Predictive Maintenance of Wind Generators based on AI Techniques

by

Emin Elmar oglu Mammadov

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2019

Abstract

As global warming is slowly becoming a dangerous reality, governments and private institutions are introducing policies to minimize it. Those policies have led to the development and deployment of Renewable Energy Sources (RESs), which introduces new challenges, among which the minimization of downtime and Levelised Cost of Energy (LCOE) by optimizing maintenance strategy where early detection of incipient faults is of significant intent. Hence, this is the focus of this thesis.

While there are several maintenance approaches, predictive maintenance can utilize SCADA readings from large scale power plants to detect early signs of failures, which can be characterized by abnormal patterns in the measurements. There exists several approaches to detect these patterns such as model-based or hybrid techniques, but these require the detailed knowledge of the analyzed system. As SCADA system collects large amounts of data, machine learning techniques can be used to detect the underlying failure patterns and notify customers of the abnormal behaviour.

In this work, a novel framework based on machine learning techniques for fault prediction of wind farm generators is developed for an actual customer. The proposed fault prognosis methodology addresses data limitation such as class imbalance and missing data, performs statistical tests on time series to test for its stationarity, selects the features with the most predictive power, and applies machine learning models to predict a fault with 1 hour horizon. The proposed techniques are tested and validated using historical data for a wind farm in Summerside, Prince Edward Island (PEI), Canada, and models are evaluated based on appropriate evaluation metrics. The results demonstrate the ability of the proposed methodology to predict wind generator failures, and the viability of the proposed methodology for optimizing preventive maintenance strategies.

Acknowledgements

First and foremost, I would like to express my sincere graditude to my supervisor, Dr. Claudio A. Cañizares for the invaluable support and guidance throughout my MASc studies. It has been a privilege to complete my degree under his supervision, and his professionalism and dedication will always be an example for me throughout my professional life. Finally, I wish to acknowledge the support of the Natural Sciences and Engineering Research Council (NSERC) for providing partial funding necessary to complete my degree.

I am particularly thankful to Dr. Mostafa Farrokhabadi, Director of Grid Analytics and Technology at BluWave-AI, for providing great feedback and relevant comments and ideas during our bi-weekly meetings, and always finding time for extra meetings to discuss progress. I would also like to thank Dr. Fakhri Karray, who has introduced me to the area of machine learning by sharing his knowledge through the outstanding lectures, and for his generous provision of the computer with GPU access, necessary to carry the thesis experiments. I am also thankful to him and Dr. Kankar Bhattacharya for serving as members of my committee, and for their insightful comments and feedback. In addition, I would like to thank Mr. Gerald Giroux and Mr. Greg Gaudet from Summerside Electrical for providing access to data, used in this work.

This has been a long journey and I am grateful for the support of my friends at the EMSOL lab, who shared both good and bad moments with me: Omar Alrumayh, Talal Alharbi, Chioma Anierobi, William Mendieta, Enrique Vera, Dario Peralta, Carlos Espinosa, Fabian Calero, Ivan Calero, Matheus Ferreira Zambroni de Souza, Mauricio Restrepo, Baheej Alghamdi and Sofia Guzman. Thank you for creating a friendly working environment in the lab.

My thesis and degree would not be possible without the help of my family. This work is dedicated to my dad Elmar Mammadov, my mom Ella Nasirova, and my sister Fidan Mammadova for their unconditional love and support.

Table of Contents

List of Figures

List of Tables

Acronyms

AdaBoost ADAptive BOOSTing

Adam Adaptive moment estimation

ADF Augmented Dickey-Fuller

AF Activation Functions

AI Artificial Intelligence

ANN Artificial Neural Network

AOI Anomaly Operation Index

ARIMA Auto-Regressive Integrated Moving Average

ARMA Autoregressive Moving Average

CEC Constant Error Carousel

CMS Condition Monitoring Systems

CNNs Convolutional Neural Networks

DirRec Direct-Recursive

ETC Extra Trees Classifier

EWMA Exponentially Weighted Moving Average

FN False Negative

FP	False Positive
GANs	Generative Adversarial Networks
GHGs	Greenhouse Gases
HMM	Hidden Markov Model
KPSS	Kwiatkowski-Phillips-Schmidt-Shin
LCOE	Levelised Cost of Energy
LSTM	Long Short Term Memory
MICE	Multiple Imputation by Chained Equations
MLCC	Multi-Layer Ceramic Capacitors
MSET	Multivariate State Estimation Technique
NARX	Nonlinear Autoregressive neural network with eXogenous input
NF	Neural-Fuzzy
O&M	Operation and Maintenance
PACF	Partial AutoCorrelation Function
PEI	Prince Edward Island
ReLU	Rectified Linear Unit
RESs	Renewable Energy Sources
RNF	Recurrent Neural Fuzzy
RNN	Recurrent Neural Networks
RPM	Rotations Per Minute
RUL	Remaining Useful Life

SCADA	Supervisory Control and Data Acquisition
SMOTE	Synthetic Minority Oversampling Technique
SPRT	Sequential Probability Ratio Test
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
VAE	Variational AutoEncoders
XGBoost	eXtreme Gradient Boosting

Chapter 1

Introduction

1.1 Motivation

The need for fossil fuels has expanded as the global energy demand has been increasing, which has been met by supplies of coal, oil, and natural gas. As it can be seen from the Figure 1.1, this demand is heavily met by the aforementioned resources.

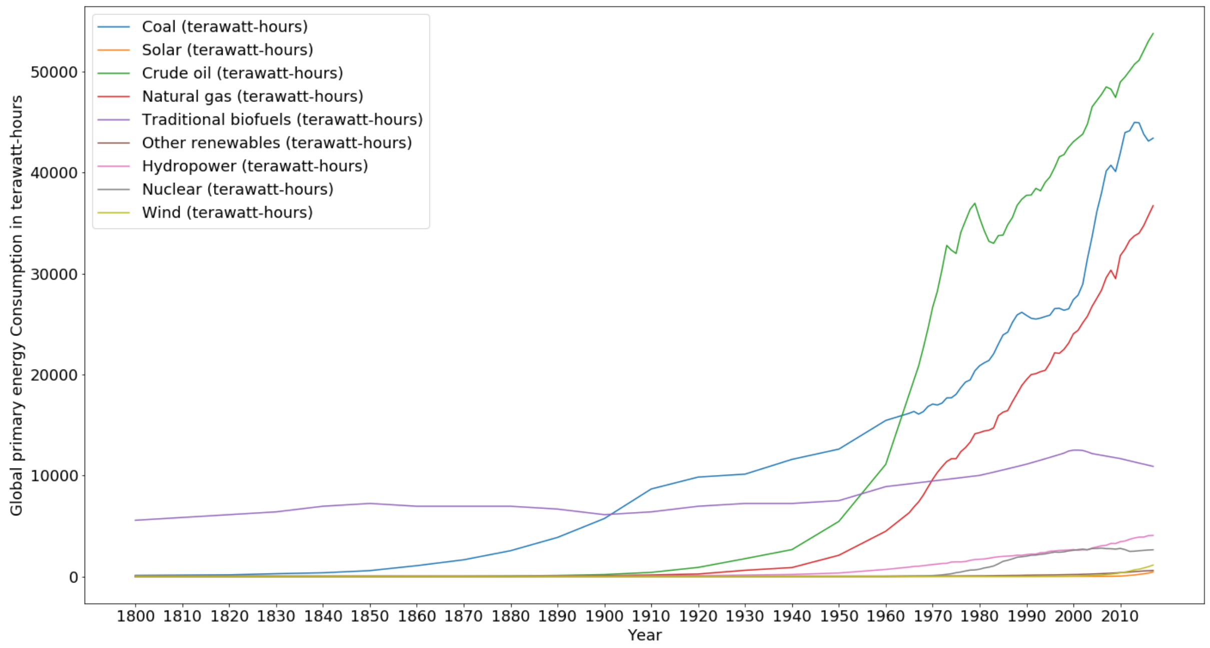


Figure 1.1: Global Primary Energy Consumption from 1800 to 2017 [1].

While being a widespread energy resource, fossil fuels present a significant problem to the environment in the form of combustion by-product: carbon dioxide, hydrocarbons and other greenhouse gases Greenhouse Gases (GHGs), resulting in a rapidly warming global climate. This issue has united countries and pushed them to propose such policies as the Kyoto Protocol and the Paris agreement to mitigate the negative impact of GHGs. One of technological approaches that can help to reduce the effect of greenhouse gases is utilization of natural resources via Renewable Energy Sources (RESs). As [2] states, RESs were initially used as back-up units and were deployed only when primary power supply was interrupted. As one of the RES, wind turbines are reported to be rapidly expanding [3], with the world-wide wind power installed capacity exceeding 539.58 GW [4].

In order to compare the cost of power production from wind turbines with other generation technologies, Levelised Cost of Energy (LCOE) metrics are utilised, which is used as a way of showing the average price of electricity that the asset must receive over its operating life in order for it to break even. Figure 1.2 shows the range of LCOE for solar, onshore, and offshore wind compared to fossil fuels, demonstrating wind being a cheaper alternative compared to solar generation technologies. LCOE is calculated as [5]:

$$LCOE = \frac{ICC \cdot FC + LRC}{AEP_{Net}} + O\&M, \tag{1.1}$$

$$AEP_{Net} = AEP_{Gross} \cdot Availability \cdot (1 - Loss) \tag{1.2}$$

where ICC is the initial capital cost, FC is the fixed charge in (%/year), O&M is the operation and maintenance cost, AEP is the annual energy production (kWh/year), and LRC is the levelized ($/year) replacement cost. From (1.1), it follows that even though wind turbines offer cheaper LCOE, the high Operation and Maintenance (O&M) will drive the LCOE up. It has been reported in [6] that O&M accounts for 10-15 % and 20-25 % of the overall LCOE for onshore and offshore wind turbines, respectively, during their 20 years of operating life, and can reach up to 35 % towards the end of their life [7]. Since there is a strong correlation between O&M expenses and LCOE, minimizing those expenses will reduce the LCOE.

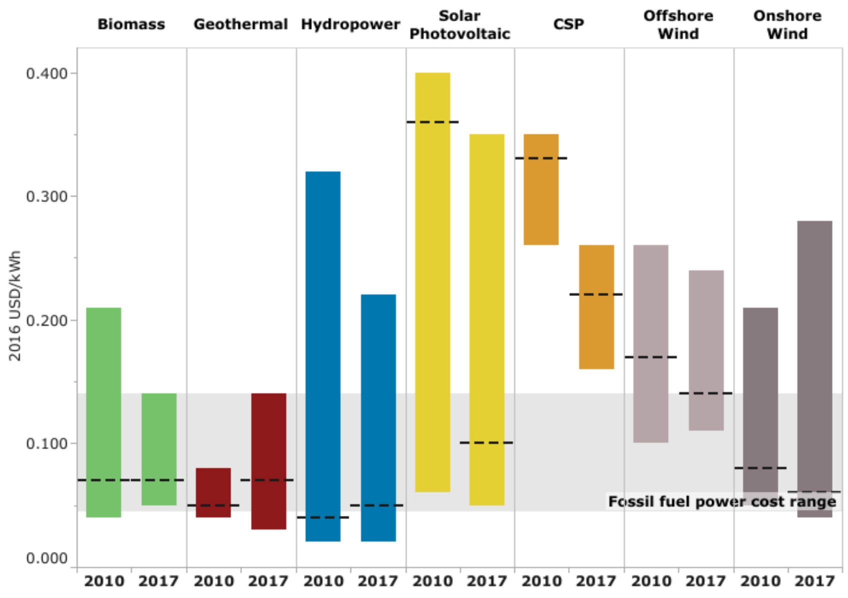


Figure 1.2: Comparison of LCOE from renewable power generation technologies compared to fossil fuels for 2010 and 2017 [8]. Permission to re-use this image was obtained from IRENA.

To ensure the proper working conditions and prolong the useful life of an equipment, wind turbines are serviced according to one of following types of maintenance [9]:

Corrective Maintenance: Corrective or “run-to-failure” maintenance is performed only when the failure occurs and a failed component is replaced with a new one. Even though this type of maintenance is considered a poor choice, it has been surprisingly popular, mainly due to being the cheapest alternative to implement, and still remains a widely used type of maintenance nowadays [10], with ongoing research efforts [11]. The major problem with corrective maintenance is that a failure might happen at the most inconvenient time, such as produced power is at its maximum level. Furthermore, due to the unplanned nature of a fault, spare equipment might not be available and thus, downtime would be prolonged. As the result, even though this is the cheapest type to implement, the final financial losses outweigh the benefits [12].

Reliability-Centered Maintenance: It was introduced in the aircraft industry as a way to optimise maintenance schedules [13]. Reliability-centered maintenance is a qualitative strategy that is a combination of tools and methods that help to identify the minimum set of preventive tasks that are required to address serious component failures without compromising service reliability [14]. This approach consists of breaking an asset down into various subsystems, and identifying the list of possible failures and their root cause. Once this information is collected and identified, a suitable maintenance strategy is related based on a prescribed list of evaluation steps. The main outcome of this maintenance strategy is a greater understanding of how different assets work and how they can be maintained, which leads to improved maintenance cost effectiveness [15].

Preventive Maintenance: This consists of periodic maintenance actions and replacement of components, scheduled at regular time intervals [10]. This type of maintenance is aimed to minimize both the downtime and the risk of unexpected failures. The time intervals between scheduled maintenance are decided by domain experts based on the historical performance of a component. Furthermore, in preventive maintenance, it is assumed that the failure behaviour of the equipment is predictable. While this approach results in avoiding long downtimes, the number of times when a turbine is unnecessary brought offline is fairly high [16]. To address this shortcoming, predictive maintenance has been introduced.

Predictive Maintenance: In this case, the maintenance is scheduled in advance based on condition monitoring techniques, which involve a constant monitoring of various parameters such as drive train vibration, lubricating oil quality, and temperature through a number of sensors installed in different parts of a turbine. Even though the benefits of using Condition Monitoring Systems (CMS) are clear, the significant costs of retrofitting sensors deter operators from installing these systems [17]. An alternative is to use Supervisory Control and Data Acquisition (SCADA) systems, which are commissioned with all large scale wind turbines. These types of systems record data with minutes granularity and as the result, can be used as a low-cost solution for early failure detection.

As a SCADA system is an integrated part of any production facility, the collected operational and status data may be used to detect incipient faults. However, since the constant manual inspection of the collected data is not feasible due to the size of the database, machine learning algorithm can be employed, as they they allow to detect complex patterns in data. Integrated with an existing SCADA systems as a part of a predictive maintenance program, these algorithms could predict upcoming failures and thus, minimize the downtime of a facility and maximize profits. Therefore, this thesis is focused on utilizing collected SCADA data for maintenance scheduling of wind generation in the plant.

1.2 Literature Review

This section presents an overview of some relevant research works for this thesis, focusing on predicting a fault. Unlike the diagnostics area, fault prognostics is a relatively new and fairly challenging domain, as it entails significant uncertainty [18]. As defined by the International Organization for Standardization: "prognostics is the estimation of time to failure and risk for one or more existing and future failure modes", i.e., estimation of the Remaining Useful Life (RUL) or of the probability that a systems operates without failing until the next inspection interval [19]. Research works cited in this section can effectively be categorized in 3 distinct categories: model-based, data-driven, and hybrid approaches [20], [21], as discussed next.

1.2.1 Model-based Prognostics

Model-based approaches, also known as physics-based techniques, depend on the accurate mathematical representation of a physical system, and apply mostly for component-level prognostics [22]. This is a white-box approach, since parameters are extracted from the system physics, which leads to interpretable models. Model-based prognostics rely on the acquired knowledge and detailed understanding of mechanical, electrical, or chemical processes that affect normal machine operations [23]. As the result, to account for a potential fault, a detailed knowledge of the system and factors that affect it is required.

In [24], the authors attempt to predict gearbox failure based on the trending of recorded oil temperature, vibration amplitudes, and oil debris particle counts. By monitoring transmission efficiency and rotational speed, and relating those parameters to the temperature rise, gearbox failures can be predicted up to 6 months in advance.

Reference [25] proposes an approach that is based on monitoring fault progression in the cases when a damage process ("hidden") occurs on a slower time scale than the observable dynamics ("fast") in a system. The proposed methodology consists of three distinct steps; in the first step, a tracking function is calculated based on a reference model short-time prediction error, and then used to approximate a tracking metric. The estimated tracking metric is used an an input to a nonlinear recursive filter that allows to evaluate the current damage level. This estimation becomes an input for the last stage of the proposed algorithm, time-to-failure is obtained from a Kalman filter by using discrete-time state transition and measurement equations, under the assumption of a particular damage evolution model. The developed algorithm is validated against an electromechanical system with a discharging battery that is treated as a hidden damage process. Simulations demonstrate that the presented algorithm can predict a failure 5 hours in advance of an actual failure.

In [26], the authors present an approach to approximate time-to-failure for rolling bearings, by combining sensor data for diagnostics with a mathematical model and historical data for prognostics. The developed mathematical model is based on Kotzalas/Harris theory and focuses on spall progression. This model provides damage build-up information in advance of a failure, while sensors supply essential information such as oil condition or debris monitor outputs and vibration signatures, which are used to update the assumptions in the model and reduce the uncertainty in RUL prediction.

The authors in [27] present an integrated model-based diagnostic and prognostic framework that has been tested in a simulated propellant loading system that includes tanks, valves, and pumps. The approach makes use of a common modeling paradigm that accounts for both the nominal behaviour and fault progression. The detection problem is formulated by representing the system under nominal operations, and marking changes in the nominal state of the system as faults. Measurement residuals are used along with prognostic predictions of how each measurement is expected to deviate from the nominal for each possible fault in the system to generate a set of fault candidates, which explain the observed deviations in measurements. The developed model is able to predict RUL of a pump with the mean accuracy of 70%, 2.7 hours prior to a failure happening.

Model-based approaches are challenging, as they have to be developed for a specific component and each one would require a different mathematical model. Furthermore, these models require extensive experimentation and verification, and changes in system dynamics and operating conditions can potentially affect the model, plus it is impossible to model for all real-life conditions. In addition to that, model-based methods have a limited capability in generating degradation behaviour for complex systems, where several faults can take place [28]. Therefore, in this work, model-based approaches will not be considered.

1.2.2 Data-Driven Prognostics

Even though data-driven prognostics are considered black box approaches, they offer undeniable advantages over model-based ones. These algorithms rely on historical data and transform collected raw measurements into relevant information to discover complex behavioural patterns of a system. As the result, the main advantage that data-driven prognostics is the lack of need to have strong expertise about a system, instead relying on the collected measurements [29]. Data-driven prognostics are conventionally divided into machine learning and statistical approaches [21], as discussed next.

Statistical Prognostics

While statistical and machine learning approaches have a similar end goal, statistical methods aim to estimate a function $f(X)$ as follows [30]:

$$y = f(X) + \epsilon, \tag{1.3}$$

where y is the dependent variable, $X = (X_1, X_2, \ldots, X_p)$ are independent variables, ϵ is a random error term, which is independent of X with $\mu = 0$. In this case, the RUL is approximated by fitting the probabilistic model to the collected data [20]. Since statistical modeling is focused on formalization of relationships between variables in the form of mathematical equations, various prior assumptions must be made [30]. Statistical approaches are simple and they capture linear relationships between the target and independent variables, however, nonlinear relationships are overlooked [31].

In [32], the authors use a logistic regression model to calculate the probability of failure for given condition variables and an Autoregressive Moving Average (ARMA) model, to trend the condition variables for failure prediction. The approach is to an elevator door motion system, where the initial signs of abnormal activity is detected 50 days prior to the failure.

The authors in [33] present two probabilistic methods based on Hidden Markov Model (HMM) for the prediction of incipient faults in the case of roller bearings. The data used in this work consist of vibration signals, composed of 20480 samples, recorded every 10 minutes with a sampling frequency of 20 kHz. The collected data are separated into classes, and the degradation process is estimated by two probabilistic methods based on HMMs. The performances of both methods are evaluated in the prediction of an impending fault for a roller bearing degradation, using 4 degradation stages, predicting failures with various time horizons depending on the stage.

In [34], the authors employ the residual delay-time concept and stochastic filtering theory to estimate the RUL for 6 rolling bearings. The bearings are tested in a laboratory setting, and the vibration signals are recorded. The proposed model is based on the idea of conditional residual delay time, which differs from the conventional concept of residuals that depend on the current age of the monitored equipment, depending also on the condition information available to date. The residual time distribution used the collected data to fit the model, with the trained model suggesting a preventive maintenance 58 hours before the actual failure.

Due to the fact that statistical approaches only capture linear relationships along with prior assumptions that must be held, these techniques will not be considered in this work.

Machine Learning Prognostics

Machine learning methods are based on the application of artificial intelligence algorithms, where failure patterns are learned from the collected historical data. These approaches are divided into supervised and unsupervised methods. Supervised learning is applied in cases where the data are labeled according to operating and failure modes. Unsupervised approaches are applied to unlabelled data, grouping data points in order to detect anomalous regions.

In [35], the authors use SCADA system data from wind turbines for wind turbine performance monitoring and prognostics, using provided power curves from a manufacturer, where deviations would indicate abnormal behaviour. The power residual (difference between measured actual power and the power expected) is used in constructing the baseline average. A three-sigma deviation from the baseline mean is used to build the upper and lower bounds on the residuals, which would detect abnormal activities in the system and minimize false warnings. Once the first signs of anomaly are captured, higher order statistical methods, such as skewness and kurtosis that the authors call *condition indicators*, are further applied to the anomaly detection and prognosis in different time windows. These condition indicators demonstrate the first signs of an impeding failure 20 days before the actual fault.

The authors in [36] analyze bearing faults in wind turbines using Artificial Neural Network (ANN). High frequency SCADA data, recorded with 10 second intervals, are collected from 24 wind turbines to be used for analysis of bearing faults. In order to detect faults, a training dataset is constructed from turbines that have not been affected by any faults. 5 different NNs with varying parameters (number of nodes and activation functions) are modelled and trained with respect to several performance metrics: absolute error, mean absolute error, relative error, mean relative error, and coefficient of determination. A data-mining approach is applied to detect generator bearing anomalies. The NN models are first tested on two different turbines for model validation, and the error residuals are analyzed using an average moving window of 360 data points (an hourly averaged error residual). The best performing model is selected to test the normal behaviour of wind turbines and detect anomalies in bearing temperatures. The same model is used for prediction of a fault, where the fault is detected 51 minutes prior to the actual fault occurring.

In addition to the previously cited work focused on predicting abnormal behaviour in bearings, [37] applies a data-mining model to predict faults of a blade pitch. Faults such as blade angle asymmetry and blade angle implausibility are analyzed to determine how they affect the wind turbine performance. The paper is focused on predicting a fault associated with the blade pitch angle using: genetic algorithms, ANN, k-nearest neighbors, partial decision trees, and bagging. SCADA data used to train and validate models are collected at 1-second intervals from 27 wind turbines for three months, and consist of both operational parameters such as power, wind speed, rotor speed, and status codes, along with status descriptions. The prediction problem is framed as a classification problem and models are evaluated on such metrics as accuracy, sensitivity and specificity. Models are trained on one third of the data and validated on the remaining data, with the best result being achieved from genetic algorithms with a prediction horizon of 10 minutes with the following results: 68.7% for accuracy, 71.2% for sensitivity (also known as recall), and 66.9% for specificity.

In [38], the authors apply Support Vector Machine (SVM) algorithms to classify and predict various faults. The SCADA data used in this research is collected from a turbine in the South-East region of Ireland, divided into 3 separate datasets: operational data, status data, and warning data. Warning data is ignored in the simulations, and both operational and status data are used for model training. Status data contain information on faults and warnings that can be used for supervised classification, and operational data contains historical information about turbine performance. The authors focus on both fault detection and prediction, where the initial goal of an algorithm is to separate faults from data points corresponding to a normal operating mode. The second stage consists of classifying a specific fault: feeding, air-cooling, excitation, generator heating, or mains failure faults. The last stage is to predict a specific fault in advance. The results vary based on the fault, where 100% predictions are obtained in the case of generator heating faults, however the problem of data leakage is present, possibly resulting in overoptimistic predictions.

The previous work is extended in [39], where the authors vary the prediction window and compared such approaches as Synthetic Minority Oversampling Technique (SMOTE) technique [40], random under-sampling and class weights to solve the problem with imbalanced classes. This work focuses on predicting generator heating and excitation faults, with a maximum prediction time horizon of 12 hours. The results presented for the aforementioned prediction window are not as accurate as in [38]. Even though the recall metrics demonstrate a fairly good result, the data are shuffled prior to splitting into training and testing sets and as the result, the problem of data leakage is once more present.

Similarly to [38] and [39], [41] frames the problem as a classification problem and divides the work into 3 distinct stages: (1) prediction of a fault; (2) prediction of a fault severity; (3) prediction of a specific fault. SCADA data recorded with 5 minutes frequency are collected for 3 months from 4 wind turbines, in two separate data sets: operational and status/fault data. The prediction window is varied from 5 minutes up to 1 hour, and models are evaluated according to accuracy, sensitivity and specificity. At each stage, different set of models are trained on the two-thirds of data, and tested on two separate test sets, i.e. for Test Set 1 with the remaining third of data, while for Test Set 2 randomly selecting 10% of the data from the labeled SCADA data. For stage 1, the models trained are: ANN, an ensemble of ANNs, boosting tree, and SVM, with an ensemble of ANN demonstrating the best performance with a prediction horizon of 1 hour. For stage 2, which consists of predicting a fault category, the models trained are: ANN, classification and regression tree, a boosting tree algorithm and SVM, with CART showing the best results. For stage 3, i.e., prediction of a fault, the models used are: boosting tree algorithm, ANN, ANN Ensemble, and SVM, with ANN Ensemble demonstrating the best result. It is worth noting that when the prediction horizon is increased from 5 minutes to 1 hour, all metrics demonstrate stochastic behaviour, which can be explained by data leakage coming from the random splitting of the data set into training and testing ones.

As not all operators provide SCADA data along with the information about faults and warnings, it is common to establish a normal operating model and any deviation from the model results is classified as a fault. Thus, in [42], the authors attempt to predict gearbox main bearing temperature and lubrication cooling oil temperature through analyzing 2 years worth of SCADA data with a 10-min recording frequency, collected from 26 Bonus 600 kW stall-regulated turbines located in Scotland. The first part of the paper focuses on detecting faults, which is achieved by estimating normal model behaviour via ANNs. The choice of the ANN model parameters is ensured via cross-correlation, where different parameters are omitted one by one and the model accuracy is checked after each step. After training various ANN models using 2 years worth of data points that correspond to normal behaviour, the authors are able to detect both gearbox and cooling oil faults. In the specific case of the cooling oil model, overheating incipient problems were detected almost 6 months in advance of the actual failure.

In [43], the authors aim to predict a fault in wind turbine generators in advance, diagnosing the state of the wind turbine generator when the fault occurs. The proposed solution is deployed and evaluated in two real-world wind power plants in China, demonstrating that the time-to-failure of the generators can be predicted 18 days ahead with about 80%, accuracy and when faults occur, the specific type of a generator fault can be diagnosed with an accuracy of 94%. SCADA data collected over a period of 18 months from two real-world wind power plants is used to identify correlated features and principal component analysis is applied to combine data with these features. A density-based spatial clustering of applications with noise method is employed to form clusters; however, since a state change from healthy to unhealthy is a continuous process, a new metric called Anomaly Operation Index (AOI) is proposed to measure the wind turbine's performance over time. The characteristics of the AOI are combined with an Auto-Regressive Integrated Moving Average (ARIMA) model, predicting time-to-failure 18 days ahead with an average prediction accuracy of 80%.

In [44], the authors approach the problem of wind turbine breakage prognosis and diagnosis by using deep autoencoder models for dimensionality reduction [45]. The autoencoders are trained on the data that corresponds to normal operating modes, which using a test data set that consists of both faulty and normal data points, calculate the square of the Euclidean distance between the reconstructed inputs and the original inputs; wind turbines with impending blade breakages show higher reconstruction error. Based on this metric, an Exponentially Weighted Moving Average (EWMA) method is applied to estimate the upper and lower limits, with the EWMA control charts being tested on 3 wind farms, detecting impending blade breakage 7 hours and 20 minutes, 6 hours and 10 minutes, and 8 hours, prior to a fault happening for each wind farm. Furthermore, it is shown that when models are trained on the full set of data without fault data points being filtered out, the EWMA control charts are only able to detect a fault 2 hours in advance; however, it is not clear if the data used to train the model is separated from the data used to validate it.

In [46], the authors built various ANN-based condition monitoring approach using a Nonlinear Autoregressive neural network with eXogenous input (NARX) to estimate the condition of the gearbox bearings. SCADA data collected from onshore wind turbines, located in the south of Sweden, is used to calculate a Mahalanobis distance measure based on a fault-free training data set, and the threshold is used to determine presence of an anomaly in the bearings. The proposed ANN model is trained on the fault free data and during the validation phase, and the trained model is able to detect a gearbox failure 1 week in advance.

Based on the aforementioned review, and considering that machine learning methods do not require prior assumptions about the underlying relationships between variables and focus on detecting the complex data patterns, these methods form the basis for the research work presented in this thesis.

1.2.3 Hybrid Prognostics

A hybrid approach is an integration of model-based with data-driven prognostics approaches. This type of technique combines the strengths of both prognostics methods to achieve finely tuned models that have better capability to manage uncertainty and can result in more accurate predictions [20].

In [47], the authors use an available physics-based model relying on a particle filtering method to predict the RUL of turbine blades. A Monte Carlo-based technique, known as particle filtering is employed to predict their probability density function to describe the degradation state, updating it when new observations are collected. The approach is tested to predict RUL of turbine blades, comparing the results against the RUL value, obtained in lab settings. The results are also compared against the variance used to describe the amount of uncertainty associated with RUL predictions. The estimated RUL is 97% accurate, but the variance increases by 18%. due to the prognostic model regression error.

The authors in [48] use data-driven models to predict future measurements, while a physics-based model is used to approximate the time-to-failure in lithium-ion batteries. The hybrid prognostic algorithm incorporates data-driven prediction into a particle-filtering learning structure, which allows to the predicted future measurements from the data-driven model to be properly managed by the particle-filtering algorithm, in order to keep updating the system model parameters during the prognostic period. ANN, Neural-Fuzzy (NF), Recurrent Neural Fuzzy (RNF) data-driven models are trained offline using historical data collected from a system with similar degradation process, and further tuned using the available data from the target system in order to account for the system dynamics. The forecast measurements from the data-driven predictors are utilized by the particle filtering through Bayesian learning to update the prediction model parameters in a recursive manner. The developed fusion prognostics framework consists of two separate steps: the state estimation and future state forecasting. During state estimation, the fusion frameworks estimates the states in parallel with model parameter identification based on the battery parameters. During the forecasting period stage, the identified model parameters are further tuned based on the the predicted system evolution from the data-driven predictor. The proposed fusion prognostic framework predicts an error 1.59 weeks in advance, which is shown superior to both the particle filtering (33.25 weeks late) and RNF (4.18 weeks late) approaches.

In [49], the authors present a fusion prognostic method in the application of Multi-Layer Ceramic Capacitors (MLCC). In order to detect a fault, a health baseline is estimated by using training data from 5 MLCCs that have not experienced failure. The data is then monitored by a Multivariate State Estimation Technique (MSET) and Sequential Probability Ratio Test (SPRT) algorithms, where MSET is used to monitor the status of the capacitors and estimate the residuals for each parameter, and SPRT is used to detect anomalous behaviour from the calculated residuals. The approach results in detection of failures in the range of 875 to 920 hours, with the actual failure taking place at the 962 hour.

The authors in [50] develop a hybrid model for degradation monitoring and fault prediction in offshore wind turbines. In this paper, physics-based models of degradation of a component and RUL are estimated from historical data of failures, or in cases of lack of this information, based on average failure rates. A prognostic model is then obtained using a Bayesian updating approach, to update the parameters of the proposed mathematical models.

While the previous literature review shows that hybrid methods are powerful, these require mathematical models and are focused on components rather than the whole system. Hence, since this thesis focuses on predicting incipient faults in a wind farm, these hybrid methods will not be considered here.

1.3 Research Objectives

Based on the aforementioned discussions, the objectives of this thesis are:

- Analyze collected information on historical failures of wind turbines, and determine how this information can be used to develop machine learning models for failure prediction.

- Develop machine learning models by tuning hyper-parameters using time-series cross-validation techniques, ensuring that no data leakage is present, and addressing such data limitations, as imbalanced data sets and missing values.
- Identify the most useful features for making a prediction through feature selection algorithms, and interpret the results and compare the outputs of various algorithms to identify the most suitable algorithm.

Collected data from Summerside wind farm, located in Prince Edward Island, will be used to develop, test, and validate the proposed fault prediction technique.

1.4 Thesis Outline

The thesis is structured as follows:

- Chapter 2 presents a background review of the main concepts in this research work. The chapter is focused on discussing multivariate time-series problems and appropriate tools for time-series analysis. A general review of data preprocessing, including missing data imputation, features selection and data reduction, is also presented. Then, the theoretical background of the various machine learning models used here is discussed, along with the parameters that affect the learning phase. Finally, metrics used to evaluate models are presented.
- Chapter 3 presents and discusses the prediction models developed, along with the results of applying these models. A comparison of the various prediction models and techniques is also presented, identifying the best models for the dataset used.
- Chapter 4 presents a summary and main contributions of the research work, identifying possible future extensions and improvements.

Chapter 2

Background Review

This chapter reviews the theoretical background of the main concepts on which the work presented throughout this thesis is based. First, here time series analysis is discussed along with the concepts of forecasting that are used for fault prediction. Then, data preprocessing steps are discussed, and finally, the machine learning methods used here, i.e., SVM, ANN, Long Short Term Memory (LSTM), and ensemble models are reviewed.

2.1 Time Series Analysis

2.1.1 Definition

These are two distinct approaches to time series analysis: time domain and frequency domain approaches [51]. In this thesis, the focus is on time domain approaches, where the primary focus is analyzing lagged relationships, as oppose to frequency domain techniques, where the objective is to investigate cycles [51].

In various applications, including the one discussed here, metrics of interest are collected at regular intervals to form an ordered sequenced of n real-valued variables. In the cases of having a single monitored feature, time series are *univariate* and mathematically expressed as:

$$X = \{x_1, x_2, \ldots, x_n\} \quad \forall\, x_n \in \mathbb{R} \tag{2.1}$$

where $\{x_i, \ldots, x_n\}$ are discrete data points in time $\{t_1, \ldots, t_n\}$. However, it is more common to encounter *multivariate* time series, where a vector of features in subsequent moments of time is observed and defined as;

$$X = \{x_1, \alpha_1, \beta_1, ..., x_n, \alpha_n, \beta_n\} \quad \forall\, x_n, \alpha_n, \beta_n \in \mathbb{R} \tag{2.2}$$

where $\{x_1, \ldots, x_n\}$, $\{\alpha_1, \ldots, \alpha_n\}$, and $\{\beta_1, \ldots, \beta_n\}$ are data points associated with different and distinct features.

2.1.2 Components of Time Series

General time series are affected by 4 major components: trend, cyclical, seasonal, and irregular ones [52]. The tendency of a time series to increase or decrease over the period of time is known as a *trend*. *Seasonal* variations in time series are fluctuations within a year during the season, due to climate and weather conditions, as well as habits. *Cyclical* variations in time series describe medium-term changes in the series, caused by certain circumstances repeated in cycles; the duration of a cycle extends over long periods of time, such as two or more years. *Irregular* or random variables are caused by unpredictable influences and do not depend on a particular pattern; there are no defined statistical techniques for measuring random fluctuations in a series.

The interactions of the aforementioned components result in one of the two distinct types of models, namely, additive or multiplicative [53], as follows:

$$Y(t) = T(t) \times S(t) \times C(t) \times I(t) \tag{2.3}$$

$$Y(t) = T(t) + S(t) + C(t) + I(t), \tag{2.4}$$

where $Y(t)$ is the observation, $T(t)$ is the trend, $S(t)$ is the seasonal component, and $C(t)$ and $I(t)$ are cyclical and irregular components at time t, respectively. The main difference between these 2 models is that, in the multiplicative model there is the assumption that 4 components can affect each other, while in the additive model it is assumed that the four components are independent of each other [52].

The major difference between modeling time series data via machine learning models and other domains such as computer vision, is that time series cannot be assumed to be independent and identically distributed (i.i.d.) variables, as they follow some pattern in the long term. This leads to limitation in data splitting and cross-validation techniques; thus, the dataset must be split in a way to preserve the temporal components inherent in the problem. Violation of this principle leads to the data leakage problems and inflated results [54].

2.1.3 Concept of Stationarity

A stationary time series is one whose properties such as mean and variance do not depend on time [55]. Time series with trends or seasonality are non-stationary, as both trend and seasonality affect time series at different times. There are two types of stationary processes: strongly stationary and weakly stationary.

A process is *strongly stationary* if joint distribution of every collections of values $\{x_{t-s}, x_{t-s+1}, \ldots, x_t, \ldots, X_{t+s-1}, X_{t+s}\}$ is independent of t $\forall s \in \mathbb{N}$ [56]. Strong stationary implies the equal distribution of random variables of the stochastic process as the series gets shifted along the time index axis. However, this type of stationarity is too strong for most applications and as the result, a *weakly stationary* concept is needed.

Time series are known to be *weakly stationary* if the mean value, μ, of X_t does not depend on t and remains constant [57]. In addition to that, the autocovariance $\gamma(s, t)$ of two time points s and t of the series depends only on their distance $d = |s - t|$, leading to any time points being apart the same distance having the same autocovariance.

As mentioned in [58], the concept of stationarity is a mathematical idea constructed to simplify the theoretical and practical development of stochastic processes. In order to design a proper model, adequate for future forecasting, the underlying time series is expected to be stationary. However, in practice, time series are often non-stationary since they exhibit some persistent increase or decrease over time. For a given time series, a visual inspection can help to decide whether the time series is stationary. Except for some obvious cases, such as a clear trend, deciding if the time series is stationary is not simple; furthermore, a time series can be large enough to make the visual inspection unfeasible. To reduce the subjectivity in determining stationarity, units root tests are employed [59]. The 2 common tests, and the ones used in this work, are Kwiatkowski-Phillips-Schmidt-Shin (KPSS) and Augmented Dickey-Fuller (ADF).

In statistics, ADF tests are employed for testing whether a unit root is present in the time series sample, which implies that time series are difference stationary. The null hypothesis in ADF tests, H_0, applies to processes with a unit root. The alternate hypothesis, H_a, suggests that the time series does not have a unit root, meaning it is stationary. A non-stationary process with a deterministic trend is transformed into stationary one by removing the trend, or detrending [60].

KPSS tests are used for testing a null hypothesis, i.e., that an observable time series is stationary around a deterministic trend versus the alternative of a unit root. The null hypothesis, H_0, corresponds to the data being stationary, and the alternate hypothesis, H_a, suggests that the data is difference stationary. To correct this non-stationarity, the following difference of a time series with respect to its lag, known as differencing is used, resulting in a new time series [61]:

$$x'_t = x_t - x_{t-1} \tag{2.5}$$

2.1.4 Forecasting

Forecasting focuses on predicting future behaviour of a time series, based on the collected historical observations. In order to make accurate forecasts, several assumptions need to hold. Thus and first of all, future observations need to display a similar behaviour to collected data. Then, a collected time series must contain patterns that can be captured, so that machine learning techniques can learn those patterns in order to produce accurate results. Historically, forecasts are divided into short-, medium-, and long-term horizons [55]. The strategies used for forecasting are recursive, direct, and direct-recursive.

Recursive Strategy

The recursive strategy trains first a one-step model f that denotes the functional dependency between past and future observations [62]:

$$x_{t+1} = f(x_t, \ldots, x_{t-d+1}) + w_{t+1} \quad \forall t \in \{n, ..., N-1\} \tag{2.6}$$

where w_{t+1} is a stochastic independent and identically distributed error process with mean zero and variance σ^2, d is number of past values used to predict future values, also known as an embedding dimension of time series. The trained f is used to recursively predict a multi-step forecast.

A major drawback of the recursive approach is its sensitivity to the estimation error, since the error propagates into the forecasted horizon. Despite this, recursive strategies have been successfully utilized in real-world applications in conjunction with machine learning models such as recurrent neural networks [63].

Direct Strategy

The direct strategy learns independent H models f_h:

$$x_{t+h} = f_h(x_t, \ldots, x_{t-d+1}) + w_{t+h} \quad \forall t \in \{n, ..., N-H\}, h \in \{1, ..., H\} \tag{2.7}$$

and returns a multi-step forecast for each f_h model in an H horizon. Unlike the recursive strategy, the direct strategy does not compute forecasts based on the approximated ones and hence, errors do not accumulate. However, there are still drawbacks to this approach. Thus, since the H models are learned independently, no statistical dependencies between the predictions are considered [62]. Also, direct methods demand more computational resources, since the number of models to learn depends on the horizon size. Different machine learning models such as decision trees have been used to implement the direct strategy for multi-step forecasting tasks [64].

Direct-Recursive Strategy

The Direct-Recursive (DirRec) strategy, proposed in [65], uses a different model for every horizon and at every time step, introducing the approximations from previous steps into the input step. The DirRec strategy learn H models f_h from the time series $\{x_1, ..., x_N\}$, where:

$$x_{t+h} = f_h(x_{t+h-1}, \dots, x_{t-d+1}) + w_{t+h} \quad \forall t \in \{d, ..., N-H\}, h \in \{1, ..., H\} \tag{2.8}$$

Unlike the previous strategies, the embedding size is not the same for all horizons. The effectiveness of the strategy has been demonstrated in [65].

In this thesis, where the primary objective is to predict a fault happening in wind generators, a direct strategy is employed, as it is widely researched and adequately accurate.

2.1.5 Problem Framework

As it has been discussed in Section 2.1.1, general multivariate time series X can be represented via (2.2). Hence, the goal of this research work is to learn a mapping function f that will allow to predict a fault in advance, based on the historical pattern of the collected signals. A function f is called a classifier, and in this thesis, it is realized in a number of ways, namely, through ANNs, LSTM networks, SVMs, and ensemble learning. In the ideal case scenario, the model would classify new data correctly; however, this goal is unachievable with absolute accuracy and thus, the goal is to minimize an objective function, and evaluate the presented algorithms according to the metrics discussed in Section 2.3.5.

In this work, the mapping function f is a binary classifier that assigns one of 2 classes to each point in the input space. By using the window of both lead and lag time variables, the forecasting function f can be written as:

$$y_{t+h} = f(x_t, \alpha_t, \beta_t, \ldots, x_{t-d+1}, \alpha_{t-d+1}, \beta_{t-d+1}) \tag{2.9}$$

where d is the selected lags used to predict future values, and y_{t+h} is the binary target variables, with 0 corresponding to the“no-fault” or ”normal” operation mode, and 1 representing a “fault”. As soon as the trained classifier recognizes the pattern in the data that is characteristic of a potential fault, the target variable at the prediction horizon h will be classified as Class 1. The size of the embedding dimension d is selected according to one of the feature selection methods that are discussed in the Section 2.4.

The goal of any machine learning model is to minimize an objective function. Since the proposed classifier f is a binary one, the objective function that will be minimized is the following log-loss objective function or binary cross-entropy:

$$L(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) - \sum_{i=1}^{N} (1 - y_i) \log(1 - \hat{y}_i) \tag{2.10}$$

where y_i is a true target of the Class i, $\hat{y}_i$ is the predicted probability of the Class i, and N is the number of independent classes.

2.2 Data Preprocessing

2.2.1 Missing Data Imputation

While there are machine learning algorithms such as eXtreme Gradient Boosting (XGBoost) that are able to handle missing data, in this work, the focus is on the applications of ANNs, SVMs, and LSTMs that cannot process the dataset with missing values. However, as it will be discussed in the Chapter 3, the collected data from the Summerside wind farm in Prince Edward Island contain missing values, which could have occurred due to sensor malfunctioning. The most popular approaches to deal with missing values are dropping them, imputing them with the mean, or filling with the most common value [66]. In this work, Multiple Imputation by Chained Equations (MICE) is employed to address the dataset limitations. The detailed description of the MICE algorithms can be found in [67], and can be summarized as follows: missing values of a feature are obtained based on running multiple regression models and treating a missing value as a dependent variable, which leads to it being treated conditionally based on the non-missing values of other features.

2.2.2 Data Scaling

Data scaling refers to normalizing the dataset as part of data preparation prior to running machine learning algorithms. Any normalization procedure is characterized by changing values of numeric columns to a common scale, while simultaneously preserving the differences in the range of values. The main objective of utilizing various normalization techniques is to ensure that algorithms that calculate the distance between points via Euclidean distance will weigh all features equally, instead of being dominated by a feature with larger values. Among normalization techniques, the most popular ones are Min-Max, mean normalization, and scaling to unit length. In this thesis, *Z-score* normalization, also known as standardization, whic is characterized by rescaling features to have properties of a Gaussian distribution with $\mu = 0$ and $\sigma = 1$ is employed. Standardization is calculated according to:

$$z = \frac{x - \mu}{\sigma}, \tag{2.11}$$

where z is the standardized value of a sample x_i

2.2.3 Imbalanced Dataset

High imbalance occurs in any real-world applications where the goal is to detect rare but crucial samples. There are two major strategies that are used to address severe imbalance: data-level and algorithm-level techniques [68]. Data-level approaches vary the class distribution via sampling procedures, and these methods are separated into under-sampling and over-sampling classes. The most common over-sampling methods used are SMOTE [40], and ADASYN [69]. On the other hand, algorithm-level methods adjust the classifier to account for imbalanced dataset; the most common approach is to adjust the misclasification cost or decision threshold [70]. In this work, class weights are employed to handle imbalanced classes. The major outcome of applying class weights is to place more emphasis on the minority classes and thus, allow an algorithm to learn equally from all classes. Class weights are applied by modifying the objective function and as the result, (2.10) is modified as follows:

$$L(y, \hat{y}) = - - \sum_{i=1}^{N} \beta y_i \log(\hat{y}_i) - \sum_{i=1}^{N} (1-\beta)(1-y_i)\log(1-\hat{y}_i) \tag{2.12}$$

where β is the ratio of negative samples over total ones [71].

2.3 Machine Learning Models

Machine learning is a sub-field of Artificial Intelligence (AI). Machine learning algorithms find a pattern in the dataset through one of 4 learning paradigms: supervised, unsupervised, semi-supervised, and reinforcement learning. In this supervised learning is employed, with the primary goal of learning a general rule that infers target, y, from training data, X. As the goal of this work is focused on testing the viability of machine learning models for fault prediction, other methods such as advanced time series techniques are not reviewed.

The central problem in machine learning is that the trained model must perform well on new, previously unseen data, which is known as generalization; thus, the ultimate goal of a machine learning algorithm is to achieve a low generalization error. The factors that determine how well a model performs are its ability to minimize the training error, and ensure that the difference between training and testing errors is small. When both of these criteria are satisfied, and the model's predicted values are close to the true values, *goodness of fit* is reached. However, to reach that ultimate goal, it is common to face 2 central challenges in a model training procedure: underfitting and overfitting, as shown in Figure 2.1, where a statistical fit is demonstrated by fourth degree polynomial.

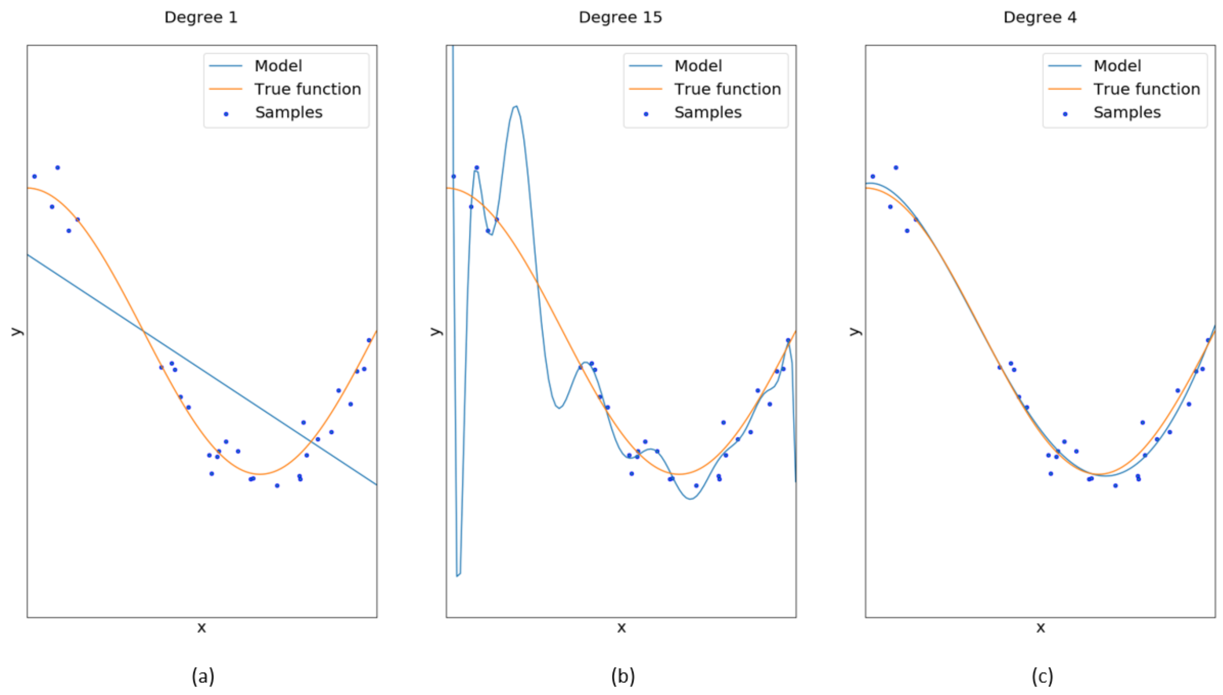


Figure 2.1: Examples of (a) underfitting, (b) overfitting, and (c) statistical fit cases [72].

Underfitting, also known as a high bias, occurs when a model cannot reduce either a training or a testing set error. The most common cause of underfitting is a model that is too simple to learn the underlying complexities of the data. In Figure 2.1, the first degree polynomial is not complex enough to fit the training samples.

Overfitting, also known as a high variance, is a more common occurence in training a model. Overfitting can be characterized by a good performance on a training set, but a poor one during the validation phase. Overfitting happens when the model is complex enough to learn the smallest details in the training set and as a result, not being able to generalize well on the test set. In Figure 2.1, degree 15 polynomial overfits the training data, as it learns the noise.

While underfitting can be solved in a simple manner by increasing model complexity, overfitting is a more complex problem. In this work, several methods to address overfitting are employed, with the most common one being a time-series cross-validation technique.

2.3.1 Hyperparameter Optimization

The hyperparameters for all the models presented in this chapter will be tuned according to Random Search [73], instead of other approaches such as grid search or Bayesian learning. This technique can be described as trying random combinations to find the best solution in the hypothesis space. In [73], the authors demonstrate that the probability of finding a combination of parameters within 5% of local optima is 95%.

Random Search will be combined with cross-Validation to test the effectiveness of a machine learning model. Traditional cross-validation techniques such as *K-fold* approach will not work here, as the underlying assumption of those approaches includes shuffling the dataset prior to splitting it, and as the result, data leakage problems occur, as the information from the future becomes part of the training set. To ensure that this problem is avoided, the cross-validation technique known as *walk-forward validation* [55], with the process being shown in Figure 2.2. This technique restricts the usage of the full set; instead, the dataset is split into blocks of contiguous samples, and at each split, the test data from the previous split becomes the part of the training dataset.

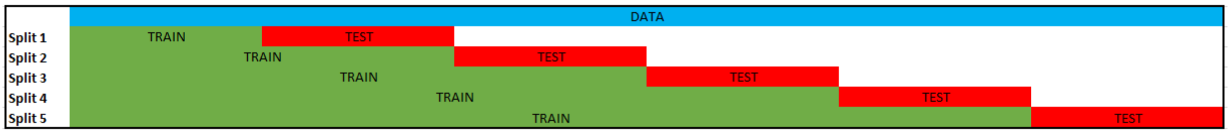


Figure 2.2: Walk-forward validation technique.

2.3.2 Support Vector Machine

Support Vector Machines (SVMs) were introduced in 1995 [74]. In addition to predictive maintenance [38], SVMs have been successfully applied in the areas of computer vision [75], speech [76] and handwritten digit recognition [77]. In this section, the theoretical framework of SVMs is presented.

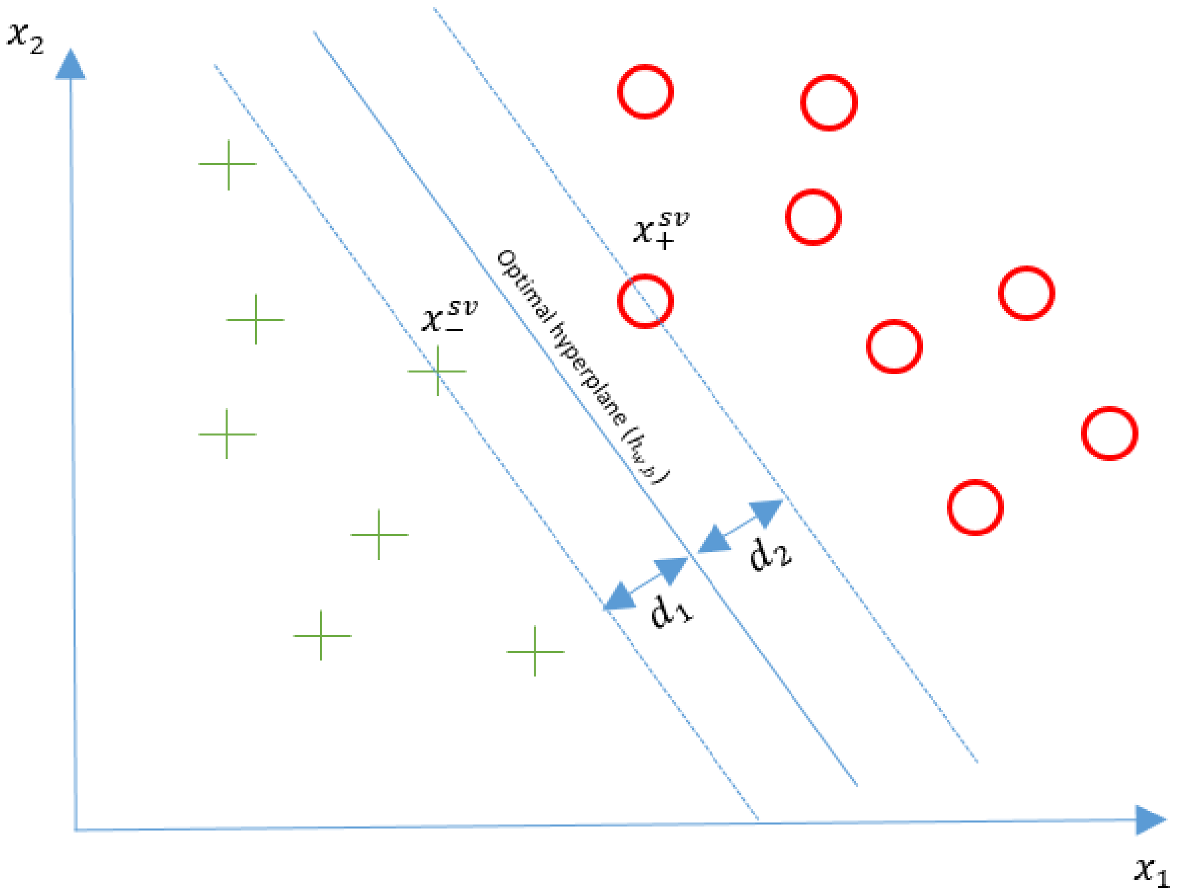


Figure 2.3: Support vector machine

SVMs are positioned as binary classifiers. A set of N samples is considered, where each input X_i is D-dimensional, and the targets consist of two classes: $y_i = +1$ and $y_i = -1$, i.e.:

$$\{X_i, y_i\} \quad \text{for} \quad y_i \in \{-1, 1\} \quad X \in \mathbb{R}^{\mathbb{D}} \quad i = 1, \ldots, N \tag{2.13}$$

The goal is then to find the hyperplane that defines the widest margin to separate both classes and such as:

$$h_{w,b}(X_i) = w^T X_i + b \tag{2.14}$$

where b is a bias term and w is the weight vector. The optimal hyperplane can be found in the infinite number of ways by scaling b and w [78]. Regardless of values for b and w, the canonical hyperplane will satisfy the following:

$$w^T X_+^{sv} + b = 1 \tag{2.15}$$

$$w^T X_-^{sv} + b = -1 \tag{2.16}$$

where both X_+^{sv} and X_-^{sv} are known as support vectors, and are the closest to the hyperplane in Figure 2.3.

If d_1 is defined as the distance from X_-^{sv} and d_2 is the corresponding distance from X_+^{sv}, then the hyperplane should be equidistant from both classes, i.e., $d_1 = d_2$. In this context, the following is defined as the margin [78]:

$$d(X_\pm^{sv}) = \frac{1}{||w||} \tag{2.17}$$

Hence, maximizing the margin is equivalent to minimizing $\frac{1}{2}||w||^2$, which allows to transform the problem into a primal optimization problem with a convex quadratic objective function, that can be solved using a quadratic programming optimizer as follows:

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}||w||^2 \\ \text{s.t.} \quad & y_i(w^T X_i + b) - 1 \geq 1 \quad \forall i = 1, \ldots, N \end{aligned} \tag{2.18}$$

This problem can be solved using Lagrange multipliers:

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{N} \alpha_i [y_i(w^T X_i + b) - 1] \tag{2.19}$$

where α_i are the Lagrange multipliers. Hence, the optimal is obtained as follows:

$$\frac{\partial L(w,b)}{\partial w} = w - \sum_{i=1}^{N} \alpha_i y_i X_i = 0 \Longrightarrow w = \sum_{i=1}^{N} \alpha_i y_i X_i \tag{2.20}$$

$$\frac{\partial L(w,b)}{\partial b} = \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.21}$$

Finally, substituting (2.20) and (2.21) into (2.19) leads to:

$$L(w, b, \alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j X_i^T X_j \tag{2.22}$$

This is known as the Dual of (2.19). Hence, the problem constraints, the following dual optimization problem can be formulated as [79]:

$$\begin{aligned} \max_{\alpha} \quad & L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j X_i^T X_j \\ \text{s.t.} \quad & \sum_{i=1}^{N} \alpha_i y_i = 0 \\ & \alpha_i \geq 0 \quad \forall i = 1, \ldots, N \end{aligned} \tag{2.23}$$

This is a convex quadratic optimization problem, which can be solved with a quadratic programming (QP) solver to get α^*, and from (2.20), the optimal w^* can be found. From w^*, the optimal value of the intercept term b would be:

$$b^* = -\frac{\max_{i:y_i=-1} w^{*T} X_i + \min_{i:y_i=1} w^{*T} X_i}{2} \tag{2.24}$$

If the data cannot be separated by a hyperplane, a soft margin is used, where the majority but not all data points are separated. In this case, (2.22) can be generalized by a slack variable ζ_i and a penalty parameter C, which controls the trade-off between the slack variable penalty and the size of the margin [80]; the effect of C is shown in Figure 2.4.

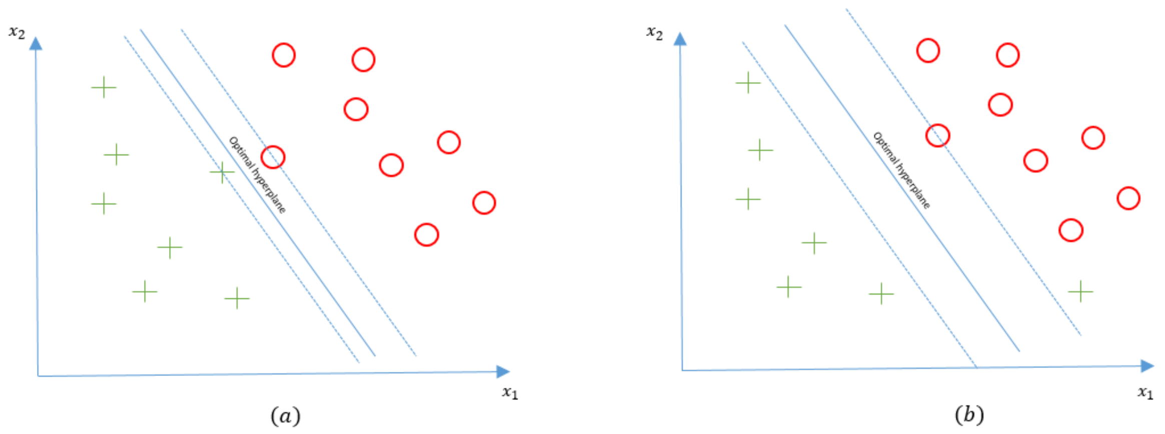


Figure 2.4: Effect of C parameter on the hyperplane: (a) large C values are assigned; (b) small C values are assigned.

The general formula for the linear kernel can be defined:

$$\begin{aligned} \min_{w,b,\zeta_i} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\zeta_i \\ \text{s.t.} \quad & y_i(w^T X_i + b) - 1 \geq 1 - \zeta_i \quad \forall i = 1,\ldots,N \\ & \zeta_i \geq 0 \quad \forall i = 1,\ldots,N \end{aligned} \tag{2.25}$$

Similarly to the case of the linearly separable data, Lagrangian is formed as:

$$L(w,b,\alpha,\zeta,r) = \frac{1}{2}w^T w + C\sum_{i=1}^{N}\zeta_i - \sum_{i=1}^{N}\alpha_i[y_i(X^T w + b) - 1 + \zeta_i] - \sum_{i=1}^{N} r_i\zeta_i \tag{2.26}$$

where α_i and r_i are the Lagrange multipliers. By setting the derivatives with respect to w and b to 0, and simplifying the resulting (2.26), the following dual optimization problem is formed:

$$\begin{aligned} \max_{\alpha_i} \quad & \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j X_i^T X_j \\ \text{s.t.} \quad & \sum_{i=1}^{N}\alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \quad \forall i = 1,\ldots,N \end{aligned} \tag{2.27}$$

In SVMs, the optimization problem (2.23) is solved in the convex space, obtaining the global extrema [78]. When the space is not linearly separable, a transformation to another space is required. That transformation is performed via the kernel function:

$$K(X_i, X_j) = \phi^T(X_i)\phi(X_j) \tag{2.28}$$

This is also known as the kernel trick, and computes the inner product between the mapped vectors, which leads to computational savings, when data are transformed into a higher-dimensional space. There are three kernels that are considered in this thesis: linear, polynomial, and Gaussian, respectively, as follows:

$$K(X_i, X_j) = X_i^T X_j \tag{2.29}$$

$$K(X_i, X_j) = (1 + X_i^T X_j) \tag{2.30}$$

$$K(X_i, X_j) = e^{-\gamma(||X_i - X_j||^2)}, \tag{2.31}$$

In Equation (2.31), γ is a parameter that defines the kernel width and as a result, the decision region [81]. The parameter C, the kernel, and γ are hyperparameters that tuned in order to get the best performance out of the model.

2.3.3 Artificial Neural Network

ANNs are computational models that have been inspired by the human brain, consisting of an interconnected network of single processing units that can learn from experience by modifying its connections. An ANN is based on a collection of connected units or nodes, known as artificial neurons. Here, feedforward NNs used.

A basic feedforward NN architecture is shown in Figure 2.5, consisting of three layers: input, hidden, and output layers. The data processing can extend over multiple layers of units, but unlike LSTM, discussed in Section 2.3.3, no feedback connections are present in the architecture. ANNs consisting of adjustable weights w, bias terms b, and activation functions f, compute their outputs y as follows:

$$y = f(w^T X + b) \tag{2.32}$$

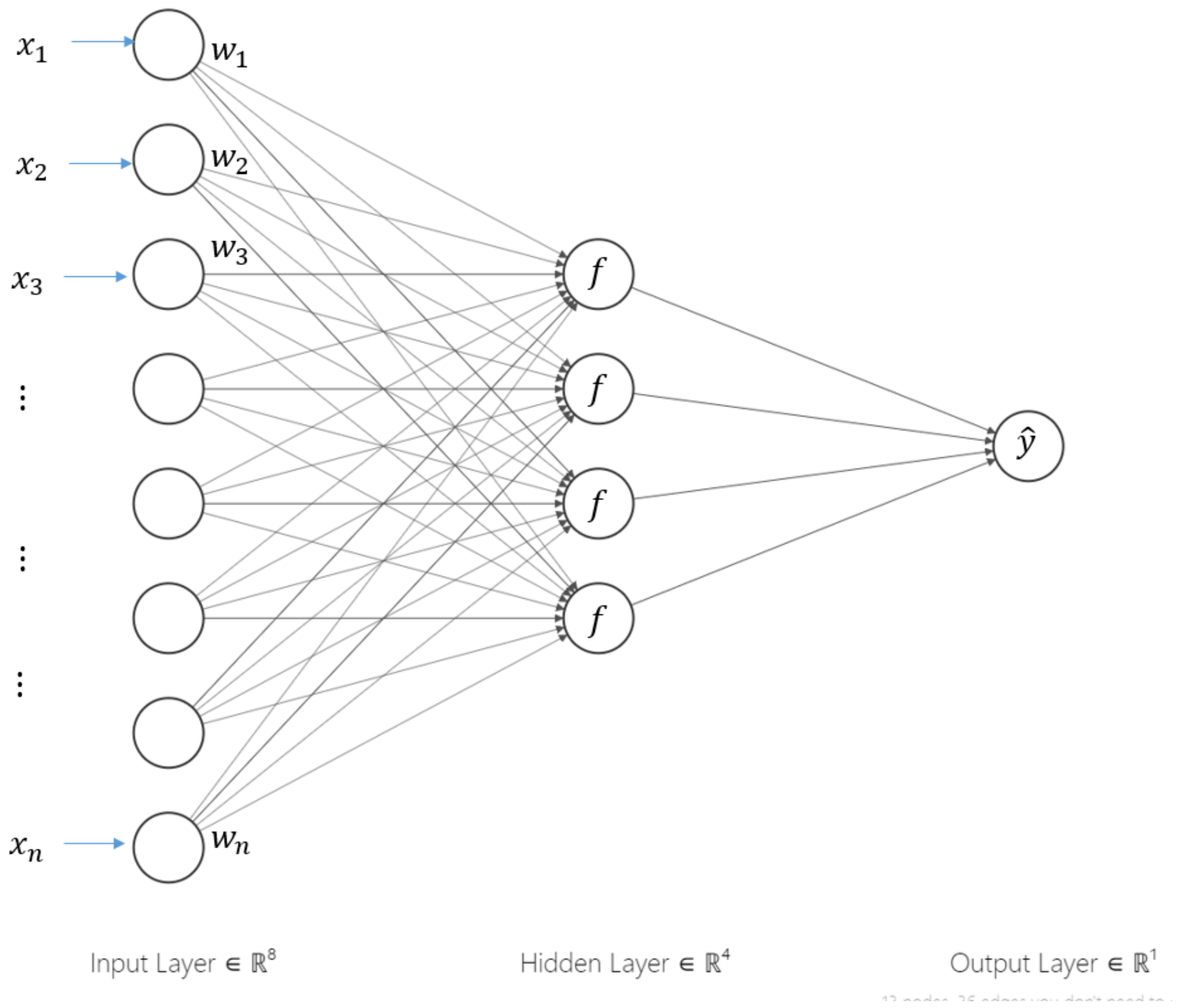

x_1
w_1
x_2
w_2
x_3
w_3
x_n
w_n
f
f
f
f
$\hat{y}$
Input Layer $\in \mathbb{R}^8$
Hidden Layer $\in \mathbb{R}^4$
Output Layer $\in \mathbb{R}^1$

Figure 2.5: An example of a basic feedforward NN.

ANNs can be trained through a backpropagation algorithm, which is an efficient method of computing gradients via a chain rule of derivatives. Once gradients are computed, an optimization algorithm applies the gradients to train the model. In this thesis, Adaptive moment estimation (Adam) is the selected optimization algorithm used to train the model.

Backpropagation

This algorithm looks for the minimum of the error function in weight space using the method of gradient descents; the combination of weights that minimizes the error function is considered a solution and is guaranteed to converge to a local optima [82]. As backpropagation takes derivative at each iteration step, the activation function used must be differentiable, and thus, step functions are not applicable, as the composite function produced by the nodes would be discontinuous, and as a result, the error function would be too [82]. A general error function that can be minimized is defined in (2.12). The backpropagation algorithm is applied by iterating over the set of observed pairs, and the general algorithm can be summarized in several discrete steps as follows [83]:

- Feed-Forward: The input vector X is fed into the input layer, and forward propagated through the network. At each step, the output of each neuron o_i is calculated according to (2.31).
- Error at the output layer: At the output layer, the error δ_j, where j is the neuron in the output layer, is calculated as follows:

  $$\delta_j o_i = \frac{\partial E}{\partial w_{ij}} \tag{2.33}$$

  where w_{ij} is the weight from the i_{th} neuron into the j_{th} neuron
- Backpropagation to the hidden layers: The error is backpropagated from the output layer to the first layer. If w_{hq} is the connection from the h_{th} to q_{th} neuron in the hidden layer, then the error is estimated as follows:

  $$\delta_h = \sum_q w_{hq}\delta_q \tag{2.34}$$
- Weight updates: At the end of each backpropagation iteration, the weights are updated as follows:

  $$w_{ij} = w_{ij} - \alpha\delta_j o_i \tag{2.35}$$

  where α is the learning rate that controls the step of the gradient descent.

The algorithm iterations are terminated when the value of the error function has become sufficiently small, or if early stopping is invoked.

Optimizer

Once the backpropagation computes gradients, the optimizer uses these gradients to train the model. The choice of an optimizer is a hyperparameter that needs to be tuned for a specific task; the Adam optimizer introduced in 2014 in [84] is employed here. Adam is an adaptive learning rate method with an ability to compute individual learning rates based on the individual features in the dataset. The learning rate is adapted for each weight, based on the estimations of the first and second moments of the calculated gradient. In addition to storing an exponentially decaying average of past squared gradients v_t, Adam also keeps an exponentially decaying average of past gradients m_t, as follows:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{2.36}$$

where m_t and v_t are estimates of the first moment (mean) and the second moment (the uncentered variance) of the gradients, respectively; g_t is the gradient of the current mini-batch; β_1 is the exponential decay rate for the first moment estimates; and β_2 is the exponential decay rate for the second-moment estimates. If m_t and v_t are initialized as null vectors, they are based towards zero; this limitation can be addressed by computing bias-corrected first and second moment estimates:

$$\begin{aligned} \hat{m_t} &= \frac{m_t}{1-\beta_1^t} \\ \hat{v_t} &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{2.37}$$

which are further utilized to produce the following Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v_t}} + \epsilon} \hat{m_t} \tag{2.38}$$

where ϵ is a number that is small enough to prevent division by zero in the implementation.

The amount α that the weights are updated during training is referred to as the step size or the learning rate, which controls the rate or speed at which the model learns. Large learning rates help to regularize the training, but if the learning rate is too large, it results in weight updates that are too large, with the performance of the model oscillating over training epochs or causing the weights to explode [45]. On the other hand, a learning rate that is too small may never converge or may get stuck on a suboptimal solution [45]. In this work, instead of treating learning rates as a hyperparameter, it is a value that changes according to the decaying learning rate method [45], which reduces the learning rate by a magnitude of 0.10 when the performance of the model plateaus.

Activation Functions (AF)

As mentioned, backpropagation takes a derivative of the error function at each iteration step, which implies that the activation function must be differentiable. Linear activations functions are rarely used in the hidden layers, since the model would behave as a linear regression model. The non-linear activation functions used in the thesis are reviewed next.

Sigmoid: This is referred to as a logistic function and is shown in Figure 2.6. Mathematically, it can be expressed as:

$$\sigma(x) = \frac{1}{1 - e^{-x}} \tag{2.39}$$

This function takes a real-valued number and compresses it into a range between 0 and 1, i.e., $\sigma(x) \in (0, 1)$. In this work, the sigmoid is used as an activation function in the output layer. As the activation function in the hidden layers, sigmoid possess two major drawbacks: is saturates and kill gradients, and the outputs are not zero-centered, which leads to large oscillations in the gradients updates [85].

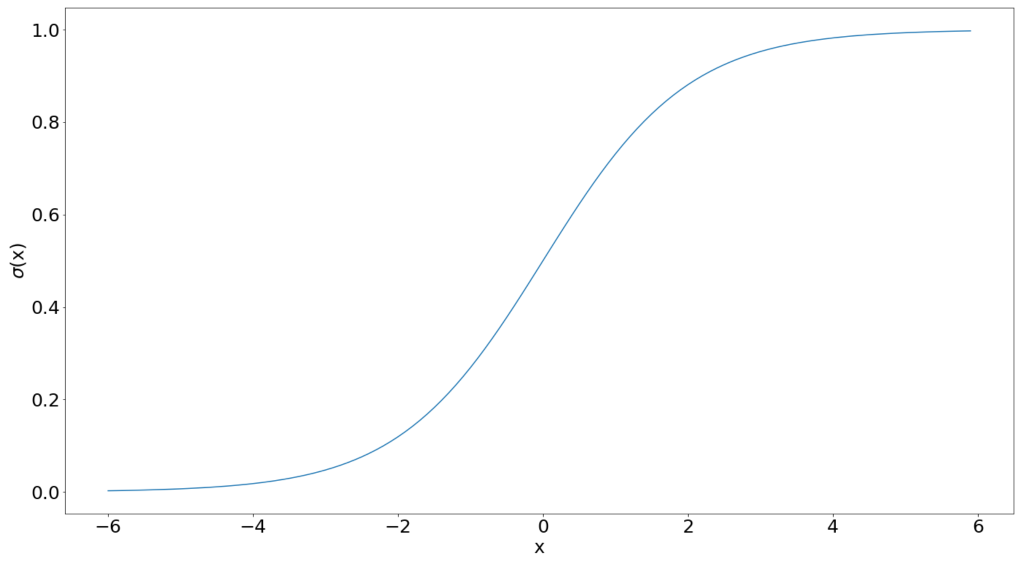


Figure 2.6: Sigmoid activation function.

Rectified Linear Unit (ReLU): In this work, the activation functions used in the hidden layers are ReLUs are proposed in 2010 [86], and are illustrated in Figure 2.7. This is the most widely used activation function for deep learning applications, and is a faster learning activation function, offering better performance and generalization compared to other activation functions [87]. The ReLU represents a nearly linear function and therefore, preserves the properties of linear models that made them easy to optimize with gradient-descent methods [45].

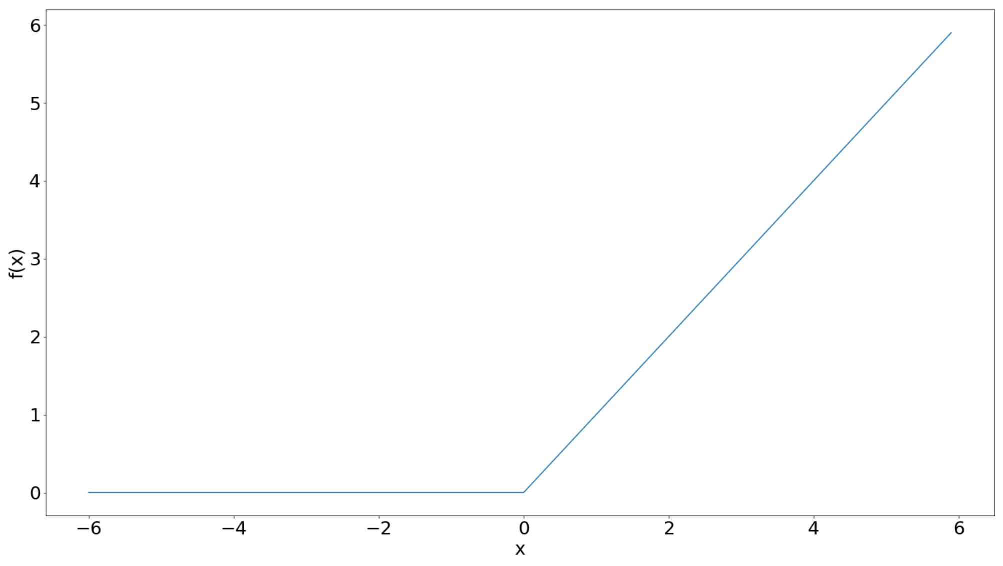


Figure 2.7: ReLU activation function.

The ReLU activation function performs a threshold operation for each input element, where values less than zero are set to zero, thus the ReLU is given by:

$$f(x) = \max(0, x) = \begin{cases} x_i, \text{ if } x_i \geq 0 \\ 0, \text{ if } x_i < 0 \end{cases}$$

This function "rectifies" the values of inputs less than zero, thereby forcing them to zero and eliminating the vanishing gradient problem observed in earlier types of activation functions. The main advantage of using ReLU is that it guarantees faster computation, since it does not require exponential and divisions, speeding up computations [85]. However, the function can easily overfit compared to the sigmoid function, which can be addressed using the dropout technique to reduce the effect of overfitting [88]. Another drawback from the simplicity of the gradient updates 0 and 1 is that it can lead to neurons "dying" during training; to avoid this problem, Leaky ReLUs have been proposed [89].

Epoch, Batch Size, and Dropout

Another set of hyperparameters that affects performance of an ANN is batch size and number of epochs. The number of epochs, which is comprised of one or more batches, is a hyperparameter that defines the number of times that the learning algorithm will propagate through the entire training dataset [90]. Running a model for too many epochs will inevitably lead to overfitting; in order to address this limitation, a regularization technique known as early stopping is used. Early stopping halts the learning process if the error on the validation set is not minimizing for a set number of epochs [91].

Batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters [90]. There are 3 major types of batch sizes, with the mini-batch gradient descent being employed here. A mini-batch is characterized by the size of the batch larger than 1 sample, but less than the whole training sample. The size of the batch is tuned during the learning process.

In addition to early stopping, another technique used in this thesis to prevent overfitting is dropout. The algorithm is named due to various random subsets of neurons being deactivated during the training [92]. Each neuron is kept with a fixed probability p, where $p = 0.5$ yields the maximum regularization and is close to optimal for a wide range of applications [92].

Number of Hidden layers and Neurons

Another important hyperparameter that is crucial to performance of ANNs is the number of hidden layers and number of neurons in each layer. Theoretically, the universal approximation theorem states that a single hidden layer network can approximate any continuous functions for inputs within a Euclidean space [93]. In [94], the author indicates that without any hidden layers, an ANN will be capable of representing linearly separable functions only. Given 1 hidden layer, an ANN can approximate any function that contains a continuous mapping from one finite space to another. On the other hand, an ANN with 2 hidden layers can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions, and can approximate any smooth mapping to any accuracy. Any ANN with more than 2 layers can learn complex representations for layered layers.

The number of hidden layers is a hyperparameter that needs to be tuned first based on the problem; in this work, 1 hidden layer is used. The second step is then to decide the number of neurons in the hidden layer. Using too few neurons results in underfitting, as the model is not complex enough to learn a pattern. On other hand, using too many neurons results in overfitting, as the model will memorize samples. Furthermore, choosing a large number of neurons will increase the training time [94].

2.3.4 Long Short Term Memory (LSTM) Network

Recurrent Neural Networks (RNN) were proposed in 1986 as a way to handle sequential data, which are distinguished by the importance of the order of the information [95]. The fundamental feature of an RNN is the ability to unfold into a chain of feed-forward neural networks, where each layer corresponds to one sequence, as it can be seen in Figure 2.8.

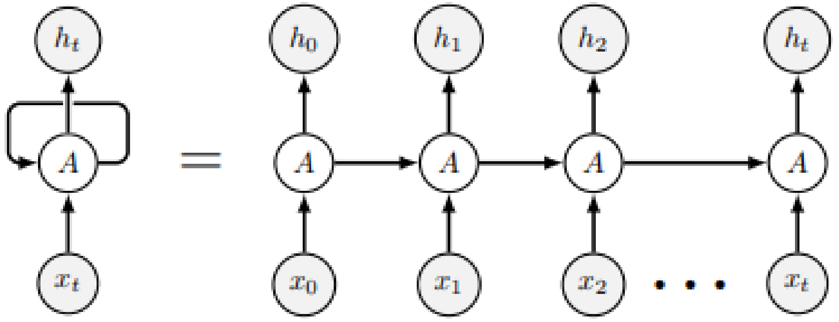


Figure 2.8: Unfolding of RNNs.

Unfolding an RNN results in the parameters being shared across the deep network structure, which performs well for various data sequence lengths [45]. Since RNNs are unfolded into a series of feed-forward neural networks, backpropagation could be a proper training choice; however, due to RNNs being very sensitive to any changes in the parameters, regular backpropagation leads to a vanishing gradient problem [96]. The solution to this is using LSTM networks, which is one of the main models used in this work [97].

Basic LSTM

The core idea behind LSTMs is that at each time step, a few gates are used to control the passing of information along the sequences that can capture long-range dependencies more accurately [98]. The internal state of the LSTM is controlled by the Constant Error Carousel (CEC) [99]. The internal gates control the flow of the information, and thus eliminate the problem of vanishing gradients. The name *gate* is derived from the fact that activation functions decide how much infomation can be let through, where 1 indicates that all information goes through and 0 cuts off the flow from another node. In the architecture presented in Figure 2.9, forget gate f^t solves the problem with long-term dependency that arises if sequential information was not broken into distinct sequences a priori. At each time step t, the hidden state h^t is updated by current data at the same time step x^t, the hidden state of the previous time step h^{t-1}, the input gate i^t, the forget gate f^t, the output gate o^t, and a memory cell c^t [100]. The following updating equations represent this process:

$$i^t = \sigma(W^i x^t + V^i h^{t-1} + b^i) \tag{2.40}$$

$$f^t = \sigma(W^f x^t + V^f h^{t-1} + b^f) \tag{2.41}$$

$$o^t = \sigma(W^o x^t + V^o h^{t-1} + b^o) \tag{2.42}$$

$$\tilde{C}_t = \tanh(W^{\tilde{C}} x^t + V^{\tilde{C}} h^{t-1} + b^{\tilde{C})} \tag{2.43}$$

$$c^t = f^t \odot c^{t-1} + i^t \odot \tilde{C}^{t-1} \tag{2.44}$$

$$h^t = o^t \odot \tanh(c^t) \tag{2.45}$$

where model parameters $W^f, W^i, W^o, W^{\tilde{C}}$ are input weights; V^f, V^i, V^o, and $V^{\tilde{C}}$ are recurrent weights, b^f, b^i, b^o, and $b^{\tilde{C}}$ are bias weights to be learned during the training stage; σ is the sigmoid activation function; and $\odot$ is the element-wise product.

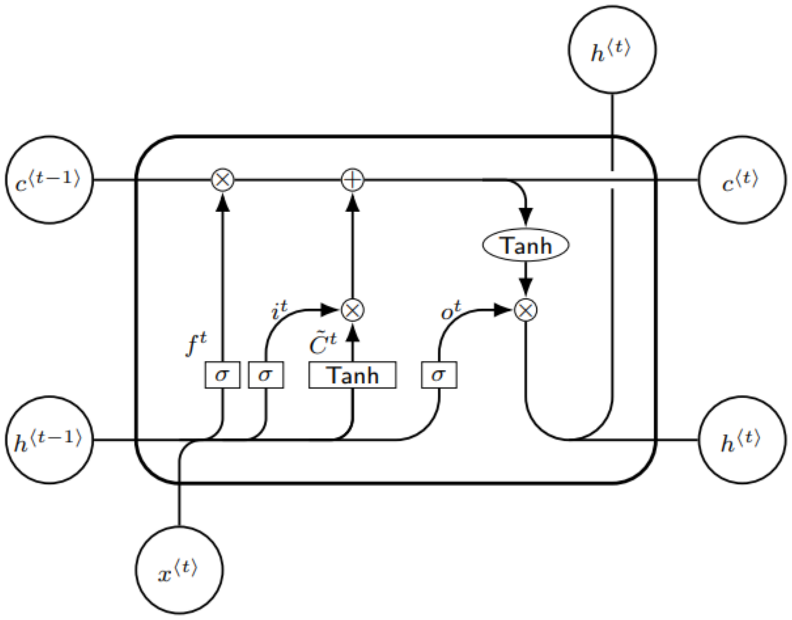


Figure 2.9: Schematic graph of LSTM cell at time t.

2.3.5 Ensemble

Ensemble systems are characterized by a set of classifiers whose individual decisions combined in some way to classify new samples. There are several approaches to ensemble learning such as dividing a training set, (*bagging*), manipulating data distribution (*boosting*), input features (*Random Forest*), and learning algorithms [101]. In [102], the author states that ensemble models generate more accurate classifications due to 3 major reasons: statistical, computational, and representational.

- Statistical: As the goal of the learning algorithm is to search the chosen space of hypothesis in order to find the hypothesis that best approximates the target function [103]. However, if the training set is much smaller compared to the hypothesis space, then statistical problems emerge. The learning algorithm trained on small training data finds the solution in the hypothesis space, producing equal results on the training set; however, the performance on the validation set varies. By constructing ensemble of learning algorithms run on the same training set, the risk of picking the wrong model is reduced [104].

- Computational: As previously discussed, the backpropagation algorithm converges to the local minima only. Having a large dataset and complex pattern in the data increases the computational burden of finding the best hypothesis. Ensemble learning with different starting points may result in a better estimation of the true hypothesis, compared to an individual classifier [105].
- Representational: This is based on the idea that there might not be a true hypothesis h^* in the hypothesis space. By combining several models from the hypothesis space, h^* can be approximated; for example, an ensemble of linear models can be employed to approximate a nonlinear classification boundary [105].

In this work, the following ensemble methods are used: majority and weighted majority voting, and Adaboost. The methods are explained next.

Majority and Weighted Majority Voting

In voting ensemble approaches, each classifier classifies an unseen sample based on its evaluation; the final class prediction is the one with the majority of votes [102]. The majority vote function can be defined as follows:

$$[d_{i,1}, \dots, d_{i,M}] \in \{0,1\}^M \quad i = \{1, \dots, B\}$$

where the outputs of the ensemble's classifier, $d_{i,M}$ are expressed through a binary vector of the size M, with M being the number of possible classes and B being the number of classifiers in the ensemble. If the i^{th} classifier votes the class C_j for the new sample, then $d_{i,j} = 1$; otherwise, it is 0. Thus, the majority voting ensemble predicts the class C_k as follows:

$$\sum_{i=1}^{B} d_{i,k} = \max_{j=1,\dots,M} \sum_{i=1}^{B} d_{i,j} \tag{2.46}$$

In (2.46), all classifiers contribute equally to the final predictions; this can be modified, so that each classifier is assigned a predetermined weight, which is known as weighted majority voting and is calculated as follows:

$$\sum_{i=1}^{B} w_i d_{i,k} = \max_{j=1,\dots,M} \sum_{i=1}^{B} w_i d_{i,j} \tag{2.47}$$

where the vector w_i represents the weight for the i^{th} classifier.

AdaBoost

The ADAptive BOOSTing (AdaBoost) algorithm is based on the idea of boosting [106], which consists of assigning a weight to each training sample, adaptively changing the weight at each boosting round [107]. Boosting algorithms place more weights on samples most often misclassified by the previous classifier, so that the next round of learning focuses on them [107]. AdaBoost is based on the idea of boosting with the sole aim of converting a set of weak classifiers, defined as classifiers with the performance only slightly better than a random guessing, into strong ones, and gets its name from adaptively configuring classifiers in favor of samples missclassified by previous ones.

In [108], the author formalizes the algorithm as follows: AdaBoost takes as input a training set $\{(x_i, y_i)\}_{i=1}^{N}$, where $x_i \in \mathbb{R}^D$ and $y_i \in \{-1, +1\}$. Based on a weak classifier, $G_m(x) \in \{-1, +1\}$, along with the following 0-1 loss function I [109]:

$$I(G_m(x), y) = \begin{cases} 0 & \text{if } f_m(x_i) = y_i \\ 1 & \text{if } f_m(x_i) \neq y_i \end{cases} \tag{2.48}$$

the pseudo-code for the AdaBoost algorithm can be defined as follows [108]:

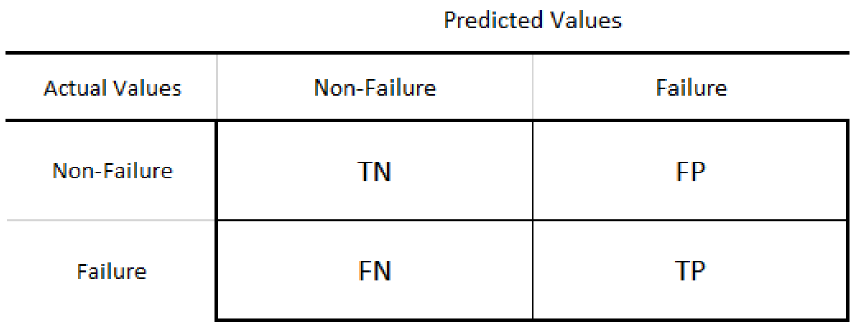
Algorithm 1 AdaBoost Algorithm

```
1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \ldots, N$
2. For $m = 1$ to $M$:
    (a) Fit a weak classifier $G_m(x)$ to the training data using weights $w_i$.
    (b) Compute $\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$.
    (c) Compute $\alpha_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m)$.
    (d) Set $w_i \leftarrow w_i \exp[\alpha_m I(y_i \neq G_m(x_i))]$, $i = 1, 2, \ldots, N$.
3. Output final classifier $G(x) = sign(\sum_{m=1}^{M} \alpha_m G_m(x))$.
```

2.3.6 Evaluation Metrics

The described algorithms are trained and applied to the test set, after which step they are evaluated according to 3 criteria: precision, recall, and F_1 score. The aforementioned metrics are calculated based on values in the confusion matrix depicted in Figure 2.10, which is not a performance measure as such but rather, a summary of prediction results on a classification problem. Thus, the diagonal cells show the number of points for which the true label is the same as the approximated labels, while the off-diagonal ones show the ones that are mislabelled by the trained classifier. In Fig 2.10, the abbreviations represent the following:

- True Positive (TP): Number of failures correctly predicted.
- False Positive (FP): Number of failures predicted that do not occur;
- True Negative (TN): Number of no failures predicted when none occur.
- False Negative (FN): Number of no failures predicted but they occur.



	Predicted Values	
Actual Values	Non-Failure	Failure
Non-Failure	TN	FP
Failure	FN	TP

Figure 2.10: Confusion Matrix.

The recall, precision, and F_1 score are then defined as follows:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.49}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.50}$$

$$F_1 = \frac{2TP}{2TP + FP + FN} \tag{2.51}$$

From these metrics, it is possible to conclude that Recall is interpreted as the percentage of actual failures that are correctly predicted, and Precision is the percentage of predicted failures that are actually true.

In fault prediction, the cost of misclassifying a faulty sample as normal is very high; hence, it is important to maximize the Recall metric. Furthermore, even though the Accuracy metric is commonly used in classification problems [110], this will not be used here, since a majority class would have a significant effect on the average performance.

2.4 Feature Selection

As irrelevant features can lead to both overfitting and increase in the computational cost, a feature selection procedure is important in machine learning [111]. Feature selection methods are categorized under one of the following categories: filter, wrapper, and embedded methods [112]. In this work, the algorithms used for the feature selection procedure are Extra Trees Classifier (ETC) and Partial AutoCorrelation Function (PACF).

2.4.1 Extra Trees Classifiers

ETC, also known as extremely randomized trees, are introduced in [113], where experiments on datasets show that this method has demonstrates similar results as a Random Forest approach, while saving computational time and being simple to implement. The Extremely Randomized Trees algorithm is an embedded method that is based on learning which features best contribute to the prediction power of the model while training the actual model. ETC aggregate the results of multiple decision trees collected in a forest to output the final classification result; by aggregating the multiple decision trees, a problem of overfitting is solved. In this algorithm an ensemble of decision trees is constructed from the whole dataset. At each test node, each tree is provided with random samples from the feature set in order to minimize a Gini index [114]. During the minimization of this index, nodes that possess the lowest Gini values are at the start of the trees, and nodes with the largest Gini values are at the end. By pruning trees, according to pre-set threshold, the features that contribute the most to the target prediction are identified.

2.4.2 Partial Autocorrelation Function

As it has been mentioned, in this work a lagged window is constructed that corresponds to the amount of data history used to allow the model to make a prediction. However, not all of the lagged variables allow predicting the target, and as the result, a feature selection on the lagged variables is needed. A comprehensive research on feature selection for lagged variables are presented in [115]; a popular technique used in this works is the PACF [116].

PACF can be considered a filter method that is independent of any machine learning algorithms, and is based on the scores of various statistical tests. PACF at lag k is defined as the correlation that results after removing the effect of any correlations due to the short term lags [116]. In this work, a "significance" level of 5% is employed, and those lags that are outside of the upper and lower limits, calculated as $\pm 1.96/\sqrt{X}$, where X is the vector length, are kept. A sample PACF is shown in Figure 2.11, using data from [117] where in the feature selection process, the lags that are inside of the bound are filtered out, resulting in a reduced size of the lag window.

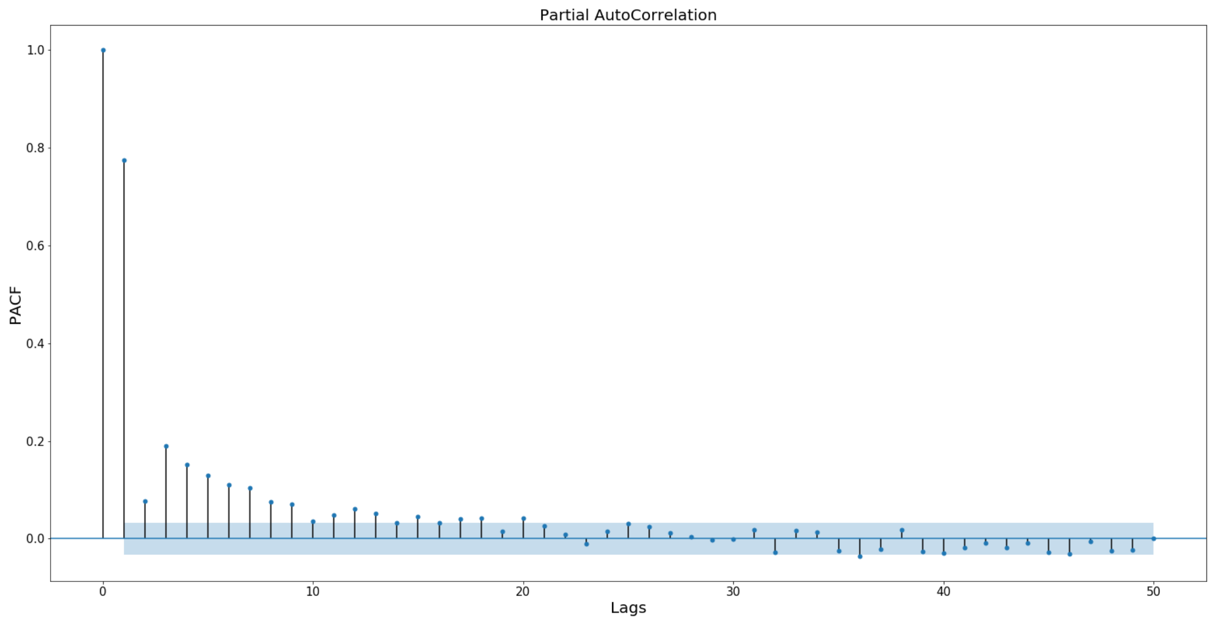


Figure 2.11: A sample PACF.

2.5 Summary

In this chapter, the background of the main concepts required to develop a predictive model is presented. First, time series analysis along with common tests are reviewed. This is followed by general data preprocessing steps, including the data imputation technique used in this thesis. Finally, machine learning is introduced, where concepts as overfitting and underfitting are discussed, along with the models and techniques used in this work to build a predictive framework, i.e., ANN, LSTM, SVM, ensemble, AdaBoost, and PACF.

Chapter 3

Architecture of the Proposed Predictive Maintenance Model

This chapter presents the case study and the results of the implementation of the failure prediction model, using algorithms presented in Chapter 2. The case study is developed to test the predictive power of several models using the collected performance data from the City of Summerside wind farm. The preprocessing steps are explained and results are presented, discussing the selection of the most adequate final model.

3.1 Fault Prediction Framework

The fault prediction framework is shown in Figure 3.1, and consists of:

- The input is historical data collected from the SCADA system of the wind farm installed in Prince Edward Island (PEI), Canada.
- Define a target variable, based on the collected data.
- Perform initial tests on the collected time series dataset to determine its stationarity.
- Create a lagged window that will be used as input variables for model training.
- Perform a feature selection.
- Tune parameters by randomly searching for combinations of parameters in a parameter grid.

The steps are discussed in details next.

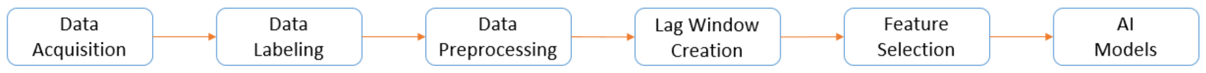


Figure 3.1: Fault prediction framework.

The models developed in this thesis are implemented in Python using the following package versions: `Python 3.6.6` [118], `Tensorflow 1.10` [119], `Keras 2.2.4` [120], `scikit-learn 0.21.2` [72], `MatPlotLib 2.2.3` [121], `NumPy 1.16.4` [122], `Pandas 0.22.0` [123].

3.2 Industrial Data Description

The data in this study are obtained from the City of Summerside wind farm, operating in PEI, Canada. Since 2009, the facility installed 4 Vestas 3.0 MW V-90 wind turbines, which is meeting 25% of the 137.5 TWh demand; the rest was imported from the NB Power [124]. The data consist of operational and status data, covering a period of 12 months; from August 2017 until August 2018.

3.2.1 Operational Dataset

The operational SCADA contains data recorded with 10-minutes granularity from 4 turbines, with 101 features. The sensitive information pertaining to measurements of a turbine are not shown to maintain confidentiality; however, the name of feature variables are shown in Table 3.1. The information included consists of records for environmental conditions, such as wind speed that provide information about the context in which the equipment was working at that specific time; measurements for wind turbine components such as average Rotations Per Minute (RPM) of a generator; and active and reactive power generated and related electrical variables such as voltages, currents, and frequency.

Table 3.1: Dataset features.

Generator Phase1 Temp. Avg.
Generator Phase2 Temp. Avg.
Generator Phase3 Temp. Avg.
Generator RPM Avg.
Generator RPM StdDev.
Ambient Temp. Avg.
Ambient WindSpeed Avg.
Ambient WindSpeed StdDev.
Grid Production CurrentPhase1 Avg.
Grid Production CurrentPhase2 Avg.
Grid Production CurrentPhase3 Avg.
Grid Production Frequency Avg.
Grid Production PossibleCapacitive Avg.
Grid Production PossibleCapacitive StdDev.
Grid Production PossibleInductive Avg.
Grid Production PossibleInductive StdDev.
Grid Production PossiblePower Avg.
Grid Production PossiblePower StdDev.
Grid Production Power Avg.
Grid Production Power StdDev.
Grid Production ReactivePower Avg.
Grid Production ReactivePower StdDev.
Grid Production VoltagePhase1 Avg.
Grid Production VoltagePhase2 Avg.
Grid Production VoltagePhase3 Avg.

Alarms Dataset

Along with the operational dataset, a different dataset that captures any changes in a wind farm operation state was provided. It is assumed that the turbine is operating in normal conditions until a new status message is generated. The status dataset comes with several indicators that are summarized in subcategories, presented in Table 3.2, where:

- *Turbine* indicates a number of turbines.

- *Category* indicates the possible reason behind an abnormal behaviour and includes the following categories: Environmental, Force Majeure, Manufacturer, Owner, Unscheduled maintenance, Utility.
- *Duration* indicates the duration of a fault or a maintenance procedure.
- *Event* corresponds to the code that helps to distinct between a fault or scheduled service, and is supported by the *Event Text* that clarifies the code in the *Event* category.
- *Comment* indicates the possible reason behind the unscheduled maintenance.
- *Responsible Source Title* has a single entry that is *'Remote pause by Owner'*.
- *Time from* and *Time to* indicate the starting and ending periods of a fault or a service
- *Total kWh* that can potentially indicate the total generated power, however most of the variables are 0.
- *Lost kWh* indicates the lost power due to an event.

Out of these categories, *Event*, *Turbine*, *Time from*, and *Time to* are the most informative and will be used in order to construct a dataset, representing abnormal behaviour.

Table 3.2: Categories in the alarms dataset.

Category
Comment
Duration
Event
Event text
Lost kWh
Parameter1
Parameter2
ResponsibleSourceTitle
Time from
Time to
Total
Total kWh
Turbine

Missing Data

The original operation dataset has 20165 samples that are missing at random for the whole wind farm; thus, missing data for individual turbines is less. Even though dropping data rows with missing values is an option, due to the high number of missing samples, it has been decided to impute them using a MICE technique, as explained in Section 2.2.1, via the `FancyImpute` package v0.3.2 [125].

3.3 Data Labelling

It order to properly train a classifier, it is important to ensure that the data is correctly labelled; otherwise, a problem known as "noisy labels" arises and currently, it is an active research area [126], [127]. In this work, data labeled as "faulty" or "normal", where samples are classified as faulty or fault free, respectively. In order to classify a faulty operation, it is also necessary to develop a set of labelled fault data. As previously mentioned, status messages with codes corresponding to the faults are selected. Due to the time and dates in the collected dataset being in different formats, a common format was adopted to ensure a consistent analysis. A time band of 10 minutes before the start and after the end for these turbine states is used to match up the associated 10-minute operational data. The 10-minute time-band is selected to capture any 10-minute period where a fault occurred; for example, for a fault 188 occuring between 10:19 and 12:19, the aforementioned approach would label the samples between 10:10 and 12:20 are labelled as faults.

It is important to mention that the goal of this work is to predict faults associated with the wind turbine system; thus, the faults that correspond to categories *Environmental* and *Force Majeure* have been dropped. Based on that information, a new feature is created: *No-Fault*, where samples are labelled as 0, if they correspond to the normal operation, or 1, if they correspond a fault, based on the *Event* code. By doing so, the problem is framed as a binary classification task.

3.3.1 Fault Dataset Generation

As it has been discussed, events code can correspond to either a scheduled maintenance, or an abnormal behaviour. While the *Event Text* category is available, the status messages are cryptic and cannot help with learning the true nature of the event. Thus, the official error code from the turbines manufacturer, Vestas, is used [128]. Hence, the faults are divided according to the codes presented in the manual, as shown in Tables 3.3-3.5, along with the statuses, presented in Table 3.6.

Table 3.3: Electrical fault codes.

Code	**Electrical**
188	External Power Supply eq.
176	Error on all wind Sensors
315	ExEx Low Voltage (Grid)
485	Circuit Breaker Open
286	Extreme High Current
707	Chopper Hardware error
959	Heating Slipring in stop
958	Heating Slipring in pause
817	Q8 close not possible
187	Q9 open
489	Timeout generator reconnecting
632	Signal error (Failing signal is listed as CAN-node number)
102	Emergency circuit open
648	Arc Detected, F60 tripped (Environmental)
687	Busbar temperature high
935	Grid Voltage above stop limit (125%)
332	Low DC voltage (Turbine)
846	Frequency error (44.43 Hz)

Table 3.4: Mechanical fault codes.

Code	**Mechanical**
604	Low Pressure in Pitch Block C
603	Low Pressure in Pitch Block B
958	Heating Slipring in Pause
154	Max Rotor RPM
162	Pitch doesnt follow reference
165	Low Oil Level, pitch hydraulic
512	Blade 1,3 could not be released
948	Hub Shut off, valves not open
151	High temperature. Generator bearing
163	Low pitch hydr. Press
296	Tower acceleration X. Alarm
734	Pitch Deviation AB
735	Pitch Deviation AC
844	Gear oil low pressure
191	Related to blade A not moving at the expected speed
560	Low water level
297	Tower Acceleration Y. Alarm
504	Defect lubrication system blade C

Table 3.5: Other Faults fault codes.

Code	**Control**
356	Extreme wind direction
144	High windspeed
100	Too many autorestrats

Table 3.6: Status Codes.

Code	**Status**
220	New Service State
276	Start auto-outyawing
224	Pause
222	Emergency
309	Pause over RCS
223	Stop

3.4 Data Preprocessing

3.4.1 Feature Selection

As not all of the features of the original data contribute to the selection of the target, the ETC algorithm, described in Section 2.4.1 is used, with the parameters in Table 3.7. The result of this algorithm can be seen in Figure 3.2.

Table 3.7: Parameters for ETC, used in *Feature Selection*.

Parameters	**Values**
number of estimators	100
criterion	Gini
max depth	None
min samples split	2
min samples leaf	1
min weight fraction leaf	0
max features	auto
max leaf nodes	None
min impurity decrease	0
min impurity split	1.00E-07
bootstrap	False
oob score	False
random state	42

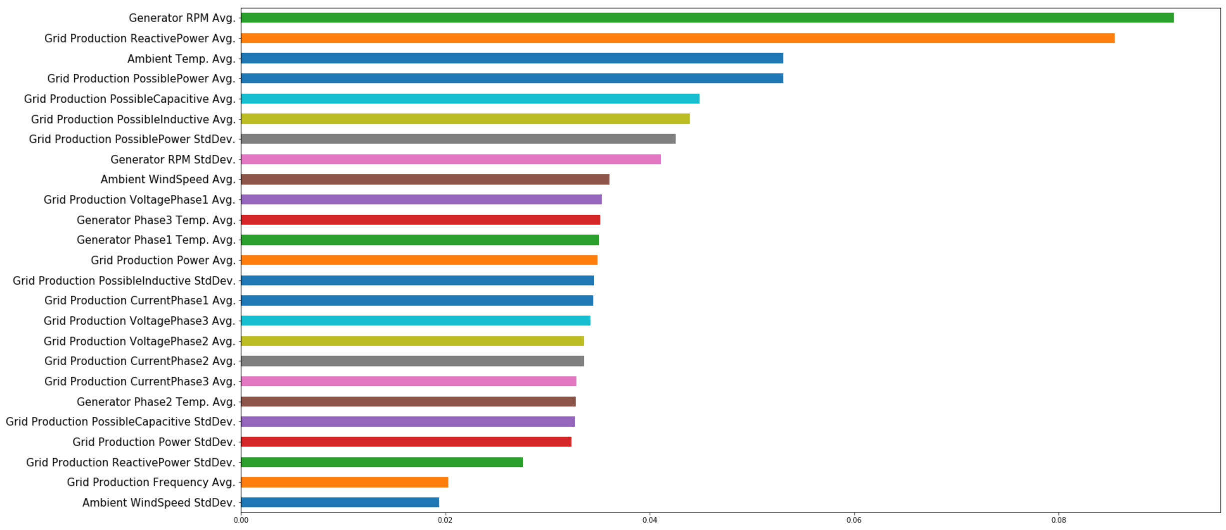


Figure 3.2: Contribution of individual features to the final target.

Based on the output of the feature selection algorithm, the threshold is set up to 0.03. As the result, the original dataset is reduced from 26 features to 23.

3.4.2 Time Series Analysis

As discussed in Section 2.1.3, to develop a proper forecasting model, time series are expected to be stationary. The ADF and KPSS are applied here, and appropriate actions are taken based on the outcome of those statistical tests. Table 3.8 shows the outcome of ADF test, and Table 3.9 shows the results of the KPSS test. In the statistical tests, a p-value can be used to reject or accept a null hypothesis, where $p < 0.05$ is used to reject H_0 and vice versa.

Table 3.8: Results for ADF tests.

Feature	ADF Test
Grid Production Power StdDev.	1.18E-29
Grid Production VoltagePhase2 Avg.	2.07E-30
Generator Phase2 Temp. Avg.	5.79E-29
Generator Phase3 Temp. Avg.	6.09E-29
Grid Production Power Avg.	9.47E-29
Grid Production CurrentPhase3 Avg.	9.94E-29
Grid Production PossibleInductive StdDev.	0
Grid Production PossibleCapacitive StdDev.	0
Grid Production VoltagePhase3 Avg.	0
Grid Production VoltagePhase1 Avg.	0
Generator Phase1 Temp. Avg.	5.86E-29
Grid Production CurrentPhase2 Avg.	9.69E-29
Grid Production CurrentPhase1 Avg.	9.39E-29
Ambient WindSpeed Avg.	6.92E-28
Generator RPM StdDev.	0
Grid Production PossiblePower StdDev.	5.20E-30
Grid Production PossibleCapacitive Avg.	2.62E-30
Grid Production PossibleInductive Avg.	4.56E-30
Grid Production PossiblePower Avg.	3.25E-28
Ambient Temp. Avg.	1.35E-05
Grid Production ReactivePower Avg.	0
Generator RPM Avg.	3.34E-30
Fault	1.07E-29

Table 3.9: Results for KPSS tests.

Feature	KPSS Test
Grid Production Power StdDev.	0.0941872
Grid Production VoltagePhase2 Avg.	0.0273075
Generator Phase2 Temp. Avg.	0.0414163
Generator Phase3 Temp. Avg.	0.01
Grid Production Power Avg.	0.01
Grid Production CurrentPhase3 Avg.	0.01
Grid Production PossibleInductive StdDev.	0.01
Grid Production PossibleCapacitive StdDev.	0.01
Grid Production VoltagePhase3 Avg.	0.01
Grid Production VoltagePhase1 Avg.	0.01
Generator Phase1 Temp. Avg.	0.0477435
Grid Production CurrentPhase2 Avg.	0.01
Grid Production CurrentPhase1 Avg.	0.01
Ambient WindSpeed Avg.	0.01
Generator RPM StdDev.	0.0136519
Grid Production PossiblePower StdDev.	0.0385984
Grid Production PossibleCapacitive Avg.	0.0862549
Grid Production PossibleInductive Avg.	0.0684028
Grid Production PossiblePower Avg.	0.01
Ambient Temp. Avg.	0.01
Grid Production ReactivePower Avg.	0.047052
Generator RPM Avg.	0.1
Fault	0.0426623

From Table 3.8, it follows that H_0 is rejected, resulting in the time series not having a unit root and hence, is stationary. On the other hand, from Table 3.9, it follows that in several cases $p < 0.05$ and thus, H_0 fails to be rejected, which implies that a unit root test and differencing is needed, as per (2.5). Following the differencing, the updated tables are given in Tables 3.10 and Tables 3.11, with results showing that the time series are stationary, according to both KPSS and ADF tests.

Table 3.10: Results for ADF tests after differencing.

Feature	ADF Test
Grid Production Power StdDev.	0
Grid Production VoltagePhase2 Avg.	0
Generator Phase2 Temp. Avg.	0
Generator Phase3 Temp. Avg.	0
Grid Production Power Avg.	0
Grid Production CurrentPhase3 Avg.	0
Grid Production PossibleInductive StdDev.	0
Grid Production PossibleCapacitive StdDev.	0
Grid Production VoltagePhase3 Avg.	0
Grid Production VoltagePhase1 Avg.	0
Generator Phase1 Temp. Avg.	0
Grid Production CurrentPhase2 Avg.	0
Grid Production CurrentPhase1 Avg.	0
Ambient WindSpeed Avg.	0
Generator RPM StdDev.	0
Grid Production PossiblePower StdDev.	0
Grid Production PossibleCapacitive Avg.	0
Grid Production PossibleInductive Avg.	0
Grid Production PossiblePower Avg.	0
Ambient Temp. Avg.	0
Grid Production ReactivePower Avg.	0
Generator RPM Avg.	0
Fault	1.07E-29

Table 3.11: Results for KPSS tests after differencing.

Feature	KPSS Test
Grid Production Power StdDev.	0.1
Grid Production VoltagePhase2 Avg.	0.1
Generator Phase2 Temp. Avg.	0.1
Generator Phase3 Temp. Avg.	0.1
Grid Production Power Avg.	0.1
Grid Production CurrentPhase3 Avg.	0.1
Grid Production PossibleInductive StdDev.	0.1
Grid Production PossibleCapacitive StdDev.	0.1
Grid Production VoltagePhase3 Avg.	0.1
Grid Production VoltagePhase1 Avg.	0.1
Generator Phase1 Temp. Avg.	0.1
Grid Production CurrentPhase2 Avg.	0.1
Grid Production CurrentPhase1 Avg.	0.1
Ambient WindSpeed Avg.	0.1
Generator RPM StdDev.	0.1
Grid Production PossiblePower StdDev.	0.1
Grid Production PossibleCapacitive Avg.	0.1
Grid Production PossibleInductive Avg.	0.1
Grid Production PossiblePower Avg.	0.1
Ambient Temp. Avg.	0.1
Grid Production ReactivePower Avg.	0.1
Generator RPM Avg.	0.1
Fault	0.043

3.4.3 Lag Window

The lag window approach is a common method that has been previously used in wind turbine diagnostics [129], to enable models to see enough past values relevant for future predictions. However, using too many lagged values increases the width of the sliding window and as a result, the dimensionality of the set is increased and thus, it can lead to overfitting. In this work, the lagged window size is chosen to be 12 hours and the most relevant lags, chosen via a feature selection procedure, are used as the input to the model. Thus, the stationary dataset, transformed to the lagged representation, results in increased dimensionality from 23 to 1656 features. Following the procedure set in (2.7), a prediction horizon of 1 hour is set as well.

3.4.4 Dataset Description

By repeating the steps discussed previously for 4 separate turbines, the dataset for 4 individual turbines can be obtained. Due to lack of enough faults in individual turbines, it has been decided to aggregate the 4 turbines into 1 "general" turbine, based on the assumption that all turbines have the same characteristics. Thus, the resulting dataset consists of 52482 samples and 1794 features, and out of the 1794 features, the data used for training a model consists of 1656 features, eliminating the last hour corresponding to the prediction horizon, i.e., the lead window, which are further reduced by running a feature selection approach.

3.5 Lag Feature Selection

The resulting dataset after creating a lagged representation is passed through the feature selection. The PACF for one of the features is shown in Figure 3.3. The features are selected according to the cut-off values, from the PACF plots. For $\alpha = 0.05$ and 95% confidence interval, the lags that lie above the 0.0085 threshold, and below -0.0085 are selected. Thus, the resulting matrix is reduced from 1656 to 692 features.

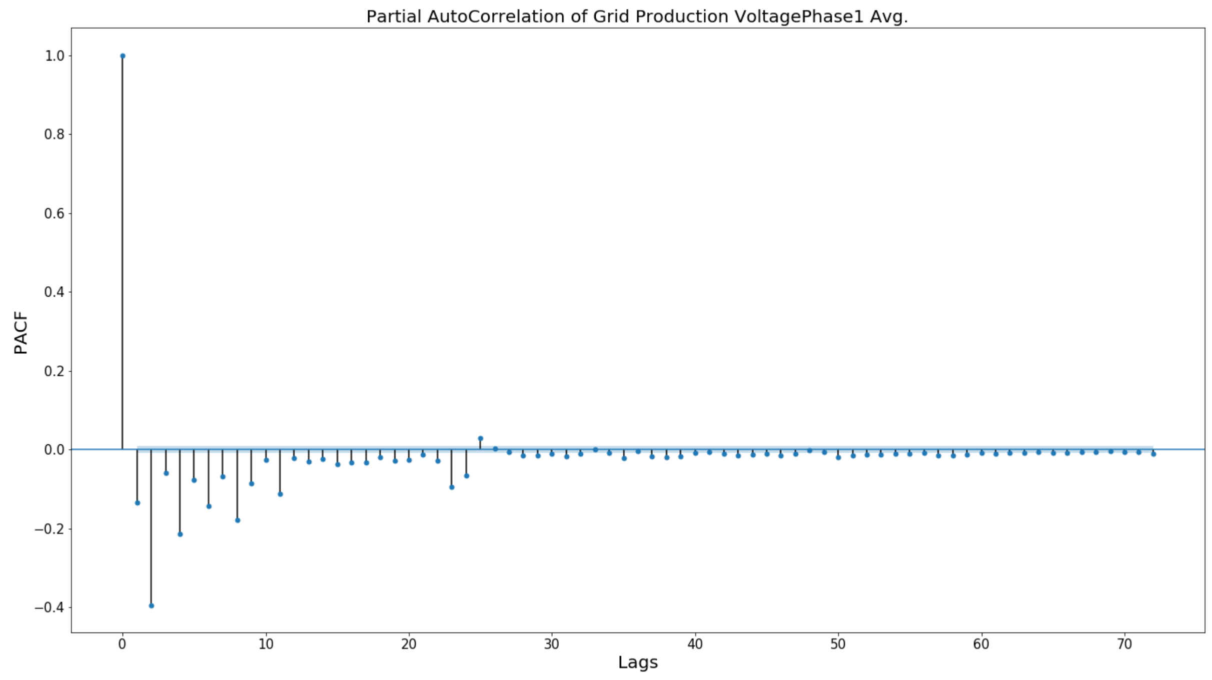


Figure 3.3: PACF of "Grid Production VoltagePhase" feature.

3.6 Machine Learning Models

The model parameters are tuned via Random Search, as explained in Section 2.3.1, with 5 splits in time-series cross validation. Prior to tuning, the data is standardized according to (2.11). In order to perform the randomized search, an object of the `RandomizedSearchCV` class from `scikit-learn` package is initialized, and the cross validation is implemented via a `TimeSeriesSplit` function in the same package. The `RandomizedSearchCV` object is used to fit the training input to the output with a list of possible hyperparameters, with each model having the set of hyperparameters to be tuned. As tuning all of the hyperparameters would take a long time to complete, the ones that have the most effect on the final results are identified and the rest are set to default values. Since, the dataset is heavily imbalanced, with 34793 samples belonging to Class 0, 1944 belonging to Class 1, the class weight approach discussed in Section 2.2.3 is utilized.

SVM (Section 2.3.2): The search grid consists of the parameters in Table 3.12, fed into `RandomizedSearchCV` class, along with the 5 splits for cross-validation. In order to get reproducible results, the `random state` parameter is set to 42, resulting in guaranteeing that same sequence of numbers are generated and thus, results are reproducible. The resulted model is saved to a file.

Table 3.12: SVM Hyperparameters.

Hyperparameters	**Values**
C	0.01, 0.1, 1, 10, 100, 1000, 10000
γ	1e-1, 1e-2, 1e-3, 1e-4, 1e-5
kernel	linear, rbf, poly

ANN (Section 2.3.3): The following parameters have been into the `RandomizedSearchCV` class, along with the 5 splits for cross-validation. Unlike SVM, in ANN another independent model is built using the selected parameters from Random Search; in this independent model, parameters are fine-tuned using a validation set. It is worth noting that the batch sizes are chosen with respect to a power of 2, as authors in [130] show that batch sizes with power of 2 offer better runtime.

Table 3.13: ANN Hyperparameters.

Hyperparameters	**Values**
Neurons in the hidden layer	15, 20, 25, 30, 35, 40, 45, 50, 55, 60
Batch size	16, 32, 64, 128
Epochs	150
Learning rate	1E-04

LSTM (Section 2.3.4): LSTM is tuned in the similar fashion as ANN, with the same parameters. The primary difference is that LSTM requires data to be in the following format: $samples, timesteps, features$. However, there is no theoretical foundations on how 2D data needs to be turned into 3D, needed for LSTM. Several experiments were carried, and the best results obtained yielded the format: $[samples, 1, features]$, as opposed to $[samples, features, 1]$. In addition, the problem of exploding gradient was observed, and thus, a setting *AMSGrad* for the Adam optimizer in the `Keras` package is set to *True*, which addresses the weight decay problem in Adam [131].

Table 3.14: LSTM Hyperparameters.

Hyperparameters	**Values**
Neurons in the hidden layer	15, 20, 25, 30, 35, 40, 45, 50, 55, 60
Batch size	16, 32, 64, 128
Epochs	150
Learning rate	1E-04
AMSGrad	True

AdaBoost (Section 2.3.5): As discussed in [132], AdaBoost does not require classes to be balanced, as AdaBoost recalculates weights for each subsequent weak learner. This results in higher weights being assigned to the minority class at each iteration. The simulation results obtained further demonstrate that the performance of the algorithm is identical when classes are balanced and unbalanced. At the same time, AdaBoost demonstrated good performance with the predefault values, and only minor manual tuning of the learning parameter was needed; hence, Random Search is not used in this case.

Table 3.15: AdaBoost Hyperparameters.

Hyperparameters	**Values**
Learning rate	1E-05
Number of estimators	100
Random state	42

Majority Voting (Section 2.3.5): There is no requirement to run Random Search on a Majority Voting algorithm, as this is a combination of several algorithms. In this case, the models are pretrained and various weights are applied to find the best combination.

3.7 Results

The available dataset to run the models is divided into training, validation, and testing sets with the following ratio: 70% for training, 15% for validation, and 15% for testing. The training set is balanced out as explained in Section 2.2.3., and thus, the objective function to be minimized is given in (2.12), except for the case of AdaBoost. The objective function for AdaBoost is (2.10). The selection of optimal parameters from the parameter grid via `RandomizedSearch` is performed on Ubuntu 16.04, GTX 1070 and 16 GB of RAM. The models are fine tuned on Windows 10, Intel® Core$^{\text{TM}}$ i7 and 16 GB of RAM.

3.7.1 SVM

The SVM model is invoked through the `scikit-learn` package, and used as the tunable model in `RandomizedSearch` with 5 cross validation time series splits. The obtained parameters are shown in Table 3.16, and the model results are given in Table 3.17, and confusion matrix is shown in Table 3.18. It is important to note that running the same model as a stand-alone will not change the final results. The SVM training took about 17 hours as the `scikit-learn` library does not support GPU hardware acceleration. The GPU-supported library, `ThunderSVM` [133], was also used for model tuning at significantly reduced computational times; however, this library is unable to save the final model, and hence it could not be used for further model processing and testing.

Table 3.16: SVM parameters from Random Search.

Parameters	**Value**
Kernel	Linear
gamma	0.1
C	1

Table 3.17: Testing dataset results for SVM model.

Label	**Precision**	**Recall**	**F_1 score**	**Data Points**
Non-Failure	0.99	0.99	0.99	7432
Failure	0.83	0.83	0.83	441

Table 3.18: Confusion matrix for SVM model.

	Predicted Non-Failure	Predicted Failure
Actual Non-Failure	7355	77
Actual Failure	77	364

3.7.2 ANN

The ANN model is trained in the similar fashion as the SVM model. The preset model parameters are shown in Table 3.19, and selected parameters returned by Random Search are shown in Table 3.20. The model results for the testing dataset are illustrated in Table 3.21.

Table 3.19: Preset hyperparameters for initial ANN model.

Preset Parameters	**Value**
Dropout	0.5
Activation Function in Hidden Layer	ReLU
Activation Function in Output Layer	Sigmoid
Epochs	150
Learning rate	1E-04

Table 3.20: Initial ANN model hyperparameters obtained with Random Search.

Parameters	**Value**
Neurons in the Hidden Layer	60
Batch size	128

Table 3.21: Testing dataset results for initial ANN model.

Label	**Precision**	**Recall**	F_1 **score**	**Data Points**
Non-Failure	0.99	0.99	0.99	7432
Failure	0.79	0.81	0.80	441

After Random Search, an improved model is built with the number of epochs set to 1000, and callbacks such as early stopping and decaying learning rate method. To fine tune the model, the validation set is used, resulting in the parameters in Table 3.22, the final results in Table 3.23, and the confusion matrix in Table 3.24. The learning curves that showing changes in learning performance over time are depicted in Figure 3.4, demonstrating how statistical fit is reached. Similar performance can be achieved with Grid Search, or with the larger parameter grid but the computational time increases.

Table 3.22: Tuned parameters for final ANN model.

Parameters	**Value**
Neurons in the Hidden Layer	50
Batch size	64
Reduced learning rate	1E-05

Table 3.23: Testing dataset results for final ANN model.

Label	**Precision**	**Recall**	F_1 **score**	**Data Points**
Non-Failure	0.99	0.99	0.99	7432
Failure	0.81	0.82	0.81	441

Table 3.24: Confusion matrix for final ANN model.

	Predicted Non-Failure	Predicted Failure
Actual Non-Failure	7347	85
Actual Failure	80	361

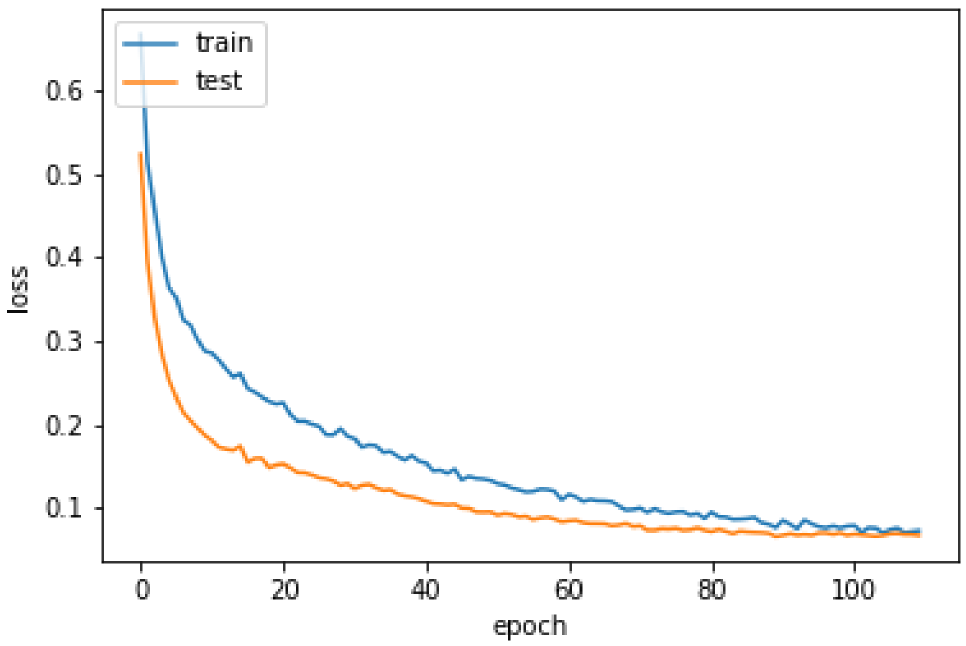


Figure 3.4: Change in the final ANN model loss versus epochs.

3.7.3 LSTM

The LSTM model is trained in similar fashion as the ANN model. The preset hyperparameters are given in Table 3.25, the obtained parameters from Random Search are in Table 3.26, and the results in Table 3.27.

Table 3.25: Preset hyperparameters for initial LSTM.

Preset Parameters	**Value**
Dropout	0.5
Activation Function in Hidden Layer	ReLU
Activation Function in Output Layer	Sigmoid
Epochs	150
Learning rate	1E-04

Table 3.26: Initial LSTM model hyperparameters obtained with Random Search.

Parameters	**Value**
Neurons in the Hidden Layer	45
Batch size	64

Table 3.27: Testing dataset results for initial LSTM model.

Label	**Precision**	**Recall**	**F_1 score**	**Data Points**
Non-Failure	0.99	0.99	0.99	7432
Failure	0.76	0.79	0.78	441

Similarly to the ANN, an improved model is built using the callbacks and validation set. The obtained parameters are shown in Table 3.28, the final results in Table 3.29, and the confusion matrix in Table 3.30. The learning curves illustrating the learning process are shown in Figure 3.5, demonstrating how statistical fit is reached.

Table 3.28: Parameters for final LSTM model.

Parameters	**Value**
Neurons in the Hidden Layer	35
Batch size	64
Reduced learning rate	1E-05

Table 3.29: Results for final LSTM model.

Label	**Precision**	**Recall**	**F_1 score**	**Data Points**
Non-Failure	0.99	0.99	0.99	7432
Failure	0.80	0.81	0.81	441

Table 3.30: Confusion Matrix for final LSTM model.

	Predicted Non-Failure	Predicted Failure
Actual Non-Failure	7345	87
Actual Failure	81	360

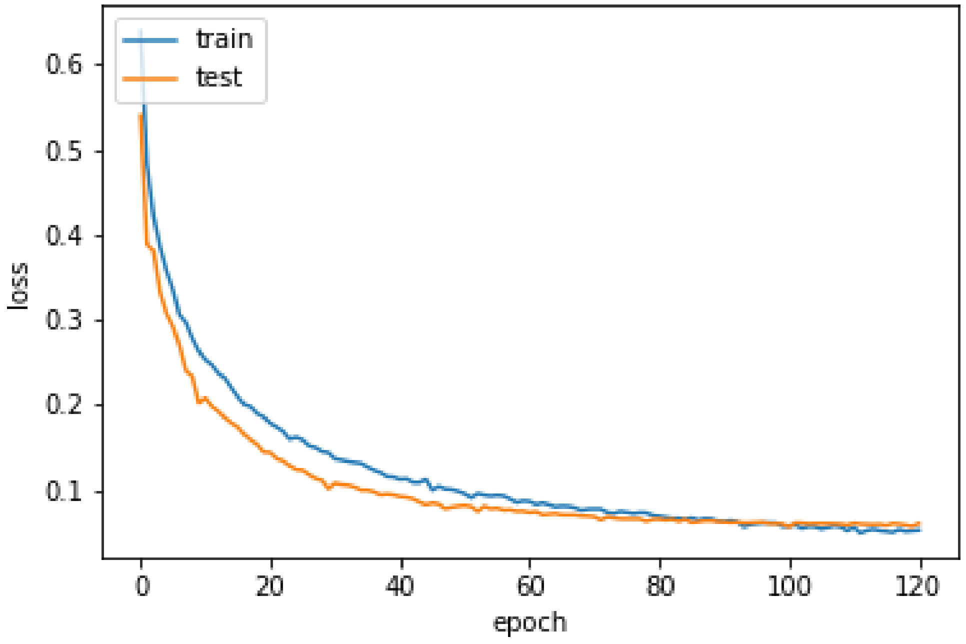


Figure 3.5: Change in the final LSTM model loss versus epochs.

3.7.4 Majority Voting and Weighted Majority Voting

As discussed in Chapter 2, the difference between majority voting and weighted majority voting is that different weight coefficients are assigned to different pretrained models in the ensemble, whereas in the case of majority voting ensemble, all models contribute to the final prediction equally. Table 3.31 shows the different weights used in simulations and Tables 3.32 to Table 3.34 show the effect of various weights. Observe that the best performance is achieved by using the weights 1 for ANN, 0.5 for LSTM, and 2 for SVM.

Table 3.31: Weights assigned to various pretrained algorithms.

Model	**Weights**		
ANN	1	1	1
LSTM	1	0.5	1
SVM	1	2	2

Table 3.32: Results with weights $[1, 1, 1]$.

Label	**Precision**	**Recall**	**F_1 score**	**Data Points**
Non-Failure	0.99	0.99	0.99	7432
Failure	0.81	0.82	0.82	441

Table 3.33: Results with weights $[1, 1, 2]$.

Label	**Precision**	**Recall**	**F_1 score**	**Data Points**
Non-Failure	0.99	0.99	0.99	7432
Failure	0.84	0.82	0.83	441

Table 3.34: Results with weights $[1, 0.5, 2]$.

Label	**Precision**	**Recall**	**F_1 score**	**Data Points**
Non-Failure	0.99	0.99	0.99	7432
Failure	0.83	0.83	0.83	441

3.7.5 AdaBoost

In the case of AdaBoost, Random Search is not needed, as good performance is achieved with the default parameters. The only parameter that needs to be tuned is the learning parameter, which is reduced to 0.00001 from the default 1; classes do not need to be balanced in this case. The results of the model are given in the Table 3.35, and the confusion matrix is given in Table 3.36. The results for the balanced dataset are shown in Table 3.37, and the confusion matrix is shown in Table 3.38. As it can be seen from the results, there is no requirement for the dataset to be balanced prior to applying AdaBoost.

Table 3.35: Results for AdaBoost model, applied to imbalanced dataset.

Label	**Precision**	**Recall**	**F_1 score**	**Data Points**
Non-Failure	0.99	0.99	0.99	7432
Failure	0.83	0.83	0.83	441

Table 3.36: Confusion matrix for AdaBoost model, applied to imbalanced dataset.

	Predicted Non-Failure	Predicted Failure
Actual Non-Failure	7355	77
Actual Failure	77	364

Table 3.37: Results for AdaBoost model, applied to balanced dataset.

Label	**Precision**	**Recall**	**F_1 score**	**Data Points**
Non-Failure	0.99	0.99	0.99	7432
Failure	0.83	0.83	0.83	441

Table 3.38: Confusion matrix for AdaBoost model, applied to balanced dataset.

	Predicted Non-Failure	Predicted Failure
Actual Non-Failure	7355	77
Actual Failure	77	364

3.7.6 Discussion

As it can be seen from the aforementioned tables, the models show similar performance in terms of precision, recall, and F_1 score. The trained models can be deployed in the real-world settings as a self-contained application, and while it is an option, there is no need to re-train the models with new incoming data. The proposed models can be used to predict from the incoming data and produce the probability of a fault occurring in an hour.

The results include prediction of both a normal and faulty conditions; the focus of this work is on predicting the faulty state 1 hour in advance. In the case of prediction of a normal operating state, the high values for precision and recall can be explained by the high number of samples. The final recall results for predicting a fault are 83% accurate, which are reasonable and in line with typical forecast scores expected from the machine learning tools studied here.

Even though all models show good performance, the final prediction methodology should include the best model. However, simplicity and interpretability of a model needs to be considered in the selection. In [134], the authors apply the Occam's razor principle, i.e., not using more than necessary, to choose the model that is simplest to interpret, explain to a client, and maintain in the long run. It is also important to add that ease model tuning is crucial during the development stage. Therefore, considering the aforementioned criteria, the AdaBoost model should be used in the proposed fault prediction method, as it is the fastest to run, and proved to be easiest to tune.

3.8 Summary

In this chapter, the algorithms discussed in Chapter 2 were applied to operation data collected from the wind farm in Summerside, PEI, illustrating a proposed fault prediction methodology. First, the data was labeled according to the error log, available online. Next, the missing data was imputed, and statistical tests to determine the stationarity of the time series are run. To capture the historical pattern, a lag window approach was employed, and to capture the most relevant lags, the PACF feature selection was applied, thus reducing the number of features in the training dataset. The simulations were conducted, and the performance of several classification models were compared, and the results were discussed to select the final model. The obtained results demonstrated the effectiveness of the proposed methodology to predict a fault in the wind farm.

Chapter 4

Conclusion

4.1 Summary and Conclusions

The research conducted in this thesis focused on the development of the methodology to predict faults in a 1 hour horizon. This goal was achieved by means of several machine learning models, i.e., ANNs, LSTMs, SVMs, and the ensemble models AdaBoost and majority voting classifiers. A case study was carried out to test and demonstrate the effectiveness of the proposed algorithms.

In Chapter 1, the motivations behind this research, along with the literature review, discussing various approaches to fault prediction were presented. Based on the literature review, the main research objectives were identified.

In Chapter 2, the theoretical background of the main techniques used in this thesis were reviewed. An overview of the statistical tests necessary in time series analysis, along with forecasting approaches, was provided. Various machine learning models were then reviewed. To improve the performance of a model performance and avoid overfitting, feature selection methods on both the lagged variables and original features were discussed.

In Chapter 3, the proposed fault prediction framework was presented, which consists of data labelling, data preprocessing, lag window creation, feature selection on the most significant lags, and AI models. One-year data, recorded with 10-minutes granularity provided by the city of Summerside, PEI, was used to develop, test, demonstrate, and compare various machine learning models, and a final prediction model was selected.

The main conclusions and findings of the presented thesis are the following:

- The results obtained with the proposed framework demonstrate the potential for its actual deployment to minimize the downtime of the wind farm studied.
- The presented results demonstrate that having enough historical data, machine learning models can properly capture the underlying failure patterns for fault prediction.
- The developed models can be trained off-line, and used on-line to quickly and adequately predict in real-time the probability of a fault happening in the next 1 hour horizon.

4.2 Contributions

The following are the major contributions of the presented research work for practical failure prediction of wind farm questions:

- Develop, compare, and apply several machine learning algorithms with the goal of determining the best model.
- Perform a feature selection analysis to identify the most relevant features and lags that would influence the prediction.
- Develop and validate a comprehensive and accurate machine-learning-based methodology.

4.3 Future Work

Based on the work presented in this thesis, the following possible areas for future work could be explored:

- Further test the proposed methodology with new data from different sites. In this case, the idea of transfer learning can be explored [135].
- Adapt the proposed framework for real-time deployment, and evaluate it performance in the real-life conditions.

- In this thesis, ANN, SVM, LSTM, and ensemble machine learning models have been utilized. Hence, exploration of other models such as k-nearest neighbors algorithms or other methods, traditionally used in different domains, such as Convolutional Neural Networks (CNNs), Generative Adversarial Networks (GANs), and Variational AutoEncoders (VAE) could be studied.
- Explore the use of reinforcement learning, which does not need large amounts of labeled data, such as in the case of [136], where it is applied to predictive maintenance.

References

[1] V. Smil, *Energy Transitions: Global and National Perspectives.* Praeger, 2017.

[2] A. Fakhouri and A. Kuperman, “Backup of renewable energy for an electrical island: Case study of israeli electricity systemcurrent status,” *The Scientific World Journal*, vol. 2014, pp. 609–687, January 2014.

[3] U.S. Department of Energy, “20% Wind energy by 2030: increasing wind energy’s contribution to US electricity supply,” Tech. Rep., 2008.

[4] S. Gsanger, “Wind power capacity worldwide reaches 597 gw, 50,1 gw added in 2018,” Jun 2019. [Online]. Available: https://wwindea.org/blog/2019/02/25/wind-power-capacity-worldwide-reaches-600-gw-539-gw-added-in-2018/

[5] C. A. Walford, “Wind turbine reliability: understanding and minimizing wind turbine operation and maintenance costs,” Sandia National Laboratories, Albuquerque, New Mexico, Tech. Rep., March 2006.

[6] B. Lu, Y. Li, X. Wu, and Z. Yang, “A review of recent advances in wind turbine condition monitoring and fault diagnosis,” in *2009 IEEE Power Electronics and Machines in Wind Applications*, June 2009, pp. 1–7.

[7] L. Colone, M. Reder, J. Tautz-Weinert, J. Melero, A. Natarajan, and S. Watson, “Optimisation of data acquisition in wind turbines with data-driven conversion functions for sensor measurements,” in 14th *Deep Sea Offshore Wind RD Conference, EERA DeepWind’2017*, vol. 137, Trondheim, Norway, December 2017, pp. 571 – 578. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1876610217353717

[8] IRENA, “Lcoe 2010-2017,” 2017. [Online]. Available: https://www.irena.org/Statistics/View-Data-by-Topic/Costs/LCOE-2010-2017

[9] R. Velmurugan and T. Dhingra, “Maintenance strategy selection and its impact in maintenance function: A conceptual framework,” *International Journal of Operations and Production Management*, vol. 35, pp. 1622–1661, December 2015.

[10] Q. Hao, Y. Xue, W. Shen, B. Jones, and J. Zhu, “A decision support system for integrating corrective maintenance, preventive maintenance and condition-based maintenance,” in *Construction Research Congress 2010: Innovation for Reshaping Construction Practice - Proceedings of the 2010 Construction Research Congress*, Banff Alberta, Canada, July 2010, pp. 470–479. [Online]. Available: https://www2.scopus.com/inward/record.uri?eid=2-s2.0-77956328783&doi=10.1061%2f41109%28373%2947&partnerID=40&md5=b4e6744f08e923890756882c0884fe13

[11] S. Nachimuthu, M. Zuo, and Y. Ding, “A decision-making model for corrective maintenance of offshore wind turbines considering uncertainties,” *Energies*, vol. 12, pp. 1–13, April 2019.

[12] S. Butler, “Prognostic algorithms for condition monitoring and remaining useful life estimation,” Ph.D. dissertation, National University of Ireland Maynooth, 2012. [Online]. Available: http://mural.maynoothuniversity.ie/3994/

[13] F. Nowlan and H. F. Heap, *Reliability-Centered Maintenance*. Springfield: National technical information Service, 1978.

[14] Z. Moravej and S. Bagheri, “Condition monitoring techniques of power transformers: A review,” *Journal of Operation and Automation in Power Engineering*, vol. 3, no. 1, pp. 71–82, Winter and Spring 2015. [Online]. Available: http://joape.uma.ac.ir/article_296.html

[15] J. Moubray, *Reliability-centered maintenance*, 2nd ed. New York : Industrial Press, 1997.

[16] R. Kothamasu, S. H. Huang, and W. H. VerDuin, “System health monitoring and prognostics — a review of current paradigms and practices,” *The International Journal of Advanced Manufacturing Technology*, vol. 28, no. 9, pp. 1012–1024, Jul 2006. [Online]. Available: https://doi.org/10.1007/s00170-004-2131-6

[17] W. Yang, P. J. Tavner, C. J. Crabtree, Y. Feng, and Y. Qiu, “Wind turbine condition monitoring: technical and commercial challenges,” *Wind Energy*, vol. 17, no. 5, pp. 673–693, August 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/we.1508

[18] F. Cheng, L. Qu, and W. Qiao, "A case-based data-driven prediction framework for machine fault prognostics," in *2015 IEEE Energy Conversion Congress and Exposition (ECCE)*, Sep. 2015, pp. 3957–3963.

[19] R. Satta, S. Cavallari, E. Pomponi, D. Grasselli, D. Picheo, and C. Annis, "A dissimilarity-based approach to predictive maintenance with application to HVAC systems," *Computing Research Repository (CoRR)*, pp. 1–15, January 2017. [Online]. Available: http://arxiv.org/abs/1701.03633

[20] K. Javed, "A robust & reliable Data-driven prognostics approach based on extreme learning machine and fuzzy clustering." Ph.D. dissertation, Université de Franche-Comté, April 2014. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01025295

[21] D. A. Mallesh, "Developing Leading and Lagging Indicators to Enhance Equipment Reliability in a Lean System," Ph.D. dissertation, University of Tennessee - Knoxville, December 2017. [Online]. Available: https://trace.tennessee.edu/utk_gradthes/4943/

[22] M. G. Pecht and R. Jaai, "A prognostics and health management roadmap for information and electronics-rich systems," *Microelectronics Reliability*, vol. 50, pp. 317–323, 2010.

[23] V. T. Tran and B. S. Yang, "Machine fault diagnosis and prognosis: The state of the art," *The International Journal of Fluid Machinery and Systems (IJFMS)*, vol. 2, pp. 61–71, January 2009. [Online]. Available: http://eprints.hud.ac.uk/id/eprint/16578/

[24] Y. Feng, Y. Qiu, C. J. Crabtree, H. Long, and P. J. Tavner, "Monitoring wind turbine gearboxes," *Wind Energy*, vol. 16, no. 5, pp. 728–740, July 2012. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/we.1521

[25] D. Chelidze and J. P. Cusumano, "A dynamical systems approach to failure prognosis," *Journal of Vibration and Acoustics, Transactions of the ASME*, vol. 126, pp. 2–8, February 2004.

[26] R. F. Orsagh, J. Sheldon, and C. J. Klenke, "Prognostics/diagnostics for gas turbine engine bearings," in *2003 IEEE Aerospace Conference Proceedings (Cat. No.03TH8652)*, vol. 7, March 2003, pp. 3095–3103.

[27] I. Roychoudhury and M. Daigle, "An Integrated Model-Based Diagnostic and Prognostic Framework," NASA, NASA Ames Research Center, Moffett Field, CA, Tech. Rep., 2011.

[28] K. Abid, M. Sayed Mouchaweh, and L. Cornez, "Fault prognostics for the predictive maintenance of wind turbines: State of the art," in *Communications in Computer and Information Science ECML PKDD 2018 Workshops*, Dublin, Ireland, March 2019, pp. 113–125.

[29] K. Javed, R. Gouriveau, N. Zerhouni, and P. Nectoux, "A feature extraction procedure based on trigonometric functions and cumulative descriptors to enhance prognostics modeling," in *2013 IEEE Conference on Prognostics and Health Management (PHM)*, Gaithersburg, MD, USA, June 2013, pp. 1–7.

[30] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: With Applications in R.* Springer Publishing Company, Incorporated, 2014.

[31] A. Kusiak, Z. Zhang, and A. Verma, "Prediction, operations, and condition monitoring in wind energy," *Energy*, vol. 60, pp. 1–12, August 2013. [Online]. Available: http://dx.doi.org/10.1016/j.energy.2013.07.051

[32] J. Yan, M. Ko, and J. Lee, "A prognostic algorithm for machine performance assessment and its application," *Production Planning & Control*, vol. 15, no. 8, pp. 796–801, February 2004.

[33] A. Soualhi, G. Clerc, H. Razik, M. El badaoui, and F. Guillet, "Hidden markov models for the prediction of impending faults," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 5, pp. 3271–3281, May 2016.

[34] W. Wang, "A model to predict the residual life of rolling element bearings given monitored condition information to date," *IMA Journal of Management Mathematics*, vol. 13, no. 1, pp. 3–16, January 2002.

[35] O. Uluyol, G. Parthasarathy, W. Foslien, and K. Kim, "Power Curve Analytic for Wind Turbine Performance Monitoring and Prognostics," in *Annual Conference of the Prognostics and Health Management Society*, Montreal, Quebec, Canada, September 2011, pp. 1–8.

[36] A. Kusiak and A. Verma, "Analyzing bearing faults in wind turbines: A data-mining approach," *Renewable Energy*, vol. 48, pp. 110–116, May 2012. [Online]. Available: http://dx.doi.org/10.1016/j.renene.2012.04.020

[37] A. Kusiak and A. Verma, "A data-driven approach for monitoring blade pitch faults in wind turbines," *IEEE Transactions on Sustainable Energy*, vol. 2, no. 1, pp. 87–96, Jan 2011.

[38] K. Leahy, R. L. Hu, I. C. Konstantakopoulos, C. J. Spanos, and A. M. Agogino, "Diagnosing wind turbine faults using machine learning techniques applied to operational data," in *2016 IEEE International Conference on Prognostics and Health Management (ICPHM)*, Ottawa, ON, Canada, June 2016, pp. 1–8.

[39] K. Leahy, R. Hu, I. Konstantakopoulos, C. Spanos, A. Agogino, and D. O' Sullivan, "Diagnosing and predicting wind turbine faults from scada data using support vector machines," *International Journal of Prognostics and Health Management*, vol. 9, pp. 1–11, February 2018.

[40] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, no. 1, pp. 321–357, Jun. 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=1622407.1622416

[41] A. Kusiak and W. Li, "The prediction and diagnosis of wind turbine faults," *Renewable Energy*, vol. 36, no. 1, pp. 16–23, October 2011.

[42] A. Zaher, S. McArthur, D. Infield, and Y. Patel, "Online wind turbine fault detection through automated scada data analysis," *Wind Energy*, vol. 12, no. 6, pp. 574–593, January 2009. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/we.319

[43] Y. Zhao, D. Li, A. Dong, D. Kang, Q. Lv, and L. Shang, "Fault prediction and diagnosis of wind turbine generators using SCADA data," *Energies*, vol. 10, no. 8, pp. 1–17, August 2017.

[44] L. Wang, Z. Zhang, J. Xu, and R. Liu, "Wind turbine blade breakage monitoring with deep autoencoders," *IEEE Transactions on Smart Grid*, vol. 9, no. 4, pp. 2824–2833, July 2018.

[45] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[46] P. Bangalore and L. B. Tjernberg, "An artificial neural network approach for early fault detection of gearbox bearings," *IEEE Transactions on Smart Grid*, vol. 6, no. 2, pp. 980–987, March 2015.

[47] P. Baraldi, F. Cadini, F. Mangili, and E. Zio, “Model-based and data-driven prognostics under different available information,” *Probabilistic Engineering Mechanics*, vol. 32, pp. 66 – 79, November 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0266892013000143

[48] J. Liu, W. Wang, F. Ma, Y. Yang, and C. Yang, “A data-model-fusion prognostic framework for dynamic system state forecasting,” *Engineering Applications of Artificial Intelligence*, vol. 25, no. 4, pp. 814 – 823, February 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0952197612000528

[49] S. Cheng and M. Pecht, “A fusion prognostics method for remaining useful life prediction of electronic products,” in *2009 IEEE International Conference on Automation Science and Engineering*, Bangalore, India, August 2009, pp. 102–107.

[50] M. Asgarpour and J. D. Sørensen, “Bayesian based Diagnostic Model for Condition based Maintenance of Offshore Wind Farms,” *Energies*, vol. 11, no. 2, pp. 1–17, January 2018.

[51] O. Brandes, J. Farley, M. Hinich, and U. Zackrisson, “The time domain and the frequency domain in time series analysis,” *Scandinavian Journal of Economics, Wiley*, vol. 70, pp. 25–42, March 1968.

[52] R. Adhikari and R. K. Agrawal, “An introductory study on time series modeling and forecasting,” *Computing Research Repository (CoRR)*, pp. 1–67, February 2013. [Online]. Available: http://arxiv.org/abs/1302.6613

[53] R. H. Shumway and D. S. Stoffer, *Time Series Analysis and Its Applications (Springer Texts in Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2005.

[54] YellowRoad, “Beware of data leakage,” Jul 2017. [Online]. Available: https://blog.myyellowroad.com/beware-of-data-leakage-f6c307009ad9

[55] R. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 2nd ed. Australia: OTexts, 2018.

[56] M. Hauskrecht, “Cs 3750 advanced topics in machine learning,” October 2018. [Online]. Available: http://people.cs.pitt.edu/~milos/courses/cs3750/lectures/class16.pdf

[57] M. Delfs, “Forecasting in the supply chain with machine learning techniques,” Master’s thesis, University of Bamberg, Germany, Jun 2018. [Online]. Available: https://www.uni-bamberg.de/en/cogsys/research/theses/advised-theses/

[58] M. Geurts, G. E. P. Box, and G. M. Jenkins, *Time Series Analysis: Forecasting and Control.* Wiley, 2015.

[59] E. Baltagi, *A Companion to Theoretical Econometrics.* John Wiley Sons, 2008.

[60] T. Iordanova, "An introduction to stationary and non-stationary processes," Jun 2019. [Online]. Available: https://www.investopedia.com/articles/trading/07/stationary.asp

[61] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice.* OTexts, 2014.

[62] S. Ben Taieb, G. Bontempi, A. Atiya, and A. Sorjamaa, "A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition," *Expert Systems with Applications*, vol. 39, pp. 7067–7083, August 2011.

[63] E. W. Saad, D. V. Prokhorov, and D. C. Wunsch, "Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks," *IEEE Transactions on Neural Networks*, vol. 9, no. 6, pp. 1456–1470, Nov 1998.

[64] V. T. Tran, B. S. Yang, and A. C. C. Tan, "Multi-step ahead direct prediction for machine condition prognosis using regression trees and neuro-fuzzy systems," *Expert Systems With Applications*, vol. 36, no. 5, pp. 9378–9387, July 2009. [Online]. Available: http://eprints.hud.ac.uk/id/eprint/16577/

[65] A. Sorjamaa and A. Lendasse, "Time series prediction using dirrec strategy," in *European Symposium on Artificial Neural Networks Bruges*, Belgium, April 2006, pp. 1–6.

[66] I. Pratama, A. E. Permanasari, I. Ardiyanto, and R. Indrayani, "A review of missing values handling methods on time-series data," in *2016 International Conference on Information Technology Systems and Innovation (ICITSI)*, Bandung, Indonesia, Oct 2016, pp. 1–6.

[67] M. Azur, E. Stuart, C. Frangakis, and P. Leaf, "Multiple imputation by chained equations: What is it and how does it work?" *International Journal of Methods in Psychiatric Research*, vol. 20, no. 1, pp. 40–49, March 2011.

[68] Z. Wu, W. Lin, and Y. Ji, "An integrated ensemble learning model for imbalanced fault diagnostics and prognostics," *IEEE Access*, vol. 6, pp. 8394–8402, 2018.

[69] Haibo He, Yang Bai, E. A. Garcia, and Shutao Li, “Adasyn: Adaptive synthetic sampling approach for imbalanced learning,” in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, Hong Kong, China, June 2008, pp. 1322–1328.

[70] T. Fawcett, “Learning from imbalanced classes,” Sep 2017. [Online]. Available: https://www.svds.com/learning-imbalanced-classes/

[71] L. Nieradzik, “Losses for image segmentation,” Sep 2018. [Online]. Available: https://lars76.github.io/neural-networks/object-detection/losses-for-segmentation/

[72] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[73] J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” *Journal of Machine Learning Research*, vol. 13, pp. 281–305, February 2012. [Online]. Available: https://www2.scopus.com/inward/record.uri?eid=2-s2.0-84857855190&partnerID=40&md5=fe76c9d1e6e5375afab7760adbe1ae4f

[74] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep. 1995. [Online]. Available: https://doi.org/10.1023/A:1022627411411

[75] Z. Zhang and A. Kusiak, “Monitoring wind turbine vibration based on scada data,” *Journal of Solar Energy Engineering*, vol. 134, no. 2, pp. 1–12, February 2012. [Online]. Available: https://doi.org/10.1115/1.4005753

[76] Changxue Ma, M. A. Randolph, and J. Drish, “A support vector machines-based rejection technique for speech recognition,” in *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, vol. 1, Salt Lake City, UT, USA, USA, May 2001, pp. 381–384.

[77] D. Gorgevik, D. Cakmakov, and V. Radevski, “Handwritten digit recognition by combining support vector machines using rule-based reasoning,” in *Proceedings of the 23rd International Conference on Information Technology Interfaces (ITI), 2001.*, vol. 1, Pula, Croatia, Croatia, June 2001, pp. 139–144.

[78] Y. Vidal, F. Pozo, and C. Tutivn, "Wind Turbine Multi-Fault Detection and Classification Based on SCADA Data," *Energies*, vol. 11, no. 11, pp. 1–18, November 2018.

[79] A. Ng, "CS 229 - support vector machines," Aug 2019. [Online]. Available: http://cs229.stanford.edu/notes/cs229-notes3.pdf

[80] T. Fletcher, "Support vector machines explained," Feb 2008. [Online]. Available: https://cling.csd.uwo.ca/cs860/papers/SVM_Explained.pdf

[81] A. Ben-Hur and J. Weston, "A user's guide to support vector machines." *Methods in molecular biology*, vol. 609, pp. 223–239, October 2010.

[82] R. Raul, *Neural networks: a systematic introduction.* Springer, 1996.

[83] O. K. Ernst, "Stochastic gradient descent learning and the backpropagation algorithm," University of California, San Diego, La Jolla, CA, Tech. Rep., 2014.

[84] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, San Diego, CA, December 2015, pp. 1–15.

[85] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *Computing Research Repository (CoRR)*, pp. 1–20, November 2018. [Online]. Available: http://arxiv.org/abs/1811.03378

[86] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, Haifa, Israel, 2010, pp. 807–814. [Online]. Available: http://dl.acm.org/citation.cfm?id=3104322.3104425

[87] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout," in *IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings (ICASSP)*, Vancouver, BC, Canada, May 2013, pp. 1–5.

[88] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, vol. 15, Fort Lauderdale, FL, USA, April 2011, pp. 315–323. [Online]. Available: http://proceedings.mlr.press/v15/glorot11a.html

[89] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in 30^{th} *International Conference on Machine Learning*, Atlanta, USA, June 2013, pp. 1–6.

[90] J. Brownlee, “What is the difference between a batch and an epoch in a neural network?” Aug 2019. [Online]. Available: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

[91] G. Raskutti, M. Wainwright, and B. Yu, “Early stopping and non-parametric regression: An optimal data-dependent stopping rule,” *Journal of Machine Learning Research*, vol. 15, pp. 335–366, 2014. [Online]. Available: https://www2.scopus.com/inward/record.uri?eid=2-s2.0-84897089085&partnerID=40&md5=b7873aed920095709eb0b51afd368216

[92] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[93] B. C. Csaji, “Approximation with artificial neural networks,” Master’s thesis, Eindhoven University of Technology, 2001. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.2647&rep=rep1&type=pdf

[94] J. Heaton, “The number of hidden layers,” Dec 2018. [Online]. Available: https://www.heatonresearch.com/2017/06/01/hidden-layers.html

[95] D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, Eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations.* Cambridge, MA, USA: MIT Press, 1986.

[96] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, March 1994.

[97] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[98] Y. Zhang, R. Xiong, H. He, and M. G. Pecht, “Long short-term memory recurrent neural network for remaining useful life prediction of lithium-ion batteries,” *IEEE Transactions on Vehicular Technology*, vol. 67, no. 7, pp. 5695–5705, July 2018.

[99] A. Graves, N. Beringer, and J. Schmidhuber, "A comparison between spiking and differentiable recurrent neural networks on spoken digit recognition," in *International Conference on Neural Networks and Computational Intelligence*, Grindelwald, Switzerland, January 2004, pp. 164–168.

[100] R. Zhao, J. Wang, R. Yan, and K. Mao, "Machine health monitoring with LSTM networks," in 10^{th} *International Conference on Sensing Technology (ICST)*, Nanjing, China, Nov 2016, pp. 1–6.

[101] M. Farrash, "Machine learning ensemble method for discovering knowledge from big data," Ph.D. dissertation, University of East Anglia, 2016.

[102] L. I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*. New York, NY, USA: Wiley-Interscience, 2004.

[103] J. Brownlee, "What is a hypothesis in machine learning?" Jun 2019. [Online]. Available: https://machinelearningmastery.com/what-is-a-hypothesis-in-machine-learning/

[104] R. Polikar, "Ensemble based systems in decision making," *IEEE Circuits and Systems Magazine*, vol. 6, pp. 21–45, September 2006.

[105] T. G. Dietterich, "Ensemble methods in machine learning," in *Proceedings of the First International Workshop on Multiple Classifier Systems*, London, UK, December 2000, pp. 1–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=648054.743935

[106] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *ICML'96 Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, Bari, Italy, July 1996, pp. 148–156.

[107] Y. Sun, M. S. Kamel, A. K. Wong, and Y. Wang, "Cost-sensitive boosting for classification of imbalanced data," *Pattern Recognition*, vol. 40, no. 12, pp. 3358 – 3378, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320307001835

[108] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. New York, NY, USA: Springer New York Inc., 2001.

[109] M. Brubaker, "CSC 411 - adaboost handout," September 2015. [Online]. Available: http://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/AdaBoost.pdf

[110] M. B. Hossin, "A review on evaluation metrics for data classification evaluations," *International Journal of Data Mining Knowledge Management Process*, vol. 5, no. 2, pp. 1–11, Mar 2015.

[111] K. Deng, "Omega: On-line memory-based general purpose system classifier," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, November 1998.

[112] A. Jovi, K. Brki, and N. Bogunovi, "A review of feature selection methods with applications," in 38th *International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia, May 2015, pp. 1200–1205.

[113] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, Apr 2006. [Online]. Available: https://doi.org/10.1007/s10994-006-6226-1

[114] J. Taylor, "Classification decision trees," Oct 2012. [Online]. Available: http://statweb.stanford.edu/~jtaylo/courses/stats202/restricted/notes/trees.pdf

[115] R. May, G. Dandy, and H. Maier, "Review of input variable selection methods for artificial neural networks," *Artificial Neural Networks-Methodological Advances and Biomedical Applications*, pp. 1–28, April 2011.

[116] P. S. P. Cowpertwait and A. V. Metcalfe, *Introductory Time Series with R*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[117] J. Brownlee, "A gentle introduction to autocorrelation and partial autocorrelation," Aug 2019. [Online]. Available: https://machinelearningmastery.com/gentle-introduction-autocorrelation-partial-autocorrelation/

[118] G. Van Rossum and F. L. Drake Jr, *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[119] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: http://tensorflow.org/

[120] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[121] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[122] T. Oliphant, "NumPy: A guide to NumPy," USA: Trelgol Publishing, 2006. [Online]. Available: http://www.numpy.org/

[123] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, Austin, TX, 2010, pp. 51–56.

[124] M. Farrokhabadi, "Data-driven mitigation of energy scheduling inaccuracy in renewable-penetrated grids: Summerside electric use case," *Energies*, vol. 12, no. 12, pp. 1–23, June 2019. [Online]. Available: https://www.mdpi.com/1996-1073/12/12/2228

[125] A. Rubinsteyn and S. Feldman, "Fancyimpute: Multivariate imputation and matrix completion algorithms implemented in python," Apr 2019. [Online]. Available: https://github.com/iskandr/fancyimpute

[126] P. Chen, B. Liao, G. Chen, and S. Zhang, "Understanding and utilizing deep neural networks trained with noisy labels," *Computing Research Repository (CoRR)*, pp. 1–13, May 2019. [Online]. Available: http://arxiv.org/abs/1905.05040

[127] J. Han, P. Luo, and X. Wang, "Deep self-learning from noisy labels," *Computing Research Repository (CoRR)*, pp. 1–10, August 2019. [Online]. Available: https://arxiv.org/pdf/1908.02160.pdf

[128] "Fehlerliste v90," Tech. Rep., Apr 2015. [Online]. Available: https://dokumen.tips/documents/fehlerlistev902944665r11.html

[129] A. Goteti, "Machine learning approach to the design of autonomous construction equipment applying data-driven decision support tool," Master's thesis, Blekinge Institute of Technology, Department of Computer Science and Engineering, 2019.

[130] S. Waslander, "Optimization for training deep models," Jun 2017. [Online]. Available: http://wavelab.uwaterloo.ca/wp-content/uploads/2017/04/Lecture-4-1.pdf

[131] P. T. Tran and L. T. Phong, "On the convergence proof of amsgrad and a new version," *IEEE Access*, vol. 7, pp. 61 706–61 716, May 2019.

[132] Y. Sun, M. S. Kamel, and Y. Wang, "Boosting for learning multiple classes with imbalanced class distribution," in *Sixth International Conference on Data Mining (ICDM'06)*, Hong Kong, China, Dec 2006, pp. 592–602.

[133] Xtra-Computing, "Xtra-computing/thundersvm," Oct 2019. [Online]. Available: https://github.com/Xtra-Computing/thundersvm

[134] M. Willcox, "Occam's razor and machine learning: Teradata blogs," Sep 2017. [Online]. Available: https://www.teradata.com/Blogs/Occams-razor-and-machine-learning

[135] L. Torrey, T. Walker, J. Shavlik, and R. Maclin, "Using advice to transfer knowledge acquired in one reinforcement learning task to another," in *Proceedings of the 16th European Conference on Machine Learning*, Porto, Portugal, October 2005, pp. 412–424. [Online]. Available: http://dx.doi.org/10.1007/11564096_40

[136] C. Zhang, C. Gupta, A. Farahat, K. Ristovski, and D. Ghosh, "Equipment health indicator learning using deep reinforcement learning," in *Machine Learning and Knowledge Discovery in Databases*, Dublin, Ireland, September 2019, pp. 488–504.