followed by a Flatten layer that transforms the 2D feature maps into a 1D vector, facilitating the transition to fully connected Dense layers. The network includes dropout layers for regularization to prevent overfitting. The final output is a Dense layer with 6 units(corresponding to the 6 classes), using softmax activation, and Adam optimizer. We use the last hidden layer of the CNN as extracted features.

3.1.5 LSTM

LSTM networks are highly effective for time series data analysis, as they can capture long-term dependencies and circumvent the vanishing gradient issue often

seen in traditional recurrent neural networks [19]. Their intricate architecture, featuring input, forget, and output gates, meticulously controls information flow, making them particularly suited for time series forecasting.

In our project, we utilized an encoder-decoder LSTM structure. The encoder starts with an input layer tailored for aerodynamic pressure time series data, followed by three LSTM layers. A dense layer then condenses this data into a 128-size latent vector, providing a compact representation of the input. This encoder is mirrored by a decoder that uses the latent vector to reconstruct the input signal. The latent vector, encapsulating key features, was extracted for subsequent analysis.

3.1.6 VAE

Variational Autoencoders (VAEs) are powerful tools for feature extraction in time series data [15], thanks to their probabilistic approach to encode inputs into a latent space. Unlike traditional autoencoders, VAEs don't just compress data, they learn the parameters of a probability distribution representing the data. By sampling from this distribution, VAEs can generate new data points, and reconstruct the input based on the sampled latent variables. We used the latent space of VAE for feature learning, which can extract the essential features of multivariate time-series data effectively.

When it comes to our implementation of VAE feature extractor, we implemented also an encoder-decoder architecture. The encoder, processing input time series from 38 sensors with 2000 measurements each, first flattens and then compresses the data through three dense layers into a latent space represented by mean and log-variance vectors. This is facilitated by a custom Sampling layer that introduces stochasticity by adding Gaussian noise, essential for the VAE's generative capabilities. The decoder reconstructs the original time series from the latent space using symmetric dense layers, culminating in a reshaped output matching the input dimensions. The loss function employed Kullback-Leibler divergence, guiding the model to learn a meaningful and structured latent space.

3.1.7 Feature Selection

For each category of features extracted, we applied post-processing steps involving feature selection and scaling. The feature selection was conducted using *VarianceThreshold* and *SelectKBest*, while scaling was performed with *StandardScaler* and *RobustScaler*.

VarianceThreshold eliminates features below a certain variance threshold, effectively removing features with low variance which are less informative for classification purposes. *SelectKBest*, a feature selection method from the scikit-learn library [1], selects the top k features based on their scores from a statistical test, in this case, the ANOVA f-value. This process helps in discarding irrelevant or minor features, potentially improving model performance by focusing on more significant data and reducing the risk of overfitting.

The final step involved standardizing the selected features using *Standard-Scaler* and *RobustSccaler*(which is robust to outliers). This is crucial as many machine learning algorithms, particularly those involving distance calculations, assume features to be centered around zero with comparable variance. Scaling the features ensures no single feature disproportionately influences the model. Additionally, in models utilizing gradient descent, feature scaling can expedite convergence.

3.2 Model Implementation

In the previous part, various feature extraction techniques were employed, each yielding distinct sets of features. After that, each set of features is separately input into a collection of classifiers. Those diverse classifiers could capture the intricacies of extracted features from different perspectives and ensembling their results could lead to a better global clustering accuracy [16].

We obtain a pipeline from each combination of preprocessing and classifier, we conducted a grid search to optimize the hyperparameter settings for each pipeline. A comprehensive comparison of these configurations will be presented in Section 5 of this report. Next, we will detail the classifiers employed in our study.

3.2.1 AdaBoost

AdaBoost, short for Adaptive Boosting, is a highly effective classifier, it works by combining multiple weak classifiers, typically decision trees, to form a strong classifier [12]. Each weak classifier is trained on the same dataset, but the weights of incorrectly classified instances are adjusted, making the algorithm focus more on difficult cases in subsequent iterations. AdaBoost is known for its robustness and adaptability, making it a strong candidate for handling the intricacies and complexities of time series data.

In the implementation of the code, we fine-tuned parameters including estimator(*DecisionTreeClassifier*, *SupportVectorClassifier*, *SGDClassifier*, *BayesianRidge*), algorithm(*SAMMES*,*SAMME.R*), learning rate, and number of estimators.

3.2.2 CatBoost

CatBoost is a versatile and powerful gradient-boosting classifier, renowned for its efficiency in handling categorical features [10]. CatBoost's unique ordered boosting technique is designed to minimize overfitting which is achieved by training the model on one subset of data while calculating residuals on another, thus preventing target leakage and enhancing model robustness. Additionally, it supports scalability and distributed training, which is beneficial for our case.

We fine-tuned *maximum depth*, *learning rate*, *L2 regularization* parameters.

3.2.3 Gaussian Process

The Gaussian Process Classifier is a non-parametric, probabilistic classification model within the realm of Bayesian machine learning, it operates by assuming a Gaussian distribution over functions, enabling it to model the underlying probability densities of classes [18]. One of the key strengths of GPC is its ability to fully define its behavior through second-order statistics, particularly the covariance function which dictates various aspects of the process, such as stationarity, smoothness, and periodicity. The GPC's ability to adapt its behavior based on these characteristics makes it highly flexible and capable of handling a variety of data distributions effectively. Hence we took GPC into our classifier candidate list, in the step of gridsearch, we tried different type of kernels including *RationalQuadratic*, *WhiteKernel*, *Radial Basis Function Kernel* and *DotProduct*.

3.2.4 Gradient Boosting

Gradient Boosting(GB) is an ensemble algorithm, known for effectiveness in creating strong predictive model from an ensemble of simpler models like decision trees [13]. Its unique approach involves sequentially correcting errors from previous stages by fitting decision trees to the negative gradient of the loss function, which can be varied to suit different data characteristics. This method effectively balances the bias-variance trade-off through parameters that control model complexity, and its capability for early stopping prevents overfitting. These advantages motivate us to choose Gradient Boosting as one of our classifiers.

Further, in the grid-search step, several essential parameters were considered, such as *max_features* and *max_leaf_nodes*, which help control the complexity of the individual trees and hence the entire model. This control over complexity is crucial for balancing the bias-variance trade-off, thus enhancing the model's generalization capabilities. The *n_iter_no_change* parameter, along with *validation_fraction* and *tol*, enables early stopping to prevent overfitting.

3.2.5 HistGradient Boosting

HistGradientBoosting also follows the core principle of boosting with decision trees. Compared with GradientBoosting, it is specially optimized for large-scale dataset by handling the continuous values through binning and supporting Nan values. The hyperparameter grids follows the same as GradientBoosting [5].

3.2.6 LGBMClassifier

LGBM(Light Gradient Boosting Machine) also follows the idea of boosting weaker learners sequentially but uses a leaf-wise growth strategy while the previous models apply a level-wise growth strategy [14]. And it is built under the LightGBM framework instead of scikit-learn. LightGBM often shows better computational efficiency in large-scale dataset.

3.2.7 RandomForest

Instead of using boosting technique, RandomForest applies bagging(or bootstrap aggregating)method, it builds numerous decision trees independently and averages their predictions, each tree is trained on a different sample of the data [4], therefore the independence between decision trees reduces the variance. Another advantage is the fast training time since every tree is trained independently. Furthermore, the interpretability and automatic feature selection characteristics motivate the use of this classifier.

3.2.8 XGBoost

XGBoost is a widely used classifier in lots of machine learning competitions. Compared to Gradient Boosting, while the foundational concept of sequential boosting of trees is shared, XGBoost introduces specific optimizations that include a novel tree learning algorithm that efficiently handles sparse data and improves computational speed. XGBoost also includes a regularization term in its objective function to reduce overfitting, which is not a standard component in traditional Gradient Boosting [7]. XGBoost's capability of handling sophisticated no-linear relationships, robustness to overfitting, speed and high generalizability make it a strong candidate for our prediction task.

3.2.9 Stacking Classifier

Stacking Classifier is an ensemble algorithm that combines the predictions from multiple models on the same dataset then using another model(called meta-learner or blender) to make a final prediction [21]. Stacking typically involves two or more layers of models. The first layer consists of a variety of base learners, and the second layer (or the final stage) is the meta-learner that optimally combines the predictions of the base models. The base models can be of different types, encouraging a diverse set of predictions. The key idea is that by combining different models, the strengths of each can be harnessed, and their weaknesses mitigated. Given the complex nature the different types of features extracted from diverse extractors and various models at our disposal, it is well suited to combine them all with stacking classifier.

3.3 OOP-ML Development

Upon finalizing the feature extractors and classifiers, we established an Object-Oriented Programming (OOP) framework to facilitate our machine learning tasks. The framework's structure is illustrated in the provided sketch, seen Figure 2.

The workflow is as follows: initially, cleaned data is processed through a series of notebooks for feature extraction, as detailed in Section 3.1. These extracted features are then stored separately for future use, indicated as D1, D2, etc., in the diagram. Each feature extractor is encapsulated within a class in FeatureExtractor.py, mirroring functions like fit() and fit_transform() akin to

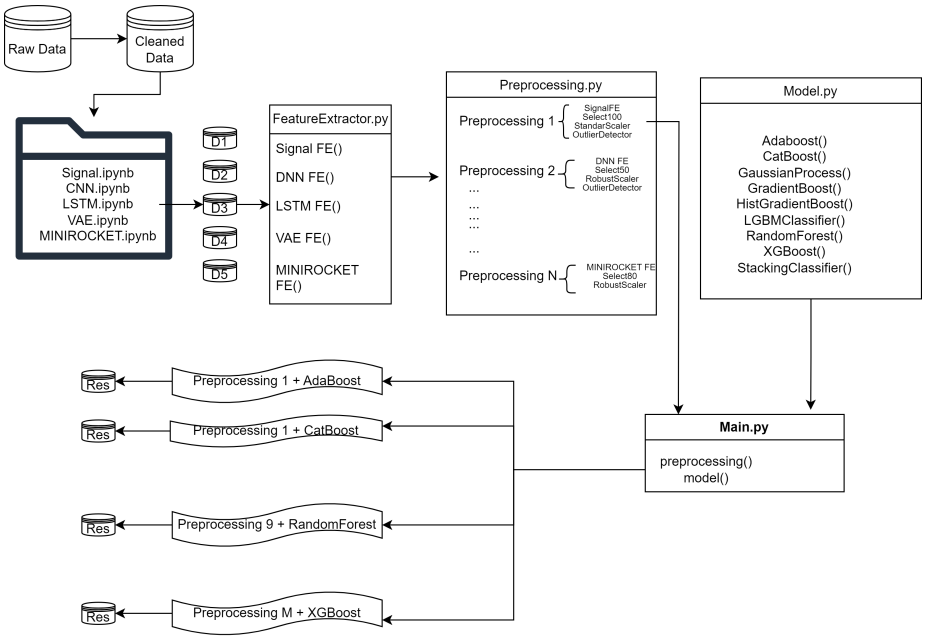

Figure 2: Structure of the code framework

scikit-learn estimators. These classes are integrated with various preprocessing methods such as SelectKBest, scalers, and outlier detector in preprocessing.py, setting the stage for defining pipeline objects. In model.py, we constructed each classifier described in Section 3.3, along with hyperparameter grids for fine-tuning. The main.py script manages the execution, performing grid search across all preprocessing-model combinations using the defined hyperparameter grids. This approach allows for meticulous record-keeping of each pipeline configuration, including settings and prediction results, for comprehensive final analysis and potential future implementations. This modular and independent structure enhances the framework's flexibility and reusability.

4 Results

This section presents the experimental results obtained using the feature extractors and classifiers discussed in Section 4. We meticulously fine-tuned the hyperparameters for each combination of feature extractor (including feature selector and scaler) and classifier. A comprehensive comparison encompassing all 45 possible grid search configurations (5 extractors $\times$ 9 classifiers) is detailed in the subsequent analysis, as seen Figure 3.

It is noteworthy that among all the feature extractors evaluated, MINIROCKET consistently outperformed others in terms of classification accuracy, closely followed by the CNN-based method. This superior performance of MINIROCKET

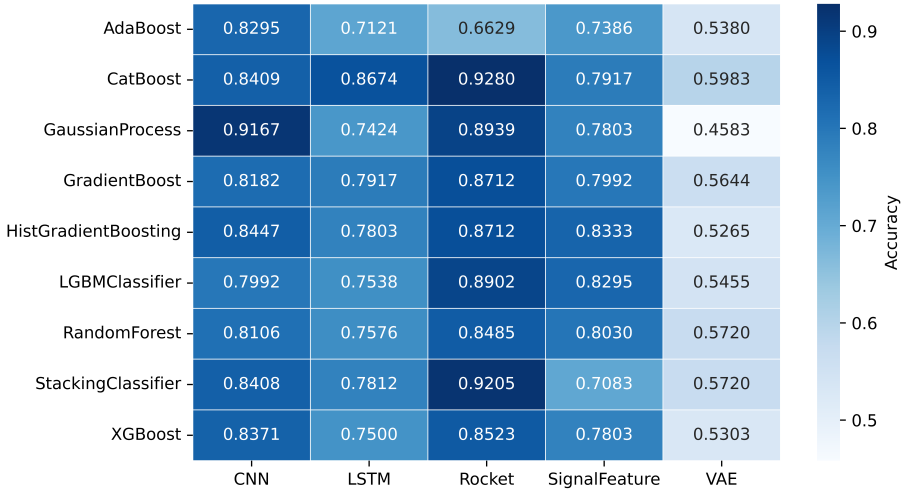

Figure 3: Comparison of all possible combination of feature extractor and classifier

can be attributed to its use of numerous random convolutional kernels, effectively capturing a wide array of characteristics in multivariate time series data. Additionally, its simplicity and feature generation approach potentially reduce the risk of overfitting, a challenge often encountered with more complex models like CNNs and LSTMs. On the other hand, the VAE method showed comparatively lower performance. This might be due to its inherent design focused more on data reconstruction and latent variable generation, rather than discriminative tasks such as classification. Moreover, the performance of VAEs can be highly dependent on the specific architecture and hyperparameter choices, suggesting a potential area for further optimization in our study.

The analysis of the results array reveals that the combination of the MINIROCKET feature extractor and the CatBoost classifier achieved the highest classification precision, recording an accuracy of 0.9280. The effectiveness of this pipeline is further illustrated in the Figure 4. The confusion matrix displays notably promising outcomes, with a significantly lower rate of misclassification, particularly in the lower diagonal. This aspect is crucial in our context, where the cost of airfoil failures is exorbitant. Consequently, a lower false negative rate is more critical than a lower false positive rate, making a cautious approach preferable.

5 Conclusion

In this study, we have developed a robust machine learning pipeline that accurately predicts the severity of structural damage in airfoils using aerodynamic pressure time-series data. The pipeline demonstrates high accuracy, with a notably low false negative rate, which significantly reduces the total cost of

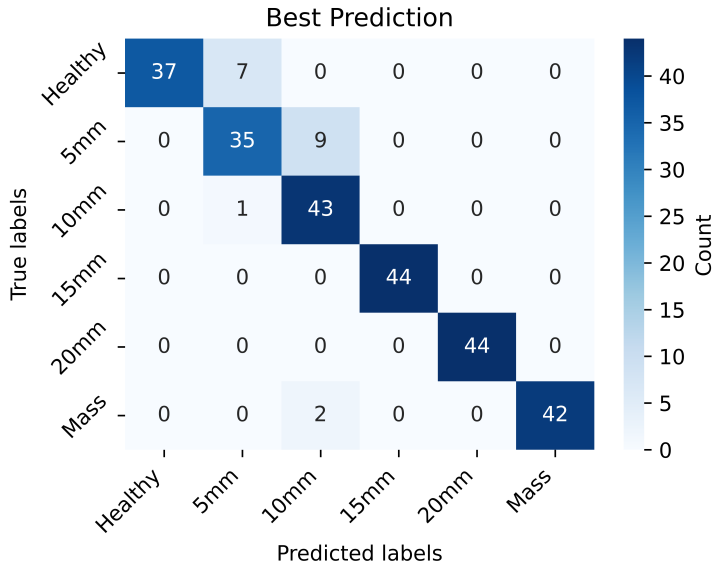

Figure 4: The prediction of the pipeline with highest accuracy: MINIROCKET+CatBoost

ownership.

The pipeline's design, rooted in object-oriented programming principles, offers high modularity and reusability. Each feature extractor and classifier is treated as an independent object, enhancing the ease of adaptation and reuse across various tasks. This design not only streamlines maintenance and facilitates future project extensions but also promotes effective team collaboration and knowledge exchange. Furthermore, the systematic tracking of optimal hyperparameters, training durations, and grid search outcomes simplifies both the replication of settings and the monitoring of performance.

A thorough comparison of various feature extractors and classifiers was conducted to evaluate their performance exhaustively. Our analysis concludes that the most effective pipeline combines the MINIROCKET feature extractor with the CatBoost classifier. This configuration ensures high prediction accuracy and a notably low false negative rate, exemplifying optimal performance in our study.

Looking ahead, a key aspect of our project's potential involves its readiness for edge computing. Owing to the swift training times and the integration of highly generalizable models, our pipeline is aptly tailored for edge computing applications, it is in line with RTDT's primary goals, paving the way for future deployment in edge scenarios. Such environments demand low latency and real-time processing capabilities, which our model is well-equipped to handle.

References

[1] Feature selection — scikit-learn 1.0.2 documentation. `https://scikit-learn.org/stable/modules/feature_selection.html`.

[2] I. Abdallah, G. Duthé, S. Gres, J. Deparday, R. Fäh, S. Barber, and E. Chatzi. Detecting damage via aerodynamic quantities on an aeroelastic structure. In *30th International Conference on Noise and Vibration Engineering (ISMA 2022) in conjunction with the 9th International Conference on Uncertainty in Structural Dynamics (USD 2022)*, Leuven, Belgium, September 12-14 2022. Conference presentation.

[3] Sarah Barber, Julien Deparday, Yuriy Marykovskiy, Eleni Chatzi, Imad Abdallah, Gregory Duthé, Michele Magno, Tommaso Polonelli, Raphael Fischer, and Hanna Müller. Development of a wireless, non-intrusive, mems-based pressure and acoustic measurement system for large-scale operating wind turbine blades, 01 2022.

[4] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] Jason Brownlee. Histogram-based gradient boosting ensembles in python. 2020.

[6] Sravan Challa, Akhilesh Kumar, and Vijay Semwal. A multibranch cnn-bilstm model for human activity recognition using wearable sensor data. *The Visual Computer*, 38, 08 2021.

[7] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.

[8] Chia Chen Ciang, Jung-Ryul Lee, and Hyung-Joon Bang. Structural health monitoring for a wind turbine system: a review of damage detection methods. *Measurement Science and Technology*, 19(12):122001, oct 2008.

[9] Angus Dempster and Daniel F. Schmidt. Minirocket: A very fast (almost) deterministic transform for time series classification. *arXiv preprint arXiv:2012.08791v2*, July 2021.

[10] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support. *CoRR*, abs/1810.11363, 2018.

[11] Philip Imanuel Franz. Shm in the context of fluid-structure interaction: Detecting structural damage under the aerodynamic loading of an airfoil, 2023.

[12] Yoav Freund and Robert E. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In Paul Vitányi, editor, *Computational Learning Theory*, pages 23–37, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[13] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[14] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, volume 30, pages 3146–3154, 2017.

[15] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[16] Richard Maclin and David W. Opitz. Popular ensemble methods: An empirical study. *CoRR*, abs/1106.0257, 2011.

[17] Leon Mishnaevsky. Root causes and mechanisms of failure of wind turbine blades: Overview. *Materials*, 15(9), 2022.

[18] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian processes for machine learning.* Adaptive computation and machine learning. MIT Press, 2006.

[19] B. Sirisha, Surakanti Naveena, Greeshma Palanki, and Pottipally Snehaa. Multivariate time series sensor feature forecasting using deep bidirectional lstm. *Procedia Computer Science*, 218:1374–1383, 2023. International Conference on Machine Learning and Data Engineering.

[20] U.S. Department of Energy. 2022 land based wind market report. `https://www.energy.gov/sites/default/files/2022-08/land_based_wind_market_report_2202.pdf`, August 2022.

[21] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.