ISyE 6748 - Summer 2024
Project

Project Title: Wind Turbine Fault Detection

Author: Nehemiah Solis

CONTENTS

I. INTRODUCTION

Wind energy is making waves in the energy space, continuously growing its share of the pie among the methods of energy generation due to its clean and relatively inexpensive operations. It has staked its place as an essential alternative energy source to fossil fuels, generating 430 billion kWh in 2022 and making up about ten percent of the total utility-scale electricity generation in the United States. Moreover, according to the Global Wind Report, the trend continues, with 117 GW of new wind power installations, 132 countries signing up to triple renewable energy capacity by 2030,[3] and governments introducing tax incentives and support to expand capacity[5]. Players in the space aim to reduce those operation costs further, efficiently produce wind energy, and manage maintenance costs, especially with increased competition on the horizon.

To better understand the project, it is worth providing a brief background on the different components of the wind turbine and how they provide signals for data analysis. The component sends signals to the supervisory control and data acquisition (SCADA) system, which is necessary to monitor and analyze the wind turbine's functions. Most of the issues found in SCADA turbine data are related to the operation of these parts.

First are the *blades*, which most people think of when they hear "wind turbine." The blades can twist to improve aerodynamic efficiency, with the velocity at the tips much higher than at the blade's root. When optimized, they can operate in freezing and icing conditions to prevent stalling and increased blade vibration levels or over-speed events after the sudden shedding of ice. In addition, turbine control algorithm optimizations ensure that temperature-sensitive components can function correctly. The *blade pitch* includes the hydraulics and controls that help manage the blade angle in abnormal environmental conditions.

Another essential component is the *gearbox*, which transmits power from the blades to the generator and consists of the main bearing and shaft. The *generator*, in turn, is used to increase voltage. The *nacelle*, a critical component, detects wind speed, wind direction, and ambient temperature. Lastly, the *yaw gear and motor* are pivotal in orienting the turbine into the wind.

With all these moving parts and components, it is no wonder that Operation and Maintenance costs contribute to a sizeable (sometimes about 20%) portion of a turbine's total expenditure[16]. The current best practices for O&M are preventative maintenance, predetermined part replacement, and design modification. For example, on a wind farm, depending on the contract between the owner and operator and the manufacturer (such as General Electric, Siemens Gamesa, and Vestas), the amount of preventative maintenance that each party is responsible for is predetermined by hours and allotted every year. In addition to the predetermined maintenance hours, these plans are often interrupted by bad weather, early breakdowns, and general machine degradation, adding more costs to the owners (or manufacturers if they signed a bad contract).

As a new entrant to the wind energy industry, this project will act as my attempt to contribute to the burgeoning field, especially concerning various fault prevention techniques.

II. LITERATURE REVIEW

There is a wealth of literature on turbine fault detection and health monitoring, so part of the challenge of this project is narrowing the focus while still attempting to contribute novelty to the existing corpus. Methods for detecting faults and predicting potential failures are richly researched and opined, which, as an aside, can make it challenging to pick a lane to attack for a project. Despite much research, most production data is proprietary, so many studies utilize the same public or private data sources the authors cannot disclose[4,7]. Other studies rely on synthetic data that engineers simulated in a laboratory. Fortunately, Glück et al. published a pre-print[7] this year of a massive dataset that seeks to establish a massive, balanced dataset for anomaly detection (though one can use the dataset for more than just anomaly detection tasks). I will go over more details of their data collection in the next section.

Each paper utilizes unique data preprocessing steps to prepare the data for analysis. They must deal with imbalanced data, improper data collection, and incorrect fault classification. Velandia-Cardenas et al.[1] addresses imbalanced data using random oversampling, generating synthetic data from the minority class and a data augmentation method to increase the number of samples available without increasing the overall dataset size. Chatterjee and Byun[2] employ the ASMOTE-ENN method, an adaptive technique that I was previously and wholly unfamiliar with that runs an adaptive neighbor selection algorithm to spot candidates for resampling rather than relying on random samples. Leahy et al. developed a library[14] providing a framework for grouping and labeling alarms in SCADA datasets. By accurately labeling the data, one can more accurately identify fault periods, improve classification performance, and even generate future alerts. Although Leahy's library has not been updated in four years, I made my first attempt at an open-source contribution by updating his requirements Python file on a forked branch. Glück et al., on the other hand, focused on establishing a balanced dataset, thus facilitating better generalization and machine learning applications.

Previous research has successfully classified faults using SCADA[10] and vibration data[12]; classification and deep learning models[11,8] to detect faults for predicting failures; generative models to rectify imbalances in the data or false fault classifications[1]; anomaly detection to detect a fault during normal operations[7]; and even old school engineering models like the Petri net model of a bipartite graph with nodes signifying places and transitions[9]. Given this vast array of methods, engineering, and implementations, I decided to use a technique not present in these papers, namely high-dimensional data analysis, such as B-splines, functional principal components, and logistic group lasso. Using the coefficients obtained from the dimension reduction, I could more efficiently utilize traditional classification models, like RandomForeset. I also expanded upon Glück et al.'s work on data collection, aiming to provide an open-source contribution to the field.

III. DATA

3.1 Data Source

For this project, I used the Glück et al. data collection[6], which can be found following the citation. Similarly, the datasets compiled by Soontronchai[15] and Ogaili et al.[12] can be located with their citations.

3.2 Description

When performing my analysis, one of the first things I had to do was find the data I would mainly use. As mentioned in the Literature Review, finding valuable data in the wind energy space can be tricky, so I was pleased to discover a few examples of good wind turbine data. I was especially happy with "CARE to Compare" 's attempt to gather and aggregate disparate sources into one location. Their massive dataset spans 95 sub-datasets, including three wind farms and 36 individual turbines across 89 years. I also found an excellent Kaggle dataset[15] already used in some research papers[10,2]. Soontronchai's dataset was helpful for initial data exploration. However, the more I explored the data complied by Glück et al., the more I found it fulfilled all the requirements I needed for my analysis.

Similarly, I made an initial attempt at exploring a vibration dataset that was collected in a laboratory to perform an analysis of the vibration data. While a SCADA system generates Glück et al.'s and Soontronchai's data, the vibration dataset represented an interesting second direction one could follow for fault detection. However, I still felt that the Glück et al. dataset provided more than enough sensor information and that further analysis with vibration data would serve as future work. Another advantage of focusing on the data of Glück et al. is that in the real world, it is more than likely that any data analyst, scientist, or engineer will be working on wind farms with dozens of turbines per wind farm.

In my initial report, I focused on building my models using the wind farm anonymized as "Wind Farm A" based on an onshore wind farm in Portugal from the EDP-open data platform. Everything in the datasets was anonymized, which is attractive due to the previously discussed challenge of how protective the owners of wind data are. The complete corpus is balanced, with eleven datasets containing anomalous data and the others operating under normal conditions. Every dataset contains one event corresponding to normal operations or a fault, such as a gearbox, transformer, or generator bearing failure. Each row in the data represents a ten-minute interval, which is standard practice in the industry.

Every interval of time corresponds to a "turbine status," of which there are six:

Table 1—Operational states of the wind turbine

Class	Description
0	Normal operation – the turbine is in normal power production mode
1	Derated operation – derated power generation with a power restriction
2	Idling - The asset is idling and waits to operate again (normal operations)
3	Service - Asset is in service mode/service team is at the site
4	Downtime - Asset is down due to a fault or other reasons
5	Other - Other operational states

The data is high-dimensional, with Wind Farm A containing 86 features, 54 of which are sensors. Wind farms B and C contain more features, 257 and 957, respectively. The sensor measurements are usually ten-minute averages, but some variables include the ten-minute maximum, minimum, and standard deviation measurements.

IV. METHODOLOGY

4.1 Data Pre-processing

With Glück et al.'s data, I built upon their data collection process, developing a novel data processing step to streamline collecting and performing basic data preparation techniques for more accessible analysis. For example, the datasets are freely provided by Glück et al., but they come in zip files for each wind farm, with individual datasets within each wind farm along with the event info and feature descriptions. While I appreciate this data collection and the tidiness of its assembly, it was cumbersome to import; there were millions of rows of data and thousands of columns. In fact, one of the most underrated but pleasing aspects I found in working on this report was optimizing the data preprocessing step. Traditionally, most data analysts will default to Python and tabular data manipulation libraries we are all familiar with, including `pandas` and `numpy`. I am no different. Using `pandas`, I wrote a script that imported the 95 CSV files, totaling 5.125 GB and 5,243,398 million rows across all three farms, into a nested dictionary of DataFrames that any analyst could easily query if they wanted to work on a subset of the data. The script also performed basic preprocessing tasks, such as converting the time features to `datetime` objects, replacing lost symbols with their correct special characters, and adding any necessary id columns.

Unfortunately, if I wanted to import the entirety of the data into a notebook, the process would take up to four minutes. That might not seem like a lot, but when you are actively testing and refactoring your code, it can be cumbersome to wait for data loading every time. Fortunately, there exists a library that performs the same tasks as `pandas`: `polars`. I pivoted from `pandas` to `polars` to optimize the importing process of this relatively massive dataset. Out of the box, not only did `polars` clean up the code (you can chain functions and queries), but it also decreased the time it took to load my data into a local notebook to about 40 seconds! I did not stop there; I was hooked and wrote in the option to allow users to take advantage of `polars` LazyFrames, which is where `polars` represents a

Lazy computation graph against a DataFrame and allows for whole-query optimization and parallelism. Loading all the data into a dictionary of Lazyframes took less than a second. From there, the user can call the `collect()` method to materialize the LazyFrame into a DataFrame in their notebook—four minutes to milliseconds.

After transforming the data for analysis, I then added a new target feature called `fault_status`, where the simple formula can determine the fault status:

$$\text{Fault} = \begin{cases} 0 & \text{if Status ID} = 0 \text{ or } 2 \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

These faults indicate an anomaly event, and by combining the other three status IDs into one fault status, I can analyze a traditional classification problem rather than a more unbalanced multivariate one.

I also ran my report with two separate datasets: one without cleaning and one with cleaning. The cleaned dataset removed negative wind speeds and any generated power that was below zero.

4.2 Data Exploration

While I spent most of my time collecting, processing, and transforming the data for analysis, I also dedicated some time to performing exploratory data analysis. During this process, I worked with various data sets. After manipulating their features and creating visual representations, I found that focusing on exploring and expanding upon the Glück et al. data would be more productive. As a result, I have created plots for each turbine in wind farm A, which are displayed below.

The power curves for each turbine roughly correspond to the same non-linear pattern, indicating that the turbines probably encounter similar ambient conditions, especially concerning wind speeds, and have similar power ratings. I interpreted this as a positive for future modeling tasks since the underlying assumptions for each turbine would be identical, and I would not need to worry about the nuances involved in inter-turbine analysis. If I ever include the turbines from other wind farms, more feature engineering would be required to build a one-size-fits-all model.

Power curves are potent tools for monitoring turbine performance. In the following figure, I have plotted the power curve for Asset 10 with its normal status

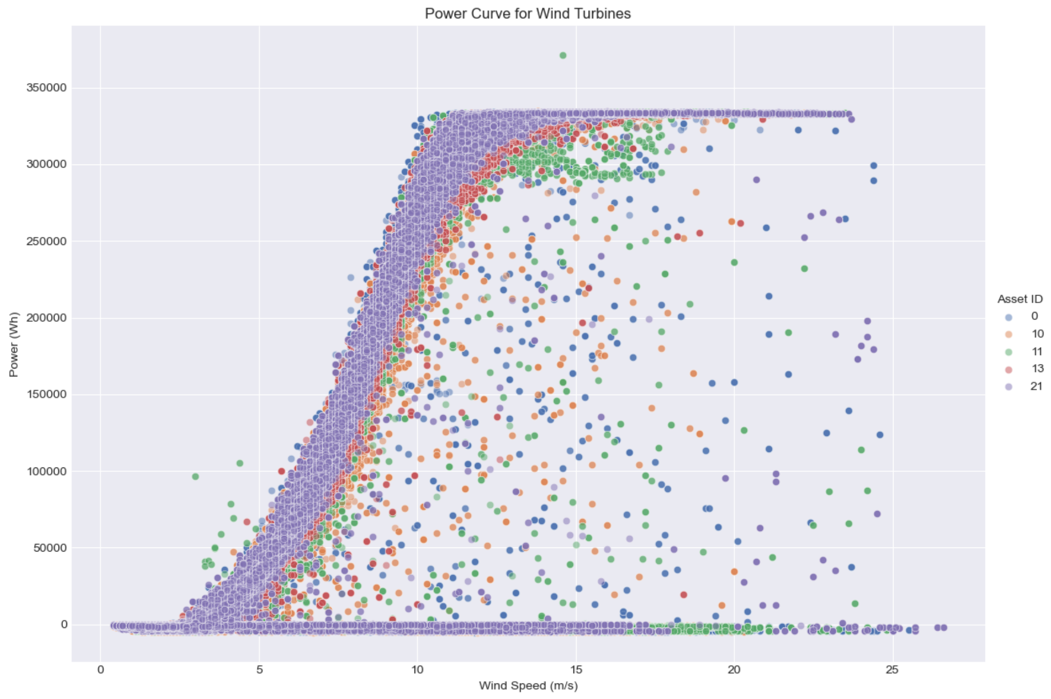


Figure 1—Power Curve for Wind Farm A: All Turbines

points and those where an abnormal event occurred.

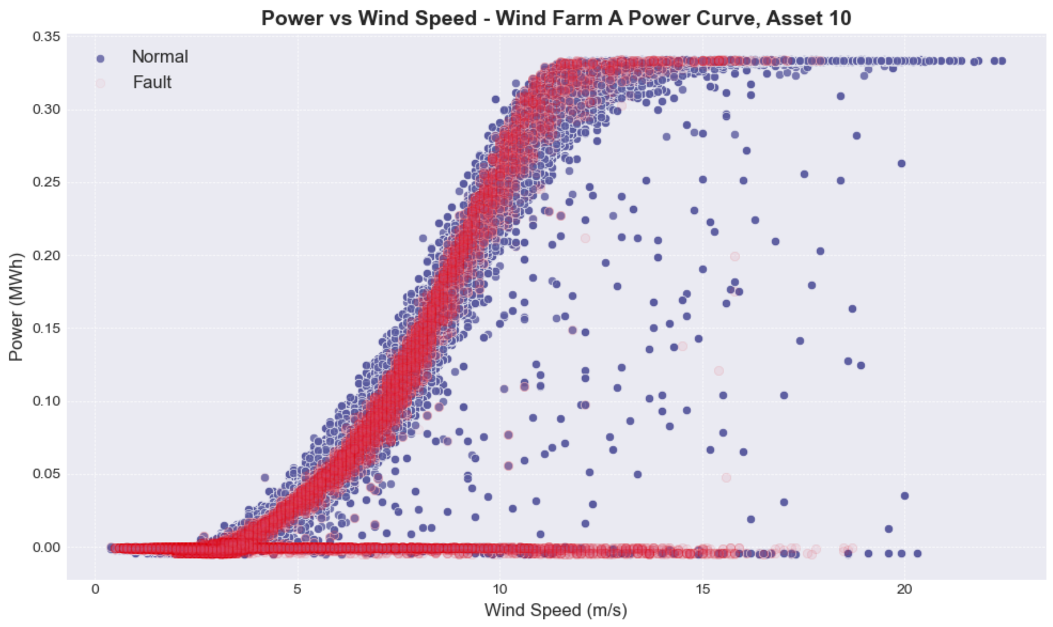


Figure 2—Power Curve for Asset 10: Normal Operations vs. Anomaly Events

Aside from the greater prevalence of fault data where the power generation is zero, it is surprising that the distribution of abnormal and normal data isn't as clear-cut as expected. Like an oblivious driver speeding down the highway on bald tires, wind turbines can generate expected power despite abnormal events. We can see that even the "outliers" are primarily no-fault data; the fault data overlays more or less neatly alongside the normal generation, presenting an interesting problem where we'll need more than power curves to detect faults.

4.3 Model Development

4.3.1 *Dimensionality Reduction and Feature Engineering*

As described in the literature review, previous studies often used traditional machine learning and deep learning. One common theme in my review of different approaches was using novel data preprocessing or dimension reduction techniques. I want to add to the corpus of methods to implement novel solutions for detecting faults and utilizing high-dimensional data analytic methods. Admittedly, I was inspired by the Georgia Institute of Technology class ISYE 8803-High Dimensional Data Analytics, and it has exposed me to methods that seem fit for sensors and machine data. My SCADA data is extremely high-dimensional, with each wind farm tallying hundreds of features, which naturally leads to trouble in model development, as the high number of features can lead to unacceptable computation times, less-than-stellar model results, and cumbersome data processing.

I used B-spline basis functions and functional principal components to generate coefficients and features that I could use to reduce the dimensionality of the data. In the next figure, I've included an example of the power of dimension reduction with B-splines, using the example of wind speed and power generation.

As we can see, the B-spline, in this case, faithfully predicted the correct fit to the underlying high-dimensional data.

For the dimension reduction with group lasso, I wrote a Python script that grouped the sensors and environmental variables so that the group lasso regularization could discover functional patterns based on the features. These groups were hard coded based on wind farm A, which limits its usefulness, especially for other wind farms or datasets with different features.

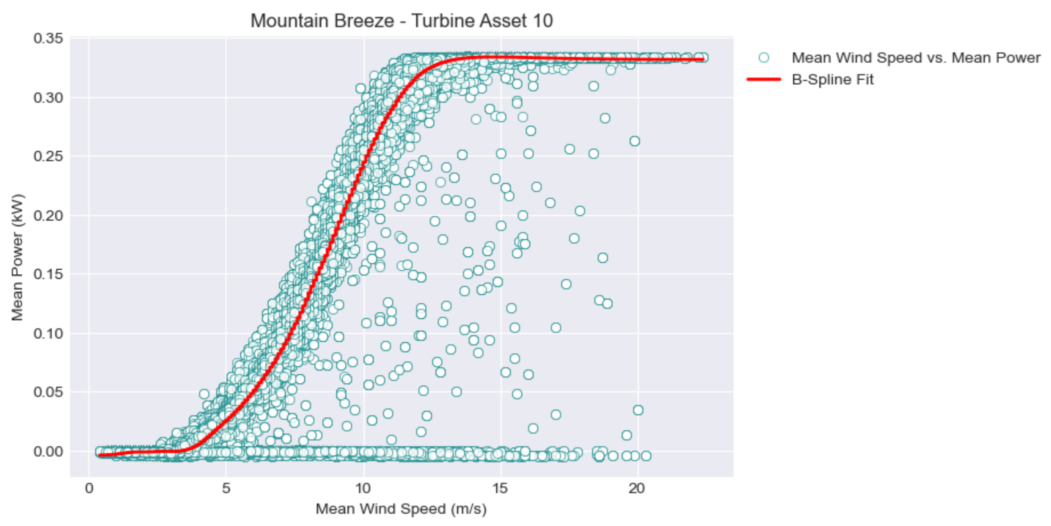


Figure 3—Power Curve for Asset 10: B-Spline Predictive Fit

4.3.2 *Classification Machine Learning*

I began with a small subset of the data and focused on Turbine Asset 10. Using the simple B-spline method to extract coefficients, I reduced the dimensionality of the data. I used the B-spline coefficients to apply a random forest classification model to the data. In my initial test, I achieved an accuracy of 0.77 with a precision of 1 and a recall of .77. However, this dataset was highly unbalanced. Still, I thought it was promising, proving that spline dimensionality reduction could be worth exploring. The most important features were wind speed, ambient temperature, wind direction, estimated wind speed, and relative wind direction.

Next, I took it up a notch and combined all of Wind Farm A's data into one dataset with five turbines, 81 training features (including 54 sensors), and 1.2 million rows of 10-minute data (850,000 when cleaned). I split the data into train and test sets and rolled the status types up into 1 or 0, with 1 being a 10 min interval where there was a fault code associated with the turbine state 1, 3, 4, or 5 and 0 being a 10 min interval where the turbine state was running or in standby (0 and 2).

I built a group lasso model to explore the high-dimensional analytics space further.

Group Lasso—The group lasso, generally reserved for regression tasks, is excellent when features are naturally partitioned into grouped variables. For example, in the wind turbine, many sensors are located in the same physical space and experience similar environmental and operating conditions, such as wind, ambient temperature, and vibration. At the same time, there are many of these sensors, and group lasso (like any regularization method) ameliorates the curse of high dimensionality, identifying the most relevant features, reducing dimensionality, and improving model interpretability.

Since the sensors will generally send signals grounded in the same environment (the turbine and the ground the turbine stands on don't change), there is a high likelihood of multi-co-linearity between many sensors. By regularizing on groups of sensors (or, in one experiment, turbines) instead of individual features, I can obtain robust parameters, thus helping to avoid over-fitting. In my first experiment, I did not perform any preprocessing data dimension reduction and used the five turbines as my lasso groups. I wanted to see if any features from the same turbine are more related, thus discovering turbine-specific patterns or differences. In my second experiment, I categorized my features into the following groups: ambient conditions, controller temperatures, component temperatures, electrical measurements, power measurements, and operational metrics.

As a mathematical aside, for my specific use case, it is advantageous to understand how the groups of features that all interact within a system lead to faults and how we can identify those patterns. To perform the group lasso, I used the `group_lasso` Python library. This method is generally reserved for regression tasks, so I extended the traditional group lasso's sparse-group penalty into a logistic regression model using the same library's experimental logistical group lasso class. Below is a quick mathematical background described by Simon et al.[13]. To convert the problem to a logistic regression task, we minimize:

$$\ell(\beta) + (1-\alpha)\lambda \sum_{l=1}^{m} \sqrt{p_l}\|\beta^{(l)}\|_2 + \alpha\lambda\|\beta\|_1,$$

where $\ell(\beta) = -1/n \log(L(\beta))$. For logistic regression, we have y, the fault status of the data point, and X, an n by p co-variate matrix divided into m groups, where I used five (turbines) or six (types of sensors and features)-sized groups. According to Simon et al., the spars-group lasso is

$$\hat{\beta} = \arg\min_{\beta} \frac{1}{n} \left[\log \left(\sum_{i \in D} \left(\sum_{j \in K_i} \exp(x_j^\top \beta) - x_i^\top \beta \right) \right) \right] + (1-\alpha)\lambda \sum_{l=1}^{m} \sqrt{p_l} \| \beta^{(l)} \|_2 + \alpha\lambda \| \beta \|_1$$

In the above equation, D denotes the set of failures and, as long as the responses in the set of j indices are still at risk of failing at time i, K_i is the set of indices.

Implementation—Continuing with the model implementation and experiments, I performed the same logistic regression group lasso on the feature groups. I used the default parameters in both experiments, so my next step was cross-validation for hyperparameter tuning.

Unfortunately, I quickly learned that hyperparameter tuning was a computationally arduous task. I started with the `GridSearCV` but had to stop the operation after an hour of training. I switched to `RandomizedSearchCV`, allegedly the more efficient of the two methods. Alas, this also did not help much. While I've worked on models with longer training times, I knew I did not have to settle.

To improve my model performance, I implemented functional principal components to reduce the dimensions of my features. I used those components as my new groups for the logistic regression group lasso model. I ran another two experiments, once with the default parameters again and once with the cross-validation. The culmination of my experiments can be seen in the algorithm on the following page.

And here are my first three principle component results.

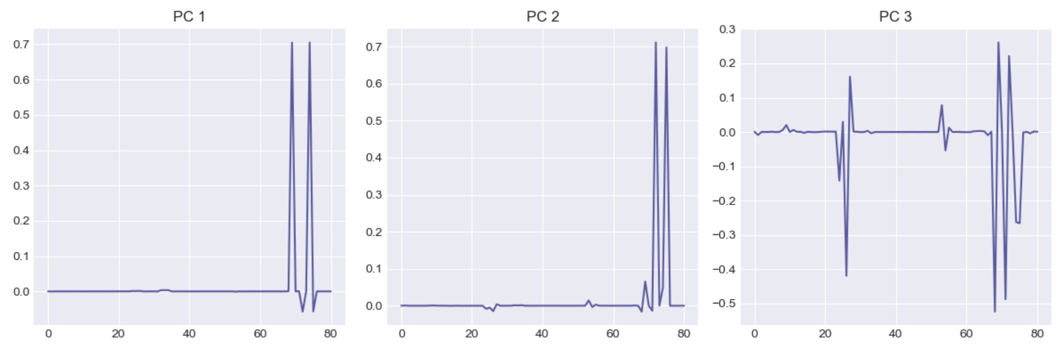


Figure 4—PCA: Feature Importance vs Feature Index

Algorithm 1 Principal Component Analysis with Logistic Group Lasso for Wind Turbine Fault Detection

```
1: **Input:** Data matrix $X \in \mathbb{R}^{n \times M}$, target vector $y \in \mathbb{R}^n$
2: **Output:** Trained model and predictions
3: $X_c \leftarrow X - \bar{X}$                                   ▷ Center the data
4: $\Sigma \leftarrow \frac{1}{n-1} X_c^T X_c$                    ▷ Compute covariance matrix
5: $(\lambda, V) \leftarrow \text{eig}(\Sigma)$                    ▷ Compute eigenvalues and eigenvectors
6: Sort $\lambda$ and $V$ in descending order
7: $k \leftarrow$ number of principal components to retain
8: $P \leftarrow V[:, :k]^T X_c$                                   ▷ Project data onto principal components
9: var_ratio $\leftarrow \lambda_{1:k} / \sum_{i=1}^{M} \lambda_i$  ▷ Explained variance ratio
10: Split $P$ and $y$ into training and test sets
11: Scale $P_{\text{train}}$ and $P_{\text{test}}$
12: Initialize groups $g \leftarrow [0, \dots, 0]$ of length $k$
13: cum_var $\leftarrow$ cumsum(var_ratio)
14: **for** $i = 1$ to $k$ **do**
15:     **if** cum_var$[i] > 0.99$ **then**
16:         $g[i:] \leftarrow [1, \dots, 1]$
17:         **break**
18:     **end if**
19: **end for**
20: Compute lasso path to determine range of $\lambda$ values
21: $\lambda \leftarrow$ lasso_path$(X_{\text{train\_scaled}}, y_{\text{train}})$
22: $n_\lambda \leftarrow 10$                                       ▷ Or another suitable number
23: $\lambda \leftarrow \text{logspace}(\log_{10}(\lambda.\text{max}()), \log_{10}(\lambda.\text{min}()), n_\lambda)$
24: Initialize LogisticGroupLasso model with $g$, group$_\lambda = 0$, and $l1_{\text{reg}} = 0$
25: Initialize cross-validation search with logistic group lasso model and parameter distribution for group$_\lambda$
26: Fit cross-validation search on $P_{\text{train}}$ and $y_{\text{train}}$
27: Fit best logistic group lasso model on $P_{\text{train}}$ and $y_{\text{train}}$
28: Predict on $P_{\text{test}}$ to get $\hat{y}_{\text{test}}$
29: Evaluate model performance
```

V. RESULTS AND DISCUSSION

The following are my results for the different iterations of the logistic group lasso implementation, with the dirty and clean data.

Turbine Groups—

Table 2—Performance Metrics for Clean and Unfiltered Data

	Clean Data			Unfiltered Data		
Metric	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Class 0	0.87	1.00	0.93	0.86	0.84	0.85
Class 1	0.33	0.01	0.01	0.51	0.55	0.53
Accuracy		0.87			0.78	
Macro Avg	0.60	0.50	0.47	0.68	0.70	0.69
Weighted Avg	0.80	0.87	0.81	0.78	0.78	0.78

Here, we can see that if we want to predict normal operating conditions, cleaning the data is the way to go, correctly identifying every instance of non-fault data. However, it struggles significantly to classify fault data. Alternatively, leaving the data as-is results in better classification of fault data.

Feature Groups—

Table 3—Performance Metrics for Clean and Unfiltered Data

	Clean Data			Unfiltered Data		
Metric	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Class 0	0.87	1.00	0.93	0.86	0.85	0.86
Class 1	0.43	0.01	0.03	0.51	0.54	0.52
Accuracy		0.87			0.78	
Macro Avg	0.65	0.51	0.48	0.69	0.69	0.69
Weighted Avg	0.81	0.87	0.81	0.78	0.78	0.78

A similar situation was encountered here. When using the default parameters and all features (albeit grouped), the unfiltered data outperformed the model using the clean data. One notable distinction is that the precision of the clean data for the feature groups was higher than the precision for the turbine groups, suggesting that there could be some advantages to using the feature groups.

***Functional PCA Groups*—**

Table 4—Performance Metrics for Clean and Unfiltered Data

	Clean Data			Unfiltered Data		
Metric	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Class 0	0.92	1.00	0.96	0.84	0.94	0.89
Class 1	0.23	0.00	0.00	0.73	0.48	0.58
Accuracy		0.92			0.83	
Macro Avg	0.58	0.50	0.48	0.79	0.71	0.73
Weighted Avg	0.86	0.92	0.88	0.82	0.83	0.81

Here, we can see that using functional PCA groups enhances the model's performance, particularly in identifying faulty instances in the unfiltered data scenario. It also improves the computational performance of the model training, so I would always recommend using functional PCA groups.

***Functional PCA Groups with Cross-Validation*—**

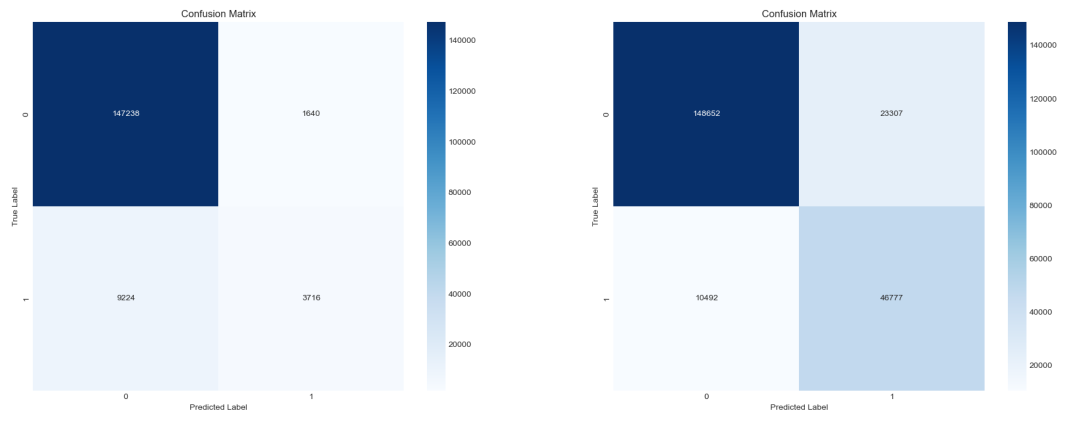
For the penultimate model, I have the results for the logistic regression group lasso model with FPCA groups and cross-validated, where the best λ= 7.872e-05.

Table 5—Performance Metrics for Clean and Unfiltered Data

	Clean Data			Unfiltered Data		
Metric	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Class 0	0.94	0.99	0.96	0.93	0.86	0.90
Class 1	0.69	0.29	0.41	0.67	0.82	0.73
Accuracy		0.93			0.85	
Macro Avg	0.82	0.64	0.69	0.80	0.84	0.82
Weighted Avg	0.92	0.93	0.92	0.87	0.85	0.86

The latest results are very promising! The model performed remarkably well across all metrics. When tested on unfiltered data, It accurately detected fault and anomaly data with a precision of 0.67 and correctly identified anomalies with a recall of 0.82. However, the performance with clean data, particularly regarding recall, remained subpar but showed improvement.

These results indicate that although the model achieves higher overall accuracy



(a) Clean Data ***(b)*** Unfiltered Data

Figure 5—Confusion Matrices for Clean and Unfiltered Data

with clean data, it demonstrates more reliable fault detection when tested with unfiltered data. In this instance, it is crucial to emphasize that model performance is not everything. One of the reasons for the unfiltered data's superior performance is that many cases of the faulted data were due to bad readings of wind speed and power generation. Features where this is the case make identifying anomalies much easier. If that is what we expect to see in the field, then that is fine, but if that is misleading (for instance, the bad data is inconsistent with the conditions in the field), then the model is misleading and may perform worse in production. Additionally, cleaning the data reduced the balance of the two classes, so future work could involve cleaning the data and then implementing resampling methods from the literature review to assist in the imbalance problem.

VI. FUTURE WORK

Working on this project has been a gratifying experience, and I am eager to continue this research beyond these initial findings. The literature I reviewed before this project was informative and inspiring, helping me identify areas for further exploration and improvement. For example, cleaning the data should be a positive step in the analysis, but its poor performance compared to the unfiltered data indicates that I need to implement some advanced resampling methods as an extra step before the modeling process. Additionally, I only used a logistic regression model for my advanced technique. I imagine there is much room for experimentation with deep-learning models.

I am excited to share my code with the public and invite anyone who stumbles across it to reach out for collaboration or to use it for their education. Any insights and contributions are valuable, and I look forward to the potential collaborations that may arise from this open sharing of knowledge.

VII. REFERENCES

[1] Cárdenas, C. Velandia, Segui, Y., and Pozo, F. (2021). “Wind Turbine Fault Detection Using Highly Imbalanced Real SCADA Data”. In: *Energies* 14, p. 1728. DOI: 10.3390/en14061728. URL: https://doi.org/10.3390/en14061728.

[2] Chatterjee, S. and Byun, Y.-C. (2023). “Highly imbalanced fault classification of wind turbines using data resampling and hybrid ensemble method approach”. In: *Engineering Applications of Artificial Intelligence* 126, p. 107104. DOI: 10.1016/j.engappai.2023.107104. URL: https://doi.org/10.1016/j.engappai.2023.107104.

[3] Council, Global Wind Energy (2024). *Global Wind Report 2024*. Joyce Lee and Feng Zhao, Lead Authors. URL: https://gwec.net/wp-content/uploads/2024/05/GWR-2024_digital-version_final-2.pdf.

[4] Effenberger, N. and Ludwig, N. (2022). “A collection and categorization of open-source wind and wind power datasets”. In: *Wind Energy* 25.10, pp. 1659–1683. DOI: 10.1002/we.2766. URL: https://doi.org/10.1002/we.2766.

[5] Energy, U. S. Department of and Ocean Energy Management, Bureau of (2024). “Advancing the Growth of the U.S. Offshore Wind Industry: Federal Funding and Incentives”. In: *Review of Advancing the Growth of the U.S. Offshore Wind Industry: Federal Funding and Incentives*. URL: https://www.energy.gov/sites/default/files/2024-04/DOE-BOEM-Fed-Offshore-wind-v3_w150.updated.pdf.

[6] Gück, C. and Roelofs, C. M. A. (2024). *Wind Turbine SCADA Data For Early Fault Detection (v1.0) [Data set]*. Zenodo. DOI: 10.5281/zenodo.10958775. URL: https://doi.org/10.5281/zenodo.10958775.

[7] Gück, C., Roelofs, C. M. A., and Faulstich, S. (2024). *CARE to Compare: A real-world dataset for anomaly detection in wind turbine data*. Version 1. arXiv:2404.10320; Version 1. arXiv: 2404.10320 [cs.LG]. URL: https://doi.org/10.48550/arXiv.2404.10320.

[8] Jia, X., Jin, C., Buzza, M., Wang, W., and Lee, J. (2016). “Wind turbine performance degradation assessment based on a novel similarity metric for machine performance curves”. In: *Renewable Energy* 99, pp. 1191–1201. DOI: 10.1016/j.renene.2016.08.018. URL: https://doi.org/10.1016/j.renene.2016.08.018.

[9] Le, B. and Andrews, J. (n.d.). *Modelling wind turbine degradation and maintenance*. n.d. DOI: 10.1002/we.1851. URL: https://doi.org/10.1002/we.1851.

[10] Leahy, K., Gallagher, C., O'Donovan, P., Bruton, K., and O'Sullivan, D. (2018). "A Robust Prescriptive Framework and Performance Metric for Diagnosing and Predicting Wind Turbine Faults Based on SCADA and Alarms Data with Case Study". In: *Energies* 11, p. 1738. DOI: 10.3390/en11071738. URL: https://doi.org/10.3390/en11071738.

[11] Lee, C.-Y. and Maceren, E. D. C. (2024). "Wind energy system fault classification and detection using deep convolutional neural network and particle swarm optimization-extreme gradient boosting". In: *IET Energy Systems Integration* 1–19.n/a. n.d. DOI: 10.1049/esi2.12144. URL: https://doi.org/10.1049/esi2.12144.

[12] Ogaili, A. A. F., Jaber, A. Abdulhady, and Hamzah, M. N. (2023). "Wind Turbine Blades Fault Diagnosis based on Vibration Dataset Analysis". Version 4. In: *Mendeley Data* 4. DOI: 10.17632/5d7vbdp8f7.4. URL: https://doi.org/10.17632/5d7vbdp8f7.4.

[13] Simon, Noah, Friedman, Jerome, Hastie, Trevor, and Tibshirani, Rob (2013). "A Sparse-Group Lasso". In: *Journal of Computational and Graphical Statistics* 22.2, pp. 231–245. DOI: 10.1080/10618600.2012.681250.

[14] Solis, Nehemiah (2024). *WTPHM*. GitHub repository, Forked from https://github.com/lkev/wtphm. URL: https://github.com/NehemiahSolis/wtphm/tree/master.

[15] Soontronchai, Wasurat (July 2023). *IIoT Data of Wind Turbine*. Version 1.0. Retrieved from https://www.kaggle.com/datasets/wasuratme96/iiot-data-of-wind-turbine/data?select=scada_data.csv.

[16] Tautz-Weinert, J. and Watson, S. J. (2016). "Using SCADA data for wind turbine condition monitoring – A review". In: *IET Renewable Power Generation* 11.4, pp. 382–394. DOI: 10.1049/iet-rpg.2016.0248. URL: https://doi.org/10.1049/iet-rpg.2016.0248.