U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Developing data-driven models to assess structures' health condition

João Francisco Lopes Cruz de Carvalho

Dissertação de Mestrado
Orientadora: Vera Miguéis

Mestrado Integrado em Engenharia Mecânica
Departamento de Engenharia Mecânica
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal
2020, June 29^{th}

Aos que são geneticamente parecidos comigo. E à Bianca.

"There's no substitute for hard work"
- Thomas Edison

Acknowledgments

This work was carried out under the project PTDC / ECI-EGC / 31355/2017 - ” S4Bridges - A smart approach for the maintenance of existing bridges”, co-financed by the Foundation for Science and Technology IP (PIDDAC) and the European Structural and Investment Funds (FEEI), through COMPETE2020 - Programa Operacional Competitividade e Internacionalização (POCI).

This project marks the end of a five year journey at Faculdade de Engenharia da Universidade do Porto (FEUP). To the institution and and everyone that welcomed, influenced, taught and helped me develop not only as a future engineer, but also as a person, thank you.

I would like to thank Professor Vera Miguéis, for her guidance and invaluable wisdom, her kindness and support throughout all the stages of the project, and for all the time and patience dedicated to ensure I could get the best experience of this thesis on a topic I cherish - Data analytics -, for this dissertation would not have been possible without her.

I would like to thank my family for all the support not only in this stage of my life, but in everything I have done in my life, the unconditional love, the education that raised me to be the person I am today and for everything they have done for me in my 22 years of existence.

To all my closest friends, who have always provided me a life of joy and fun, to my chlorine friends, who were there in the end of the day for a rather needed time of unwinding, and to all my colleagues for making a difference in my life in their own way, thank you.

Finally, I would like to thank my girlfriend, Bianca, the person who followed my journey on this project the closest, for all the love, support, companionship and for watching my mental decline from a rather unsafe distance.

I am truly grateful for everything that led me to this present day, for everything I was given, for every experience I lived, and mostly, for meeting every person that was part of my journey so far.

Abstract

More than ever, humanity is requiring more complex and more secure infrastructures. The finite amount of building space, the unsustainability of demolishing old buildings to create new ones, both in a environmental and economical perspective, and the intensification of natural catastrophes create the need for a better maintenance of already existent buildings.

Structural Health Monitoring (SHM) emerges as a tool to better assess structural behavior of a construction, in order to ensure optimal levels of performance, assist maintenance planning and prolong its service life. In this context, this project aims to develop a tool for damage detection in a real-life, small scale structure, via data-driven models.

Information from several variables is gathered for a period of just over a year. Data is recorded for both a damaged and an undamaged condition. Several temperature variables are used to predict structure's behavior, described by the acceleration, slope, displacement and strain, measured in different regions. Data from the undamaged situation are used to construct prediction models based on Random Forests (RF), Support Vector Regression (SVR) and Feed-forward (ANN) and Recurrent (LSTM) Neural Networks models. These models aim at predicting the behavior of the structure on a regular, undamaged state. Two Ensemble models are created from the first three models. In order to mimic a real use of the models, acceleration, slope, displacement and strain are predicted for future periods. Model performance is evaluated with the estimation of MAPE, MSE and R^2 metrics, applied to testing data from undamaged period. Hotelling T^2 and EWMA quality control charts are constructed to detect abnormal situations, using predictions' residuals. From control charts' output, two damaged detection rules are developed. One rule is based on percentage points out of control on a moving window (PPOC), and the other on the cumulative sum of control charts' score (CUSUM).

The prediction models developed presented similar prediction error for testing data. Regarding anomaly detection, Hotelling Control Charts outperformed EWMA for all models. The combined use of RF/SVR and Hotelling chart failed to successfully detect abnormal situations, while other models presented satisfactory results. Hotelling charts built with residuals from Ensemble and ANN models failed to differentiate damaged and undamaged situations in damage detection. LSTM models presented the best results with 0,0% false positives and 48% false negatives for PPOC approach, and 0,10% false positives, 7% false negatives and 13 hours and 45 minutes damage detection time for CUSUM.

Overall, the most complex and advanced LSTM models yielded the best results across all stages. LSTM models showed potential for damage detection, when paired with Hotelling T^2 Control Charts and both PPOC and CUSUM damage detection rules.

Summing up, the results obtained support the relevance of the methodology proposed. In fact, this constitutes a tool that may be used to accurately monitor the condition of a real structure in real time.

Resumo

Mais do que nunca, a humanidade requer infraestruturas mais complexas e mais seguras. O limitado espaço disponível para construção, a insustentabilidade da demolição de edifícios para criação de novos, ambiental e economicamente, e a intensificação das catástrofes naturais levam à necessidade de uma melhor manutenção dos edifícios existentes.

Desta forma, *Structural Health Monitoring* (SHM) surge como uma ferramenta para melhor aferir o comportamento estrutural de uma construção, por forma a garantir níveis ótimos de performance, auxiliar planos de manutenção e prolongar a sua vida útil. Neste contexto, este projeto pretende desenvolver uma ferramenta para deteção de dano numa estrutura real, de pequena escala, através de modelos numéricos de análise de dados.

Dados referentes a várias variáveis são recolhidos durante um período pouco superior a um ano. Os dados são gravados tanto para um estado não danificado da estrutura, como para um estado danificado. Diversas variáveis de temperatura são usadas para prever o comportamento da esturutura, descrito pela aceleração, inclinação, deslocamento e tensão, medidos em diferentes pontos da mesma. Os dados referentes à condição não danificada são usados para desenvolver modelos preditivos baseados em modelos *Random Forests* (RF), *Support Vector Regression* (SVR) e *Feed-forward* (ANN) e *Recurrent* (LSTM) *Neural Networks*. Estes modelos pretendem antecipar o comportamento da estrutura numa situação regular, não danificada. Dois modelos Ensemble são desenvolvidos a partir dos três primeiros modelos mencionados. De forma a simular o uso real destes modelos, as variáveis de aceleração, inclinação, deslocamento e tensão são previstas para períodos futuros. Os modelos são avaliados com a estimativa do MAPE, MSE e R^2 para dados de teste referentes ao estado não danificado. Cartas de controlo de Hotelling T^2 e EWMA são construídas a partir dos resíduos das previsões dos diferentes modelos, com o objetivo de detetar situações anómalas na estrutura. A partir do *output* das cartas de controlo, duas regras para a deteção de dano são desenvolvidas. Uma regra baseia-se na percentagem de pontos fora de controlo numa janela flutuante (PPOC), e a outra baseada na soma cumulativa do *score* das cartas de controlo (CUSUM).

Os modelos preditivos apresentaram erro de previsão semelhante para os dados de teste. Relativamente à deteção de situações anómalas, os resultados das cartas de controlo de Hotelling superam os resultados das cartas EWMA. O uso combinado de modelos preditivos RF/SVR e cartas de controlo Hotelling fracassou na deteção de situações anómalas. Cartas de Hotelling desenvolvidas com resíduos dos modelos ANN e Ensemble não conseguiram distinguir, com sucesso, as duas condições da estrutura, na aplicação das regras de deteção de dano. Os modelos LSTM apresentaram os melhores resultados de deteção de dano com 0,0% falsos positivos e 48% falsos negativos para a abordagem PPOC, e 0,1% falsos positivos, 7% falsos negativos e um tempo de deteção de dano de 13 horas e 45 minutos para a abordagem CUSUM.

No geral, os modelos LSTM, mais complexos e avançados, produziram os melhores resultados em todas as etapas do processo. Estes modelos demonstraram o potencial para a deteção de dano, quando combinados com Cartas de controlo de Hotelling T^2 e ambas as regras de deteção de dano estudadas.

Concluindo, os resultados obtidos suportam a relevância da metodologia proposta. De facto, o projeto constitui um ferramenta que pode ser usada para melhor monitorizar a condição de uma estrutura real, em tempo real.

Contents

List of Abbreviations

- ANN - Artificial Neural Networks
- RF - Random Forests
- SVR - Support Vector Regression
- SVM - Support Vector Machines
- LSTM - Long Short-Term Memory
- CBLSTM - Convolutional Bi-directional Long Short-Term Memory
- KDD - Knowledge Discovery in Databases
- EWMA - Exponentially Weighted Moving Average
- UCL - Upper Control Limit
- CUSUM - Cumulative Sum Control Chart
- PPOC - Percentage Points Out of Control

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Project framework and motivation

The health and functionality of structures around the world is vital from an economical, environmental and security point-of-view. The need for reliable and secure infrastructures is ever growing as the human population grows. However, there is a finite amount of space to build new structures, leading to the necessity of preserving the already existing buildings. Demolishing old buildings to free up space for new ones is not economically viable in many situations, neither environmentally sustainable. Furthermore, the increasing complexity of new structures, and the intensification of natural disasters, puts at risk human safety and the security of structures. All these factors appeal to a better maintenance of buildings.

In this context, monitoring systems have gained relevance, in order to assist the decision making process regarding said structures. These are *Structure Health Monitoring (SHM)* systems, which are responsible for collecting, preparing, transforming and interpreting data, to detect anomalies and potential damages in structures.

Even though many data about structures has been collected throughout the years, few were the cases where it was applied to support the decision making process. (Tomé et al., 2019)

1.2 Project Objectives

This project aims to develop a SHM tool to aid the damage detection process. Initially, the tool is developed with a small scale structure and applied to it. On a second phase of the project, this tool should be applied on a full size structure.

The SHM tool is intended to correctly predict the normal behaviour of the structure. When damage is induced to the structure, the model should not be able to predict its behaviour successfully. Therefore, prediction error, or residual, should increase, indicating the abnormal damaged situation of the structure.

The developed SHM tool should avoid false positives, i.e. wrongfully signaling damage in the structure, as it would take its functionality in discredit and induce in over maintenance. Additionally, it should also correctly detect damage induced to the structure in a considerably small time window since the inception of the damage state, as failing to detect damage or detecting it after a considerate amount of time could induce on irreversible damage to the structure and potential human and economic loss.

1.3 Method

Following a KDD methodology, damage detection is divided into six parts: data selection, data cleaning, data transformation, data mining, evaluation and interpretation and decision making. In order to develop an optimal damage detection tool, several techniques are studied, tested and developed along the several phases of the project.

Data Selection, Cleaning and Transformation aim to process and prepare data into a ready to use state to feed the models, both for training and further predicting.

In Data Mining, several algorithms are applied to predict the normal behaviour of the structure, being those Random Forests (RF), Support Vector Regression (SVR), a Feed-forward Neural Network, named Artificial Neural Network (ANN), and two Recurrent Neural Networks, i.e. Bi-Directional Long Short-Term Memory Networks (CBLSTM), one outputting one dependent variable per model (single output), and another outputting all dependent variables for one model (multiple output).

As predictions are calculated for all data, models are evaluated with performance metrics(MAPE, RSE and R^2), and control charts are constructed to signal abnormal observations, via predictions' residuals. Finally, control chart's results are used to develop two different damage detection tools.

1.4 Structure

To rightfully represent the workflow of this project and following the forementioned method, this thesis is divided into 5 further chapters. Chapter 2 presents Background Knowledge, helpful for a better understanding of the techniques applied. Chapter 3 introduces the State of the Art on Structure Health Monitoring, describing the most common techniques and alternatives paths to the one followed in this project.

After the theoretical introduction, Chapter 4 presents the structure used, data gathered and the methodology in great detail. To support the explanation of procedures, many additional content can be found on Annexes, namely code, in R, for every process.

Chapter 5 presents the results attained, comparing each technique used and presenting the final formulation of the optimal damage detection tool. Similarly to Chapter 4, additional detail on results, complementary to the one presented in the chapter, is available is Annexes.

Finally, Chapter 6 presents the conclusion. Here, a summary of all steps is made, results are presented in a compact manner, the final optimal damage detection tool is emphasized. To conclude, suggestions for future work are done.

Chapter 2

Background Knowledge

The present section introduces relevant subjects to the development of the project. Firstly, it is introduced the concept of Knowledge Discovery in Databases, as the starting point of the project, regarding important matters such as data cleaning and processing. After that, Data Mining is presented in more detail, as a step of KDD; this subsection introduces different Data Mining algorithms applied in the project. Finally, Structure Health Monitoring is presented.

2.1 Knowledge Discovery in Databases (KDD)

The world we live in is ever changing; with it, new information arises everyday and the insatiable human mind cannot simply get enough data. The digital era brings a whole new level of possibility for data wharehousing, to a point where most of the data ever collected has not ever been analyzed by a human being.

From this problem surges the need to use computation not only for storage, but also to analyze the data stored. From that need comes the concept of Knowledge Discovery in Databases (KDD).

According to Brachman and Anand (1994), knowledge discovery in database (KDD) processes are described as methods for the nontrivial extraction of implicit, previously unknown, and potentially useful information from data.

Regarding this definition, *data* are a set of facts (cases in a database); *pattern* is an expression in some language describing a subset of the data or a model applicable to the subset; the term *process* implies that KDD comprises many steps, which involve data preparation, search for patterns, knowledge evaluation, and refinement, all repeated in multiple iterations; *nontrivial* means that it is not a straightforward computation of predefined quantities like computing the average value of a set of numbers. (Fayyad et al., 1996)

Therefore, KDD can be thought as the application of the scientific method to Data Mining. In addition to performing data mining, a typical KDD process model includes a methodology for extracting and preparing data, as well as making decisions about actions to be taken once data mining has taken place. This process is crucial for relevant data mining, specially for large databases. (Roiger, 2017)

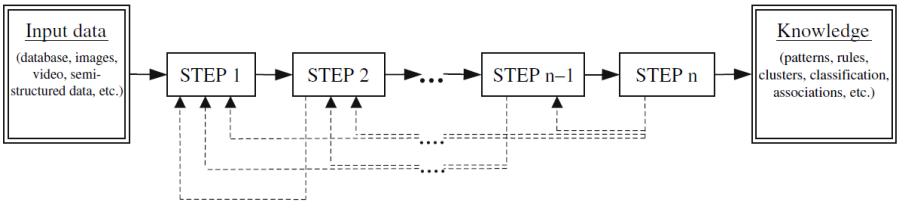


Figure 2.1: Sequential structure of a KDD process. Cios et al. (2007)

2.1.1 A Prototypical KDD System

Many KDD systems exist, varying considerably in design. However, it is possible to create a prototypical KDD System, sort of a foundation upon which KDD systems are built. (Frawley et al., 1992)

The KDD process is interactive and iterative. It consists of multiple steps that are executed in a sequence. Each subsequent step is initiated upon successful completion of the previous step, and requires the result generated by the previous step as its input. (Cios et al., 2007)

The major input for a KDD process is raw data present in a database, from which it is possible to achieve the desired output: knowledge in form of patterns, rules, classifications or clusters. Overall, a standard KDD process can be divided in 5 main steps, as shown in figure 2.2: selection, preprocessing, transformation, data mining, interpretation/evaluation. (Usama et al., 1996)

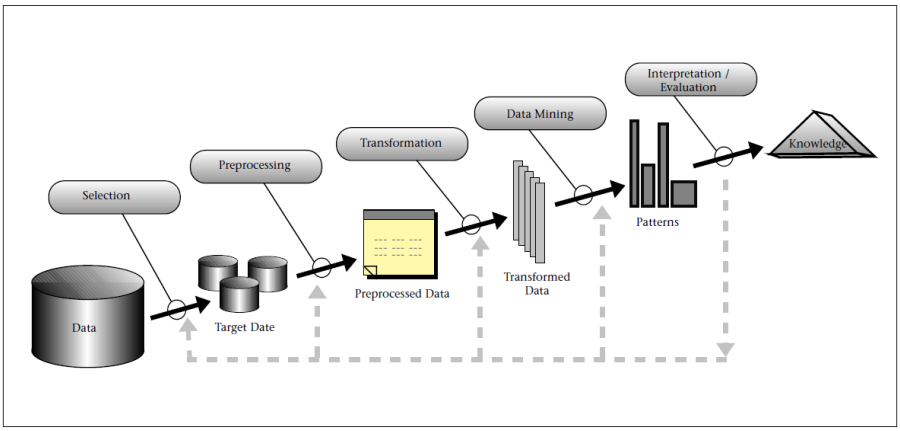


Figure 2.2: An Overview of the Steps That Compose the KDD Process. Usama et al. (1996)

A practical view of the KDD process is given by Brachman and Anand (1996), emphasizing the interactive nature of the process. This process is outlined in basic steps by Usama et al. (1996), as shown below:

1. Understanding the scope of the project and the goals from the costumer's point of view;
2. Selecting a data set, or a subset of variables or data samples, from which knowledge will emerge.
3. Cleaning data and preprocessing; operations such as noise removal, handling missing fields or values and accounting for time-sequence information and known changes should be performed in this step;
4. Data reduction and projection; representing data depending on the goal of the task, reducing, if possible, the number of variables in study;
5. Matching the goals of the KDD process (step 1) to a particular data-mining method, such as classification, regression, summarization or clustering; these methods will be further studied in section 2.2;
6. Exploratory analysis and model and hypothesis selection; this step includes choosing the data mining algorithm(s) and selection method(s) and deciding which models and parameters might be appropriate;
7. Data mining; searching for patterns of interest in a particular, intuitive representational form or a set of representational forms;
8. Interpreting mined patterns, returning to any of the steps from 1 to 7, if needed, for further iteration;
9. Using the knowledge; knowledge retrieved from KDD process must be directly applied into a system or further actions and documented; it is also important to check for and resolving conflicts with previous knowledge

Every step of the KDD process is crucial to attain the best, most relevant and useful knowledge from any database. Tasks like data cleaning and preprocessing may be the most time consuming in the process, specially for large databases, despite being sometimes overlooked. However, most of the steps serve as tool to aid the data-mining process, either by preparing data to ease the data mining process or by interpreting and representing in an intuitive fashion the knowledge scoured. The next section presents data-mining in a detailed perspective, from the main goals and motivating challenges for its development to the different data mining methods and algorithms.

2.2 Data Mining

Data Mining is defined as the process to automatically find structure in data repositories. Therefore, Data Mining techniques are used as a tool to investigate large databases in search for useful patterns that otherwise would not be found.

According to Roiger (2017), data mining comes from the areas of statistics, mathematics, machine learning and business. It combines methods from machine learning, such as modeling techniques, search algorithms and learning theories along side with hypothesis testing, sampling and estimation from statistics, as represented on figure 2.3.

As Tan et al. (2014) state, not all data discovery methods are considered Data Mining. Tasks like searching for individual records of a swimmer are considered information retrieval. Although these tasks require complex algorithms and data structures, they fall on the scope of regular data organization and indexing. Nevertheless, Data Mining methods can be applied to these traditional algorithms, in order to enhance them.

Many Data Mining algorithms techniques exist; however, all methods use *induction-based learning*. Induction-based learning is the process of conceptualizing a general idea based on specific events, previously observed. (Roiger, 2017)

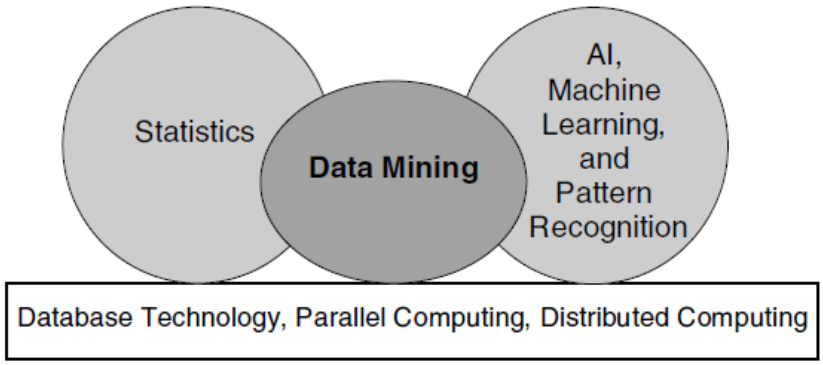


Figure 2.3: Data mining as a confluence of many disciplines. (Tan et al., 2014)

2.2.1 Motivating challenges for Data Mining Development

As previously state, new larger and more complex datasets have brought practical difficulties to traditional data analysis techniques, motivating the development of data mining. These difficulties are presented below. (Tan et al., 2014)

Scalability As data sets grow in scale, from gigabytes and terabytes to even petabytes, data mining algorithms must be scalable. Many algorithms employ special searching strategies to handle exponential search problems.

High Dimensionality Data sets have grown not only on number of records, but also on number of attributes for each record. Traditional data analysis techniques, built for low-dimensional data sets, often do not work well with high-dimensional data. For many common algorithms, the computational complexity increases rapidly as the dimensionality increases.

Heterogeneous and Complex Data Traditional data sets often deal with attributes of the same type, categorical or continuous. However, new data sets, with more complex data, often present attributes of different types. Examples of such non-traditional types of data include collections of Web pages containing semi-structured text and hyperlinks; DNA data with sequential and three-dimensional structure; and climate data that consists of time series measurements (temperature, pressure, etc.) at various locations on the Earth's surface. Techniques for mining such complex data sets should take new relationships in data into consideration, as the ones mentioned above.

Data Ownership and Distribution Sometimes the data needed for mining techniques is stored in different geographical locations. This requires the development of distributed data mining techniques. Such techniques should face three main challenges: (1) how to reduce the amount of communication needed to perform the distributed computation, (2) how to effectively consolidate the data mining results obtained from multiple sources, and (3) how to address data security issues.

Non-traditional Analysis Traditional statistics approach reveals itself as a labor-intensive method. Current data analysis tasks often require the generation and evaluation of thousands of hypotheses, and consequently, the development of some data mining techniques has been motivated by the desire to automate the process of hypothesis generation and evaluation. Additionally, new data sets are not often built upon carefully designed experiments, making the data not random.

2.2.2 Data Mining Methods

Data mining goals are defined by the intended use of the system. It can be distinguished into verification and discovery. With *Verification*, the system is limited to verifying the user's hypothesis. With *discovery*, the system autonomously finds new patterns. Discovery can be further subdivided into *prediction*, where the system finds patterns to predict the future behaviour of some entity, and *description*, where the system finds patterns for representation at a human-understandable form. (Usama et al., 1996)

These goals can be achieved from a variety of methods, presented below. (Usama et al., 1996)

Classification builds a function that classifies a data item into one of a sort of pre-defined classes.

Regression builds a function to classify a data item into a continuous scale.

Clustering is a descriptive task made to identify a finite set of categories (clusters) to describe data.

Summarization aims for finding a compact description for a subset of data. For example, tabulating the mean and standard deviation for all fields of a data set. More sophisticated methods involve derivation of summary rules, multivariate visualization techniques and the discovery of functional relationships between variables.

Dependency modeling builds a model that describes significant dependencies between variables, at either a structural level or quantitative level.

Change and deviation detection aims to discover the most significant changes in data from previously measured normative values.

Section 2.3 studies regression Data Mining algorithms used in this project, namely Random Forests, Support Vector Regression and feed-forwards (ANN) and recurrent (LSTM) Neural Networks.

2.2.3 Model Overfitting

The errors commited by a predictor can be grouped into *training errors* and *generalization errors*. Training errors refer to the misprediction of training records, while generalization errors refers to the expected error on predicting unseen data. A good predictor presents both training and generalization errors are low. One predictor might have low training error but high generalization error. This event is called *model overfitting*. Potential causes for this event might be the presence of noise in training data and lack of representative samples. (Tan et al., 2014)

2.2.4 Model Evaluation

In order to compare the performance of different predictors, model evaluators are applied. These mathematical formulas compare one model predictions with actual values, outputting a metric representing the suitability of the model to predict the dependant variable. Three evaluators are applied: R^2, MSE and MAPE. In the following expressions let y_i represent the real value and f_i the predicted value of observation i.

R^2, or, Coefficient of determination represents the proportion of the variance of the dependent variable being predicted. It is mathematically formulated as:

$$R^2 = \left(\frac{n \sum_i^n (f_i y_i) - (\sum_i^n f_i)(\sum_i^n y_i)}{\sqrt{(n \sum_i^n f_i^2 - (\sum_i^n f_i)^2)(n \sum_i^n y_i^2 - (\sum_i^n y_i)^2)}} \right)^2$$

MSE, or Mean Squared Error, measures the average of the squared errors, being mathematically formulated as follows:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - f_i)^2$$

MAPE, abbreviation of Mean Absolute Percentage Error, represents the error as a percentage of the real value and is formulated as such:

$$MAPE = \frac{1}{n} \sum i = 1n \left| \frac{y_i - f_i}{y_i} \right|$$

2.3 Data Mining Algorithms

The following paragraphs explore some of the existing data mining algorithms. This detailed analysis describes decision trees, feed-forward and recurrent neural networks, support vector machines and random forests.

2.3.1 Decision Trees

A decision tree is a data mining algorithm for classification or regression solutions. It is structured as a sequence of carefully crafted questions about the attributes of the data. After each question a conclusion is taken, advancing to a different question or a final decision. Recursive partition is done until simple models can be fit on data from each group, assigning a final value for the output variable for records from said group.

Questions are organized in a decision tree, following the example shown on figure 2.4. The tree is organized in a hierarchical structure consisting of directed edges and 3 different types of nodes:

- One *root node* that has no incoming edges and zero or more outgoing edges;
- *Internal nodes*, each having exactly one incoming edge and two or more outgoing edges;
- *Leaf* or *terminal nodes*, each having one incoming edge and no outgoing edges. Each leaf node is assigned to a value of a continuous scale,in regression problems.

In regression problems, the value inherent to each node corresponds to the average of the dependant variable of all training records in the node. (Breiman et al., 1984) Assigning a value to a test record is a straightforward task after building the decision tree. Starting on the root

node, conclusions are taken based on the question on each node, and the correct path is followed respecting the directed edges, leading to a leaf node, containing the output.

For a given data set, there are exponentially many possible decision trees. While some are more accurate than others, finding the optimal decision tree is infeasible from a computational perspective, due to the large amount of possibilities. Therefore, it is needed to develop an efficient algorithm to induce a reasonably accurate, although sub-optimal decision tree in a reasonable amount of time. A commonly used strategy is *Hunt's algorithm*, a strategy that builds a decision tree by recursively taking different possible divisions for each data subgroup of a node, and estimating a metric of purity, in order to choose the division that provides a better distinction of records. This algorithm is the base for many Decision Trees algorithms and Random Forests (section 2.3.2), being the starting point of CART. (Tan et al., 2014)

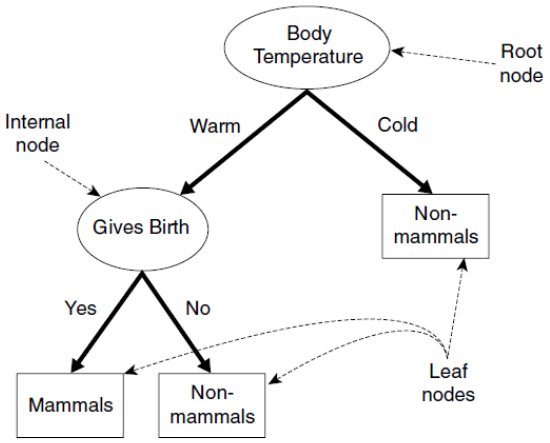


Figure 2.4: Example of a decision tree. Tan et al. (2014)

Hunt's Algorithm

Hunt's Algorithm builds a decision tree in a recursive way, by dividing the training records into successively purer subsets. For each node, an *attribute test condition* is selected to partition the records into smaller subsets. In order to select the attribute test condition, a measure for the best split must be applied.

In regression problems, decision making for partitions is aided by sum of squared errors metric. For a regression tree, the sum of squared errors is as follows:

$$S = \sum_{c \in nodes(T)} \sum_{i \in C} (y_i - m_c)^2, \text{where} \tag{2.1}$$

$$m_c = \frac{1}{n_c} \sum_{i \in C} y_i \tag{2.2}$$

Growing a decision tree follows a 4-step process:

1. Calculate m_c and S for a single;
2. Search over all independent variables for the binary split that provides the greatest reduction in S;
3. If said reduction would be less than a pre-established threshold or one of the resulting nodes would contain less than an n amount of points, usually under 5% of total training records, stop the process before making the split. Otherwise, take the split creating two new nodes;
4. In each new node, repeat the process from step 1.

Hunt's algorithm works for data sets where every combination of attributes is present and has an unique value. However, this assumptions are rarely met, so additional conditions to handle these situations are needed. In case of a node with no associated records, it is declared as a leaf node with the same class label as the majority of the records of the node immediately above. In case all the records associated with a node have the exact same attributes, the node is declared as a leaf node with the same class label as the majority of the records in the same node.

Classification and Regression Trees (CART)-split criterion

Classification and Regression Trees (CART)-split criterion originated from the CART program proposed by Breiman et al. (1984), is a method to assist the construction of decision trees, widely known for its application on Random Forests algorithm. At each node, the best cut is selected by optimizing the CART-split criterion, either based on the *Gini Impurity*, for classification, or the Prediction Squared Error, for regression. In regression problems, in each node it is calculated the best split of every attribute; after that, the best cuts of each attribute are compared to determine the best of the best cuts, to define the split.

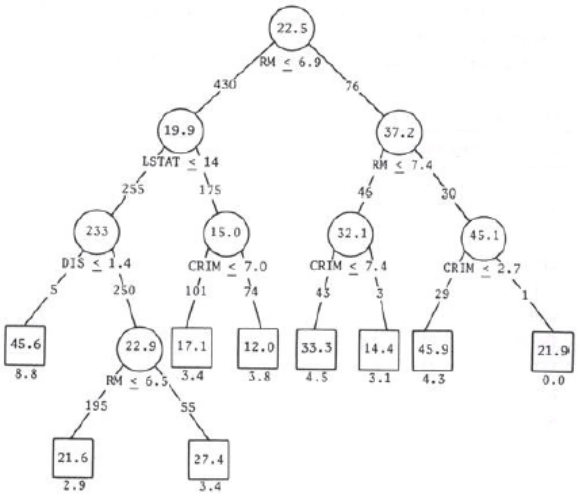


Figure 2.5: Example of a decision tree for a regression problem. Breiman et al. (1984)

2.3.2 Random Forests

Random Forests, as first proposed by Breiman (2001), is a successful general-purpose supervised classification and regression method. The method creates several randomized decision trees and aggregates their results by averaging. Random Forests show good results in sets with more variables than observations, it is versatile enough for large-scale problems and is easily adapted to *ad-hoc* learning tasks. A simple example of the process for creating random forests is presented in figure 2.6. (Biau and Scornet, 2016)

This algorithm operates according to a simple, yet effective motto: "divide and conquer"; fraction data in samples, create a Decision Tree for each sample and aggregate the different predictions. Aside from being simple in its concept and overall versatile for a wide range of problems, it is widely recognized for its accuracy. (Biau and Scornet, 2016) As Howard and Bowles (2012) sate *"ensembles of decision trees often known as random forests have been the most successful general-purpose algorithm in modern times".*

Random Forests are supported by two critical mechanisms: *Bagging*, introduced by Breiman (1996a), and CART-split criterion, presented in section 2.3.1.

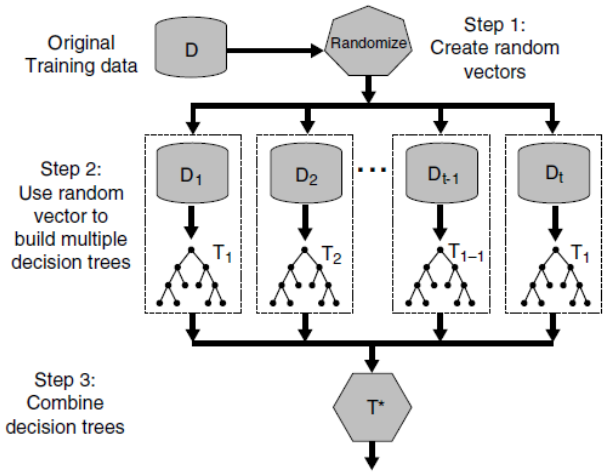


Figure 2.6: Random Forests. Tan et al. (2014)

Bagging

Bagging, contraption for bootstrap-aggregation, generates bootstrap samples from the data set, builds a predictor for each sample, and decides by averaging. It is most effective in large-scale projects where finding a good model in one step is nearly impossible, as shown by Bühlmann and Yu (2002); Kleiner et al. (2014); Wager et al. (2014).

Taking a learning set $\mathcal{L}$, consisting of data $\{(y_n, x_n), n = 1, ..., N\}$, where y is either the class label or numerical response, one can take a procedure to create a predictor $\varphi(x, \mathcal{L})$. Imagining a given sequence of learning sets $\{\mathcal{L}_k\}$, each consisting of N independent observations from the same distribution, N different predictors could be built, $\{\varphi(x, \mathcal{L}_k)\}$. If y is numerical, a straightforward procedure is to take the average of all the predictions as the aggregate prediction, $\varphi_A(x) = E_{\mathcal{L}}\varphi(x, \mathcal{L})$. If y is a class label, the method of aggregation is by voting, i.e. the aggregate prediction is the most voted class label, $\varphi_A(x) = \text{argmax}_j N_j$. (Breiman, 1996a).

Most times, there is not more than one training set; however, this process can be replicated with bootstrap samples of $\mathcal{L}$, $\mathcal{L}^{(B)}$, following the same process for aggregate predictions. More on bootstrapping can be can be found on Efron and Tibshirani (1993).

The term "Random Forests" might be ambiguous. While some authors take it as a generic expression for the aggregation of random trees, others refer only to the specific algorithm firstly proposed by Breiman (2001). Other ensemble methods were proposed such as AdaBoost by Freund and Schapire (1997) and Arcing by Breiman (1996b). In this section, the term Random Forests refers to the original algorithm as an introduction to the subject. However, in following sections, when using a different algorithm, the use of the term is clarified.

Random Forests mechanism carries with it a generalization error, by building predictors based on subsets of data. This generalization error has been theoretically studied and proven to be limited when the number of trees is sufficiently large. (Tan et al., 2014)

2.3.3 Support Vector Machines (SVM)

Support Vector Machine (SVM), firstly introduced by Vapnik (1995), is one of the most common classification and regression techniques. With roots in statistical learning theory, this technique presents good results for high-dimensional data, and represents the decision boundary using a subset of the training examples, known as *support vectors*. (Tan et al., 2014) More on statistical learning can be found on Christmann and Steinwart (2008).

To better understand SVM it is necessary to first comprehend the idea of *Maximum Marginal Hyperplane*. Picture 2.7 illustrates a data set of observations of two different classes, represented as squares and circles. Such data is linearly separable, meaning there is at least one hyperplane capable of fully separating both classes.

In the example shown, there are infinitely suitable hyperplanes; however, the predictor must choose one of the suitable hyperplanes, to represent its decision boundary. The test made is for the

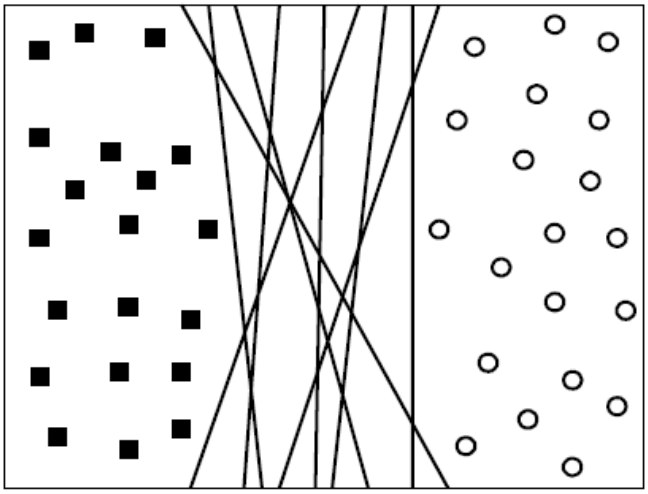


Figure 2.7: Example of linear decision boundaries. (Tan et al., 2014)

maximum margin; to illustrate this, and considering the same example, for each hyperplane, two parallel hyperplanes are drawn and moved away from the decision boundary until each touches one of the classes, as picture 2.8 shows. The margin for said decision boundary is the distance between both hyperplanes. Decision boundaries with better margins tend to have better generalization errors.

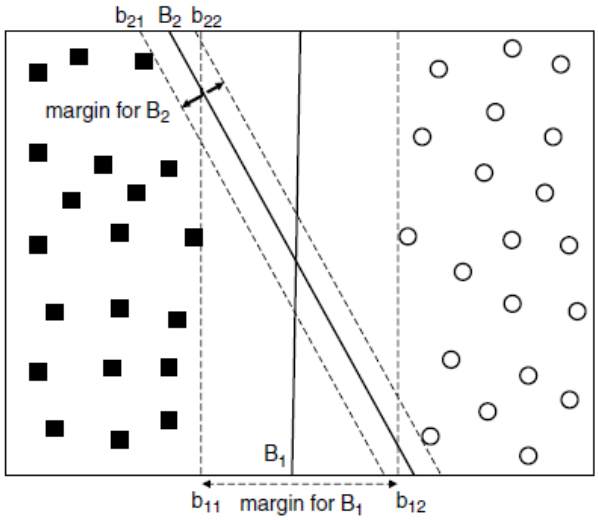


Figure 2.8: Example of margin for a decision boundary. (Tan et al., 2014)

A more formal explanation for this concept is a statistical principal known as *Structural risk minimization*. Such principles provide an upper boundary to the generalization error of a classifier, based on its training errors, number of training observations, and model complexity.

Decision boundaries can be grouped into linear and nonlinear. Linear SVM seeks for a hyperplane with the greatast margin possible, the reason why it is often called a maximum margin classifier. In cases where a linear decision boundary can not be applied, it is necessary to transform the coordinate system into a new one so that a linear decision boundary can be applied. This solution might seem not efficient for high-dimensional data, for which it is applied the *kernel trick*. Christmann and Steinwart (2008) More on this subject can be found on Schölkopf and Smola (2018). After the transformation, the problem can be solved with linear SVM. An example is presented to better understand linear SVM.

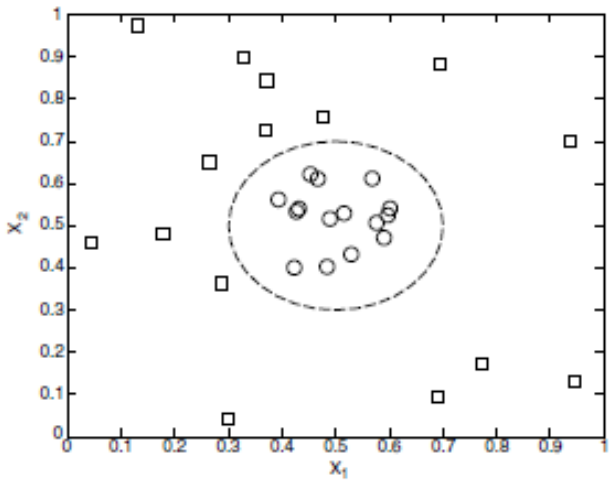


Figure 2.9: Example of a nonlinear decision boundary. (Tan et al., 2014)

Linear SVM

Linear SVM purpose is to find a hyperplane with the maximum margin; considering a binary classification problem with N training examples $(\mathbf{x_i}, y), i = 1, 2, ..., N$, where ($\mathbf{x_i}$ represents the attribute set of the i^{th} example. Let $y_i \in \{-1, 1\}$. The decision boundary can then be described as:

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{2.3}$$

where $\mathbf{w}$ and b are the parameters of the model. The class label y can then be predicted by:

$$y = \begin{cases} 1 & , \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ -1 & , \text{if } \mathbf{w} \cdot \mathbf{x} + b < 0 \end{cases} \tag{2.4}$$

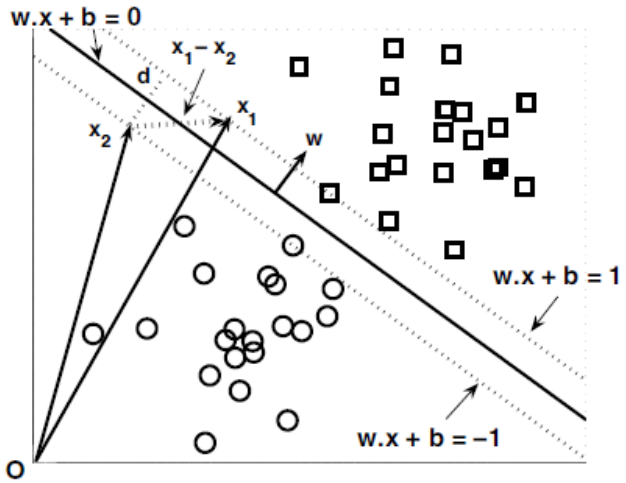


Figure 2.10: Example of a linear SVM. (Tan et al., 2014)

The two parallel hyperplanes in figure 2.10 represented the margin of the decision boundary. It is possible to rescale the parameters $\mathbf{w}$ and b so that the parallel hyperplanes are represented by:

$$b_{i1} : \mathbf{w} \cdot \mathbf{x} + b = 1 \quad b_{i2} : \mathbf{w} \cdot \mathbf{x} + b = -1 \tag{2.5}$$

From here, the margin of the decision boundary can be computed as:

$$d = \frac{2}{||\mathbf{w}||} \tag{2.6}$$

As SVM imposes that the margin must be maximized, it can be mathematically described as followed.

$$\begin{aligned} \min_{\mathbf{w}} &= \frac{||w||^2}{2}, \\ \text{subject to} \quad & y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1, i = 1, 2, ..., N \end{aligned}$$

Such formulation can be solved using the *Lagrange multiplier*. (Tan et al., 2014) More on this can be found on Tan et al. (2014); Christmann and Steinwart (2008).

The present example perfectly separates the class labels; this particular case is called *separable case*. When this is not possible, a method known as *soft margin* should be applied. (Tan et al., 2014)

Support Vector Regression (SVR)

Support Vector Regression outputs a continuous value, instead an output from a finite set, as in classification. To adapt SVM theory to SVR it is introduced a ε-insensitive region around the function, called the ε-tube. This tube reformulates the optimization such as it now seeks the tube that best approximates the continuous-valued function. The complex mathematical formulation of this subject can be found on Awad and Khanna (2015).

2.3.4 Artificial Neural Networks (ANN)

As Tan et al. (2014) states, artificial neural networks (ANN) aim to simulate biological neural systems. The human brain is structured in nerve cells called *neurons*, linked between each other with strands of fiber called *axons*, which transmits impulses from one neuron to another, when stimulated. Neurons connect to axons through *dendrites*; the linking point between a neuron and an axon is called a *synapse*. Neurologists found human brains learn by changing the strength of the synaptic connection upon repeated stimulation from the same impulse.

Likewise, in Artificial Neural Networks, the presence of weights between nodes simulates the strength of a synapse. Each input to a neuron is scaled with a weight which affects the function computed in said unit.

Learning occurs when training data is fed into the model. A set of variables functions as inputs, which are propagated through neurons with weights as intermediates. The computing function then yields an output value. Said output value is compared to the actual value, and weights are adjusted accordingly.

Models are trained by repeating this process, aiming to reduce prediction error. Eventually, function will be able to correctly predict values for the output variable only given its respective inputs values.

Alike decision trees, studied in section 2.3.1, ANN are composed of nodes, often called *neurons* (Roiger, 2017) and directed links. There are three types of nodes: (1) input nodes, related to the input layer; (2) hidden nodes, related to hidden layers; (3) output nodes, related to the output layer.

Structures with one or more hidden layers are called *Multi-Layer Artificial Neural Networks*; these are complex structures, where the hidden layers serve the purpose to extract patterns associated with the process or system in study. (Nunes Silva et al., 2017)

Each input node represents an independent variable from the data set $(x_1, x_2, ..., x_i)$, and the output node represents the output variable, y. Each directed link has a *weight*, $(w_1, w_2, ..., w_i)$ representing the strength of the link, emulating the strength of the synaptic connection between neurons; then, the weighted input signals are gathered by the *linear aggregator*, Σ. A fix value, or bias, θ, might be introduced to the function, to properly set the threshold for which the neuron should generate a trigger value for its output. The sum of these two parcels, *Activation potential*, u, feeds the *Activation function*, g, which yields the output value for the output node.

Any ANN with a given set of nodes and weights defines a function from input to output vectors. Just by changing the value of one weight, many different functions can be simulated. Indeed, Hornik

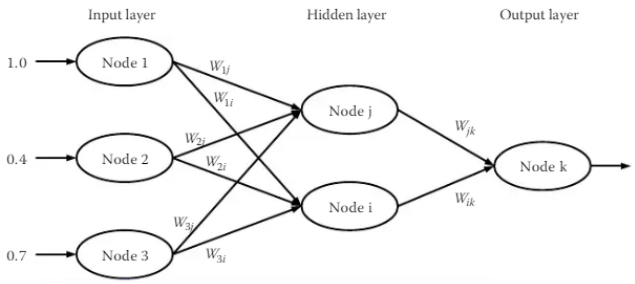


Figure 2.11: A fully connected neural network with one hidden layer. Roiger (2017)

et al. (1989) proved a network with just one hidden layer composed of a sufficient number of nodes is enough to approximate any continuous function.

To better understand ANN, an example of the simplest model, called *Perceptron*, is provided. (Tan et al., 2014; Nunes Silva et al., 2017)

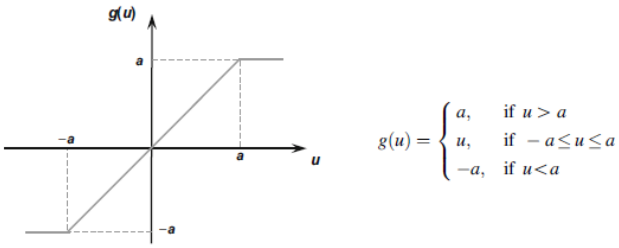


Figure 2.12: Example of an activation function. Nunes Silva et al. (2017)

Perceptron

Figure 2.13 represents the data set and the diagram for a boolean function. Data contains three input variables (x_1, x_2, x_3) and one output variable y. Each input node has the same weight, 0.3. A perceptron computes its output value, $\hat{y}$, by performing the weighted sum on its inputs.

x1	x2	x3	y	ŷ
7	6	3	5	4,8
4	6	0	3	3
3	8	7	5	5,4
8	2	2	3	3,6
3	1	3	2	2,1
3	4	8	4	4,5
1	2	3	2	1,8

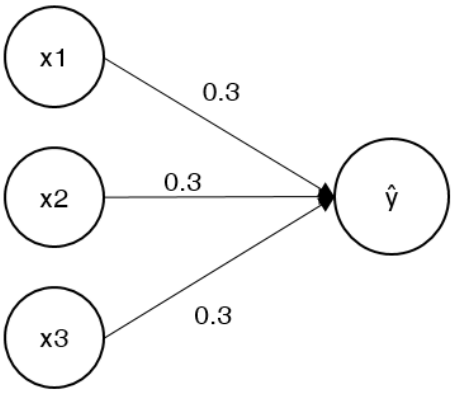
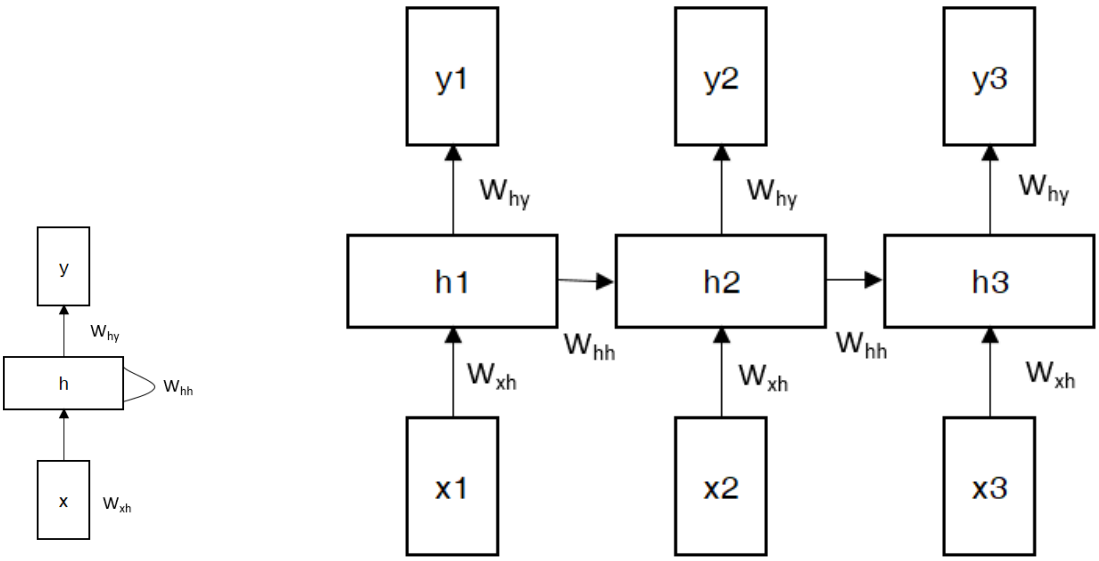


Figure 2.13: Example of a Perceptron.

The output of a perceptron can be expressed as follows:

$$\hat{y} = w_d x_d + w_{d-1} x_{d+1} + ... + w_1 x_1 \tag{2.7}$$



(a) Recurrent Neural Network

(b) Time-layered Recurrent Neural Network

Figure 2.14: Representation of Recurrent Neural Networks

where $w_1, w_2, ..., w_d$ are the weights of the input links $x_1, x_2, ..., x_d$. The difference between the actual value y, and the predicted value $\bar{y}$ is called *generalization error*

In a perceptron, only the output node computes a value. Therefore, all calculations are visible to the user.

Multilayer Neural Networks

More complex neural networks present intermediate *hidden* layers, as computations performed are not visible to the user. (Aggarwal, 2018) Such Neural Networks are denominated Multilayer Neural Networks.

Typically, successive layers feed into another in a sequential fashion, from the input to the output layers. Such specific architecture is a *Feed-forward Neural Network*, as information is passed in a forward direction. In Feed-forward Neural Networks, each node from a given layer is usually connected to every layer of the following layer. Therefore, after the number of layers and neurons per layer is settled, models are almost fully defined.

A drawback of feed-forward networks and its application to sequence predicting is the inability to use contextual information. In a sequence, observations are dependant on one another. Therefore, it is helpful for the model to use information from past observations to process novel data.

From the necessity of storing and using data in a recursive fashion, *Recurrent Neural Networks (RNN)* are developed. These were created for sequential data like text sentences, time-series and discrete biological sequences. RNN take as an input a sequence of past observations of the variables used to predict the output value. RNN models allow inputs to interact directly with hidden information referent to previous inputs.

Figure 2.14a represents a Recurrent Neural Network, where x is the input, y the output and h the hidden representation of past observations. Note to the represented self loop, meaning that the hidden state is updated after the input of each x. This behaviour is shortly described by the expression $h_t = f(h_{t-1}, x_t)$. As RNN mostly handle finite sequences, its representation can be time-layered, as follows on 2.14b.

2.3.5 Long Short-Term Memory (LSTM)

As studied in last paragraphs, Recurrent Neural Networks have the advantage of using contextual information when mapping sequences. However, the range of such contextual information is quite limited. In a Neural Network, the relevance of a given input on the output either decays or blows up exponentially. This event is further described in the literature and it is named *vanishing gradient problem*. More on this topic can be found on Hornik et al. (1989).

Many new models were studied and proposed in the 1990s a way to solve the forementioned problem. (Graves, 2012) One of these models is Long Short-Term Memory (LSTM), proposed by Hochreiter and Schmidhuber (1997).

A LSTM consists of a network of recurrently connected subnets, known as memory block. As exemplified in Figure 2.16, each memory block is composed by one or more self-connected memory cells, followed by three multiplicative gates: input, output and forget. These gates allow memory cells to store and access information over long periods of time, as a way to solve the vanishing problem. LSTM models are fed by sequential data of past observations for increased contextual information. Using the LSTM architecture in a bidirectional recurrent neural network, it allows the usage of long range context in both input directions. (Graves and Schmidhuber, 2005) More on the mathematical formulation of LSTM can be found in Graves (2012); Hochreiter and Schmidhuber (1997).

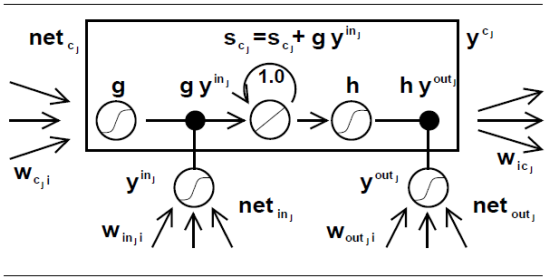


Figure 2.15: Architecture of a memory cell. (Hochreiter and Schmidhuber, 1997)

LSTM models can take one or more inputs, to output one or more variables. This formulates 4 different types of model. (Iliadis and Maglogiannis, 2016)

1. Single Input Single Output, SISO
2. Signle Input Multiple Outpur, SIMO
3. Multiple Input Signle Output, MISO
4. Multiple Input Multiple Output, MIMO

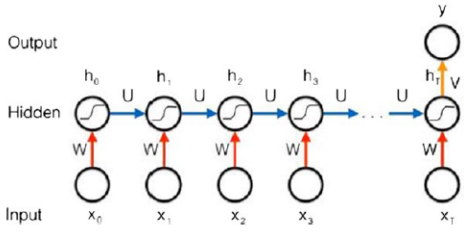


Figure 2.16: Example of multi-input-single-output LSTM. (Manaswi, 2018)

2.4 Structure Health Monitoring (SHM)

As Khazaeli et al. (2016) states, Structure Health Monitoring (SHM) main goal is to assess structural behaviour to ensure the expected level of performance, structural integrity and safe operation

of said structures. The two most common approaches for said goal are physical and data-driven modelling. While the first relies on the application of analytical methods directly relating features to physical parameters and implies a previous deep understanding of the structure, mainly through a physical-based model, the latter deals with statistical features, comparing them to evaluate different states of the structure. Moreover, environmental effects such as temperature fluctuations can be considered by means of data-driven models.

SHM can be seen as the application of Data mining and new data analysis methods to structure control.

The process detecting damage in a structure generally follows the four steps presented below: (Li et al., 2016)

1. *Operational Evaluation* . Describes the purpose of the SHM, relation with economic, security, environmental and operational conditions.

2. *Data acquisition* Describes the steps to follow for obtaining and processing data.

3. *Feature extraction* Determines the relevant signals for damage detection.

4. *Statistical modelation* Creates a model of structure behaviour.

Even tough visual inspection is still the primary way of damage detection, more complex methods have been developed. These methods can either be *explanatory* or *time-series based*. Explanatory models use one or more variables, i.e. explanatory variables, to predict structure characteristics, i.e. dependent variables. Time-series based models aim to find a trend in structure characteristics over time or model a relationship between variables.

Health monitoring can be sorted into two main groups, based on its approach. On a simpler way, the *global approach* health monitoring only determine whether or not damage is present in the structure. However, in a more complex detailed view, the *local approach* aims to locate and classify the severity of damage present in the structure. Chang et al. (2003) The *local approach* can be seen as the next step for the global-based SHM stage, first finding damage, and secondly characterizing it.

Chapter 3

Literature Review

Several studies on Structure Health Monitoring and the application of Data Mining techniques on the field have been presented in the past years. Many approaches exist, varying in damage detection algorithm, variables studied, damage location, and so on. The following paragraphs revise different approaches taken, as a way to study common practices in the field.

Firstly, studies are supported by different types of data recorded from different sources. Regarding its source, projects can be divided into four groups:

- Data generated from a virtual structure, for both damaged and undamaged states. (Kim et al., 2003; Kromanis and Kripakaran, 2013; Yan et al., 2005)
- Data sourced from a real structure in an undamaged state, and virtual generation of data for the damage state. (Sousa Tomé et al., 2020; Tomé et al., 2019; Wipf et al., 2007)
- Data gathered from a small size model structure for both undamaged and damaged state (Cross et al., 2011; Kesavan et al., 2008; Tibaduiza et al., 2011).
- Data recorded from a real-life structure for both undamaged and damaged state. (Da Silva, 2017; Reynders et al., 2014; Worden et al., 2012)

The different variables used to support the SHM models depend on the nature of the algorithms used for this purpose. Most algorithms used in SHM studies are based on continuous-value variables. There is a clear preference for vibration, strain and temperature. Other variables, such as flow (Dunia and Qin, 1998; Slišković et al., 2012), displacement (Posenato et al., 2008), impedance (Park and Inman, 2007) are also covered.

Simpler projects work with discrete variables. For example, Alipour et al. (2017) studies bridges load-capacity rating using variables such as climate zone, existence of water and number of lanes of the structure. This aimed at labeling bridges as load posting or not posting trough the application of Decision Trees and Random Forests.

Defined the data sources, studies can follow one of two approaches. On a global approach, data is collected and analyzed to create a model to simulate structure's normal undamaged condition. This model is then used to predict future values, that are compared with the actual recorded values, for abnormal situation detection.

A local approach aims to quantify damage in specific parts of the structure. Following a two-step process, the location of the damage is first determined, followed by the quantification of damage severity. (Lopes et al., 2000).

As for condition monitoring, both explanatory models, based on Data Mining algorithms, and time series analysis, can be applied. Both Classification and Regression Data Mining algorithms can be applied.

In classification methods, variables describing structure's behavior are recorded and used to build a model that labels its behavior as regular or abnormal; after that, said model is fed new unseen data to classify structure's present behavior. (Kurian and Liyanapathirana, 2020). For Classification purposes, the most common algorithms applied are Support Vector Machines (Anaissi

et al., 2017a,b; Barthorpe, 2010), Artificial Neural Networks (Alipour et al., 2017), and Random Forests SHM35.

Regression models are trained to predict dependent variables that characterize structure's behavior, taking as input explanatory independent variables; after that, prediction residuals are analyzed by a damage detection algorithm. The most common Data Mining Regression algorithms are Linear Regression (Phares et al., 2013), Autoregressive Models (Park and Inman, 2007), Artificial Neural Networks (Lopes et al., 2000), Support Vector Machines (Anaissi et al., 2017a) and Random Forests (Qifeng et al., 2013). As for damage detection, studies tend to apply Hotelling T^2, EWMA and Square Prediction Error (SPE) Control Charts and Moving Fast Fourier Transform.

Time series analysis models predict variables that define the structure's behavior by studying a set of past observations, without the need of introducing independent variables. The application of said algorithms follows a process similar to Regression Data Mining algorithms. The most used time series analysis algorithm is cointegration (Tomé et al., 2019; Wipf et al., 2007; Worden et al., 2012). Alike Regression algorithms, Time series analysis are usually followed by Hotelling T^2, SPE and X-bar Control Charts.

Besides explanatory and time series analysis, Principal Component analysis is commonly applied (Da Silva, 2017; Dunia and Qin, 1998; Posenato et al., 2008; Reynders et al., 2014; Tibaduiza et al., 2011).

The current uprising of Structure Health Monitoring projects come with the need of developing the field with better techniques and optimizing the already existing technology. Regarding the sensors used for data recording, Yi et al. (2015); Zi-yan et al. (2006) present studies on the optimal sensor placing; Satoru et al. (2014); Kumar and Hossain (2018); Cheng et al. (2018) present the development on new technology on sensors for common use, while Boukabache et al. (2011) develops new sensors for aeronautics structures; Aygun and Gungor (2011); Dragomirescu et al. (2010); Hamasaki (2013); Ge (2017) develop wireless technology for new upcoming sensors. Regarding wireless, internet technology and cloud computing, new projects, such as Khazaeli et al. (2016); Li et al. (2016) aim for a better implementation of these tools.

Chapter 4

Data and Methodology

A real-life structured is used to support the development of SHM models. With the main goal of creating algorithms with the purpose of detecting damage in structures, data is collected for over a year, gathering not only data from a normal situation, but also data from a simulated damaged scenario. Following a KDD methodology, project is divided in 6 steps:

1. Structure and Data Retrieval
2. Data Cleaning
3. Data Transformation
4. Data Mining
5. Evaluation and Interpretation
6. Decision Making

Firstly, in Structure and Data Retrieval, structure is presented alongside relevant data gathered. After that, Data Cleaning process removes outliers and corrects values. With data cleaned, it is now ready to be transformed, being divided with subgroups for different operations and reorganized in order to match the correct input for each model. Variables are distinguished between dependent and independent; independent values are fed into the models to predict dependent variables, from which relevant information is expected to be extracted.

Transformed data is fed into different regression models for Data Mining process, where models are trained to predict a normal undamaged condition of the structure. As all data is then predicted, models should fail to predict values for a damaged condition of the structure, indicating said abnormal situation.

In Evaluation and Interpretation, predictions' residuals are evaluated to assess model performance and control charts are build to extract information of abnormal situation in the structure. With relevant information extracted, damaged detection rules are constructed, evaluated and compared in Decision Making step.

The following section present each step in greater detail.

4.1 Structure and Data Selection

The real-life structure under study, as represented in picture 4.1, is constituted by two iron beams in a vertical displacement, being the lower one the thickest. Serving as a partial representation of a bridge, the structure is equipped with electrical resistance extension meters, accelerometers, inclinometers, displacement transducers, GPS and thermometers. In total, data is collected from the structure by four accelerometers, seven thermometers, four strain gauges, two displacement transducers and three inclinometers, making a total of twelve variables. Of the seven thermometers, six gather temperature information from the structure, while one retrieves the surrounding ambient temperature.

Data is collected every 15 minutes, from April 2018 to May 2019. Its aim is to collect data from both an undamaged and damaged situation; to simulate a damaged condition, the structured is connected to a concrete building, which has an expansion joint, where it is planted, through a steel cable. Thermal contraction and expansion of the building applies diferent tensions to the structure, via the steel cable. Such varying tensions applied to the structure are taken as an anomaly, representing the damaged state. The damaged state was simulated starting 10/09/2018 until the end of the data retrieval process.

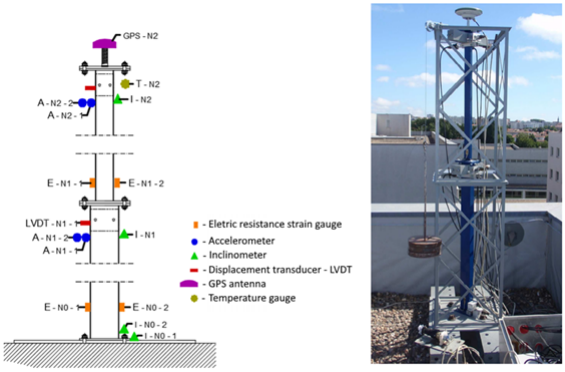


Figure 4.1: Visual representation of the structure. Source:(Rodrigues, 2020)

4.2 Data Cleaning

In a strenuous process developed by Rodrigues (2020) data was cleaned and provided ready to use.

Firstly, data collected by accelerometers is discarded, as these sensors only collect data thrice a day. Remaining sensors collect data for the variables listed below:

- Temperature
  - Lower Face 1 Temperature
  - Middle Face 1 Temperature
  - Middle Face 2 Temperature
  - Upper Face 1 Temperature
  - Upper Face 2 Temperature
  - Upper Face 2 Uncovered Temperature
  - Ambient Temperature
- Displacement
  - Middle LVDT
  - Upper LVDT
- Strain
  - Lower Face 1 Strain
  - Lower Face 2 Strain
  - Middle Face 1 Strain
  - Middle Face 2 Strain
- Slope
  - Lower Inclinometer
  - Middle Inclinometer
  - Upper Inclinometer

Presence of outliers is expected due to the use of sensors. In a real scenario, predictors have access to data from both damaged and undamaged periods. As models aim to correctly predict an undamaged state of the structure, outliers are detected and removed from this group of data, so that there are no interferences.

Several methods were studied to remove outliers. The one which yielded better results was comparing the absolute difference between each observation and past and future records. Every observation is compared with the observation immediately after and immediately before. These two differences are tested against the mean of all differences: if the absolute value is higher than the mean and three standard deviation, the record is removed. This iteration correspond to a time gap of 15 minutes; the process is sequentially repeated for time differences up to 1 day, corresponding to a gap of 96 observations. A visual representation is presented in figure 4.2.

Finally, records from the damaged situation were corrected; when damage is induced, observations suffer a shift in recorded values. This value is assumed to be the difference between the means of last 672 observations of the undamaged state and the first 672 observations of the damage state, corresponding to one week of records.

4.3 Methodology

Explanatory models are used to support the damage detection system. Initially, temperature variables are defined as independent variables, and the remaining are defined as dependent variables. As temperature values are collected from several points of the structure and also from surrounding ambient, it is expected that variations between these seven variables can predict the behavior of

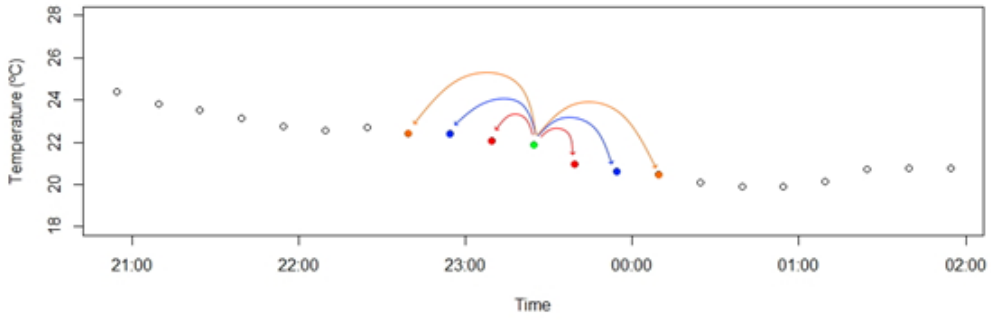


Figure 4.2: Representation of the outlier removal method. Source: (Rodrigues, 2020)

the structure in its different regions. Said behaviour is described by the dependent variables, i.e. displacement, strain and slope.

Dependant variables are predicted by the algorithms, taking the observed values of independent variables as the input. Four different algorithms are applied: Random Forests, Support Vector Machines, a feed-forward Neural Network (ANN), and a recurrent Neural Network (LSTM). Finally, an ensemble model is built using the first three models.

The resultant predictions are compared to its correspondent observed values. The error, or residue, of each prediction can be thought of as the part of the dependent variable that several temperature variables cannot explain. A smaller error, in normal circumstances, means a better ability to predict such variable using temperature. Models are evaluated with MAPE, MSE and R^2.

Anomally detection is achieved comparing predicted and observed values, through control charts. In normal situations, the residual should be conserved within a reasonable interval. However, in abnormal situations, where the structure is damaged, the values observed for dependent variables should be drastically different. Without any significant change in the independent variables, it is expected to see an increasing error in the predictions, as temperature variables can no longer predict the dependent variables correctly. Two different control charts are applied: Hotelling and EWMA.

Predictors should not only have a low amount of false negatives, but also a low amount of false positives, as it is also important not to signal the structure as damaged in normal circumstances. Understandably, models are trained only using data from an undamaged state, as this is the behaviour to be predicted.

The modelation of predictors follows the following stages:

1. Parameter tuning;

2. Final model training using optimal parameters;

3. Variable prediction;

4. Model evaluation.

Data is then divided into damaged and undamaged state. Furthermore, undamaged period is divided into training, validation, testing and prediction data. Such data is divided in a chronological manner: training data is older than validation data, which is older than testing data and so on. This method simulates the process of a live prediction, where data is fed into the model chronologically, as represented in figure 4.3.

Next paragraphs present the procedures in detail. While the procedure for RF, SVR and ANN models are similar, LSTM and Ensemble models follow different methods.

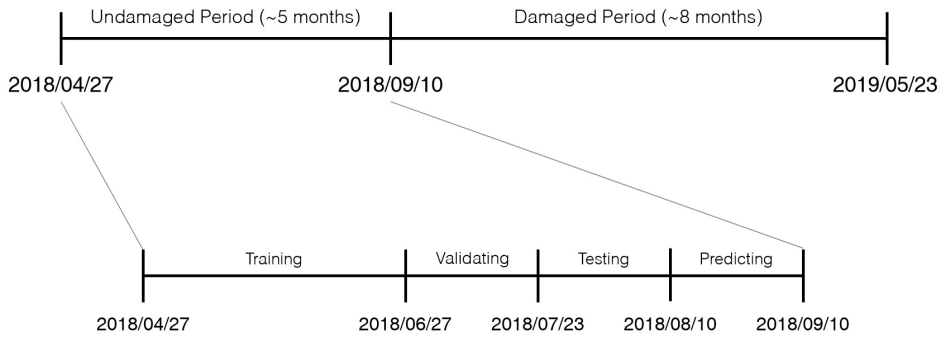


Figure 4.3: Data division.

4.3.1 Model development

Random Forests, Support Vector Machines and Artificial Neural Networks

Procedures for the forementioned algorithms are identical, being those:

- Training data is used to train models during parameter tuning;
- Validation data is used to validate the parameters tested;
- Training and Validation data are then both used to fit the final model;
- Testing data measures the performance of the final model;
- Data from Prediction and Damaged periods are used to compare observed and predicted values.

In the first step, parameters are optimized with training and validating data. Both datasets are divided in ten equal-sized folders and matched in pairs, i.e., one training folder feeds a model which is validated by one validating folder, in order to avoid overfitting. During this process the number of independent variables needed for predicting is tested in Random Forests, while Support Vector Machines optimizes the cost and gamma values and Artificial Neural Networks tunes size and cost values. Models are trained with different parameter's values, being then used to predict the output for the validating period, to which it is fed training data; after this, the model executes predictions for validation data, which are used for model evaluation with MAPE and MSE metrics.

Each set of parameters and its respective metrics is stored and then compared. A final model is built with the best parameters, i.e., the set that enables to obtain the best prediction performance on the validating set. Afterwards, all available data is fed to the new model and testing data is used to evaluate the performance of the model to new unseen data. Predictions and real values are plotted for comparison purposes.

The code for each algorithm can be found in annexes B.1 (RF), B.2 (SVR), B.3 (ANN).

Ensemble Models

Ensemble Models are develop in order to combine predictions from the 3 forementioned algorithms into one final prediction, as a possibility of reducing prediction error. Two different ensemble models are considered: the first one is constructed with a simple average over the predictions of the three models; the second model is built upon a weighted average of the same three predictions, where each one is weighted by the overall prediction error of each algorithm. Procedures for each model can be found in annex B.4.

LSTM

The process for the development of LSTM follows a larger number of more complex steps.

LSTM models are fed sequential data. This means that for each observation of the dependent variable, one or more sequences of one or more steps of past observations of independent variables are fed into the model. This process results in the loss of some of the first observations, for purposes of data sequencing. A practical example is given for a clear understanding.

For a given data set constituted of 10 observations of independent variable x and dependent y, as represented in figure 4.4, data is modelled to attain 1 sequence of 3 steps for each observation. Each sample has three input time steps and 1 output time step. The process results in the loss of the first two observations for a total of 8 observations.

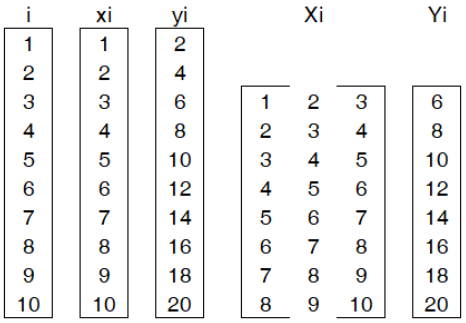


Figure 4.4: Data preparation for LSTM models.

The cleaning process removed observations from the undamaged period, resulting in break points of discontinuities. As LSTM models are fed sequential data, samples can only be created from blocks of uninterrupted observations. These blocks need processing in order to create samples. Time stamps are cleaned: even though data is recorded each 15 minutes, some discrepancies in seconds or minutes might occur; therefore time observations are modelled to be in multiples of 15 minutes. From these organized time stamps, points of discontinuity are signaled, and blocks are built. As the creation of samples requires a minimum of sequential observations, blocks are selected based on the number of steps and sequences of each sample fed to the model; the greater the number of past observations used in each sample, the least number of data blocks can be used. Figure 4.5 represents the number of continuous past observations for each record of undamaged period. Models fed with samples with 10 or more past observations result in a significant loss of data. As damaged period did not suffer outlier removal process, only few break points occur due to sensors malfunction, with minor impact to the sample creation process.

Code for data preparation for LSTM models can be found at annex A.2.

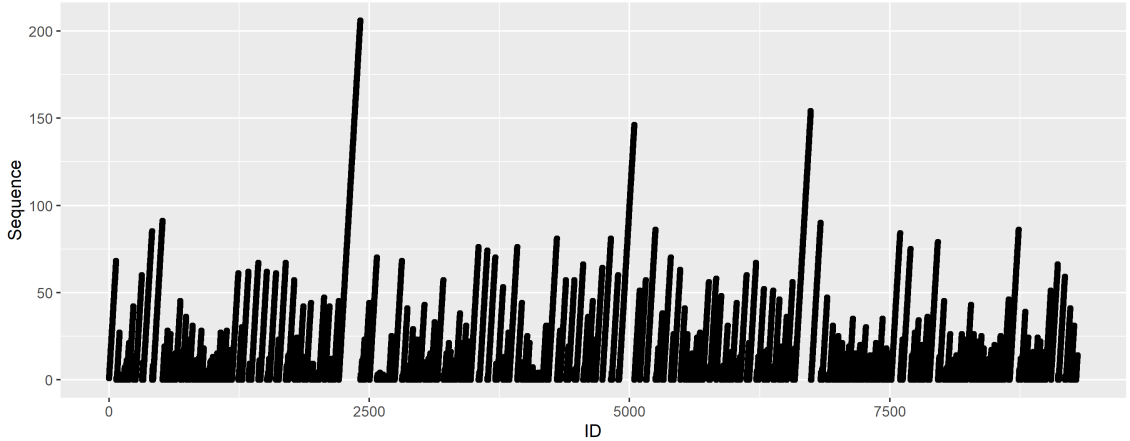


Figure 4.5: Number of continuous past observations in a row for each record.

As the number of samples fed to the model depends on the number of past observations used for each sample, model training needs to take that variable into account, besides overall model performance. Different combinations of steps and sequences are evaluated using a simple 1 convolutional and 1 LSTM layers. After testing, it is defined that models should be fed samples with 3 sequences of 3 steps, as it is the best combination of model performance and least data loss.

After data preparation, several LSTM models were considered. These models are constituted by a stacked combination of convolutional layers and bi-directional LSTM layers. Models were built considering 1-5 convolutional layers and 1-5 LSTM layers. Models are trained with training data and evaluated with validation data. After 4 convolutional layers and 3 LSTM layers, results are not significantly better so that it is worth the additional computational effort. This model is therefore refered as Convolutional Bi-directional LSTM (CBLSTM), or simply LSTM.

Training and validating data are not divided to train LSTM models, as the reduced amount of training samples and increased complexity of the model could result in poor model training.

As the final model is defined, it is trained using training and validation data, evaluated with testing data and all data is then predicted. Predictions and real values are plotted for comparison purposes. Both single and multiple output LSTM models are constructed under same definitions.

Code for CBLSTM single and multiple output models can be found in annexes B.5 and B.6, respectively.

Model Evaluation

With models finally developed and predictions made, each algorithm is evaluated with MAPE, MSE and R^2 metrics. After this, control charts are applied. Two different control charts were considered: Hotelling T^2 and Exponentially Weighted Moving Average (EWMA). Both can be used in multivariate datasets and, therefore, can evaluate the predictions of every dependent variable all together. Hotelling T^2 takes into account the mean of each individual residual and the matrix of covariances between each pair of residuals; in its turn, EWMA takes into account a moving average of residuals, weighted such as the more recent moving averages have more representativeness. Both control charts are a two-phased quality control chart: in the first phase, a dataset is fed to set the base parameters and calculate an Upper Control Limit; in the second phase, testing data is fed and compared to the parameters set in phase I. For this project, phase I is fed with training and validation undamaged period, while phase II is fed with data from testing and prediction undamaged period as well as damaged period . Points above the UCL indicate an abnormal behavior of the structure compared to other observations.

The two different types of control charts are evaluated based on the number of false positives and false negatives. The best performing control chart is then used for damage detection rules.

Code for both Control Charts developing can be found in Annex C.

Damage Detection

To signal damage in the structure, information from control charts needs to be analyzed. Two different rules are studied and applied.

Firstly, a simpler rule is considered: percentage of points above UCL in a moving window. 3 different window widths were considered: 4, 8 and 12 observations, corresponding to 1, 2 and 3 hours, respectively. Different percentages of points out of control are also considered, ranging from 50 to 75 percent.

A second, more complex rule is also considered: Cumulative Sum Control Chart (CUSUM). In a nutshell, CUSUM monitors change in a sequence; it involves the calculation of a cumulative sum, which is for a drastic change over previously defined limits.

Each observation is normalized, using the mean and standard deviation calculated with training and validating data. With X being the observations value, relative to the control chart, $\bar{x}$, observations' average and σ observation's standard deviation, calculated from training and validating data, observations are normalized as such:

$$Z_n = \frac{X_n - \bar{x}}{\sigma} \tag{4.1}$$

where Z_n is the normalized value of observations, fed to the model. The cumulative sum is then calculated following equation 4.2

$$\begin{cases} S_0 = 0 \\ S_{n+1} = S_n + Z_n - \omega_n \end{cases} \tag{4.2}$$

where ω represents the weight assigned to each observation, for sensitivity optimization.

After a predefined threshold, measured in standard deviations is surpassed, either positively or negatively, a change in value is signaled, corresponding to damage detection, in the current project.As the present project studies the increase in points out of control, only a positive anomaly is relevant. Therefore, the cumulative sum equation is changed as such:

$$\begin{cases} S_0 = 0 \\ S_{n+1} = \max(0, S_n + Z_n - \omega_n) \end{cases} \tag{4.3}$$

Two different inputs were considered and the results of each compared: a binary sequence representing whether each point is above or below the UCL, and the output of the selected control chart.

Code for both Damage Detection Rules can be found in Annex D.

Chapter 5

Results

5.1 Model Performance

As described before, in order to detect anomalies in the a structure, it is proposed to predict the behaviour of the structure in a regular condition. The prediction models are constructed using the training data and optimized based on the validation set. For each set of parameters tested performance metrics are estimated, in order to choose the best set of parameters for the model. After this, the final model is constructed and final predictions are calculated for unseen data from undamaged and damaged periods. Predictions referent to undamaged testing period data are compared to real values through performance metrics, such as MAPE, MSE and R^2, applied to undamaged testing data, and visual representation.

It is expected that models fail to accurately predict values the monitored variables' values for the damaged period.This fact may be used to correctly detect damage through a significant increase in residuals. Visual representation effectively demonstrates the increase in predictions' residuals.

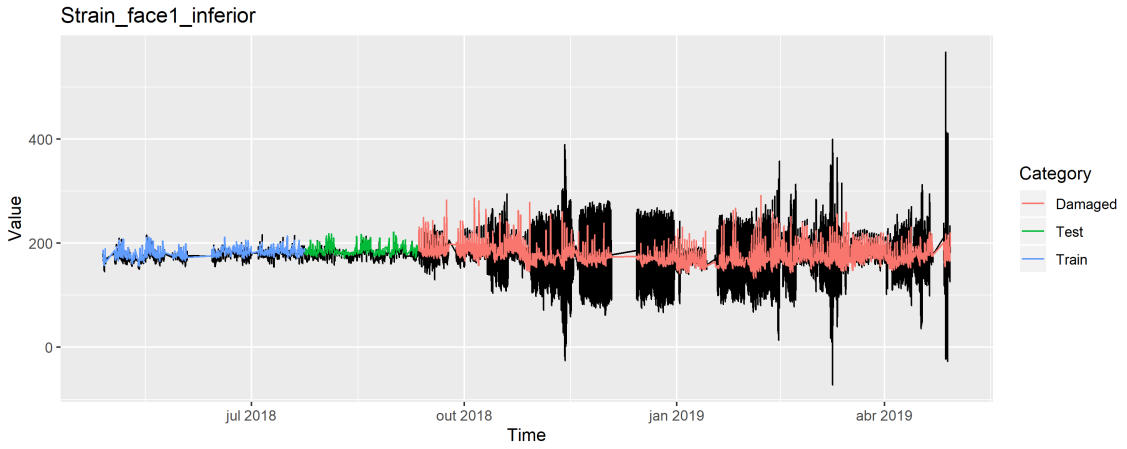


Figure 5.1: LSTM single predictions for Lower F1 Strain.

Figure 5.1 represents predictions for Lower F1 Strain, calculated by LSTM single output, compared to actual values. It is possible to observe that the model predicts accurately values for undamaged period, failing, however, to predict values for damage period. Graphics for each variable for each model can be consulted in annex E.

Table 5.1 presents, for each model, the MAPE estimated base on the testing data. Overall, percentage errors are conserved under 4% for all variables, except for Middle LVDT, Upper LVDT and Lower F2 Strain. As these variables present abnormally large errors for all models, over 10%, they are posteriorly ignored; for this reason, these variables are not predicted in the LSTM multi-

Table 5.1: MAPE for undamaged test data for models constructed

MAPE (test)	Lower Inclinometer	Middle Inclinometer	Upper Inclinometer	Middle LVDT	Upper LVDT	Lower F1 Strain	Lower F2 Strain	Middle F1 Strain	Middle F2 Strain
RF	0,0057	0,0075	0,0077	0,6071	0,1103	0,0280	0,1153	0,0233	0,0364
SVR	0,0057	0,0068	0,0082	0,6965	0,1431	0,0269	0,1059	0,0243	0,0364
ANN	0,0057	0,0058	0,0051	0,3857	0,0763	0,0263	0,1203	0,0231	0,0364
Averaged	0,0053	0,0061	0,0069	0,5745	0,1158	0,0251	0,1039	0,0225	0,0348
Weighted	0,0053	0,0061	0,0068	0,5700	0,1148	0,0250	0,1039	0,0225	0,0348
LSTM single	0,0071	0,0096	0,0042	0,3909	0,0959	0,0264	0,1032	0,0230	0,0267
LSTM multi	0,0103	0,0089	0,0102	-	-	0,0248	-	0,0194	0,0291

variable model. In order to compare models, the average error for the six variables considered relevant for the study is calculated, as presented in table 5.2.

With the presented results it is concluded that, as expected due to its relative simplicity, Random Forests and Support Vector Regression perform worse, compared to other algorithms; Ensemble models performed better compared to the three models upon which they were built, with Weighted Ensemble performing slightly better than Averaged Ensemble. ANN and LSTM models are part of the top performers. As the extension of the number of steps and the number of sequences is harshly restrained by the data blocks created by the discontinuities present due to the outliers removal, potentially better LSTM models could not be tested and applied. Despite this, LSTM presented overall the best results, with LSTM single model having the best performance of all models considered.

As all models average error is conserved within a small range, control charts are built for each model, for further comparison.

Table 5.2: Average MAPE for each model constructed

MAPE (test)	RF	SVR	ANN	Averaged	Weighted	LSTM single	LSTM multi
Average	1,812%	1,806%	1,707%	1,679%	1,675%	1,616%	1,713%

5.2 Control Charts

In order to detect anomalies, values predicted are compared to actual values, calculating prediction residuals. These residuals are fed both to Hotelling T^2 and EWMA control charts. Both control charts signal points out of control when residuals are relatively higher than expected. Such points reflect an inability of the model in study to accurately predict values for the dependant variables, meaning that an abnormal situation might be occurring.

It is expected that control charts identify few points out of control, i.e. above the UCL, for undamaged period data and signal a high number of points out of control for damage data. Points out of control in undamaged period data correspond to false positives. A high amount of false

positives reflects poor prediction ability of the model, resulting in unnecessary maintenance and overall discredit of the model. False negatives on undamaged period data correspond to an inability of the model to identify abnormal situation, which can result in lack of maintenance, leading to irreversible damage to the structure and possible human and economical loss.

Figures 5.2 and 5.3 present control charts made for ANN model undamaged period data. These reflect what is observed to all models, i.e. the models constructed to model each variable. Hotelling T^2 control charts present a lower amount of false positives than EWMA ones. For this reason, EWMA control charts are discarded, meaning that further procedures are solely applied to Hotelling's charts. Control Charts for the different prediction techniques can be found in Annex F.

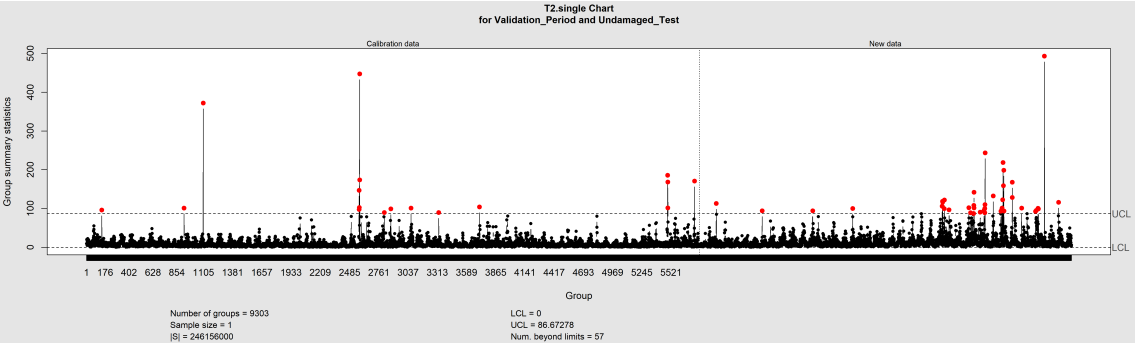


Figure 5.2: Hotelling T^2 control chart of undamaged period data for ANN model.

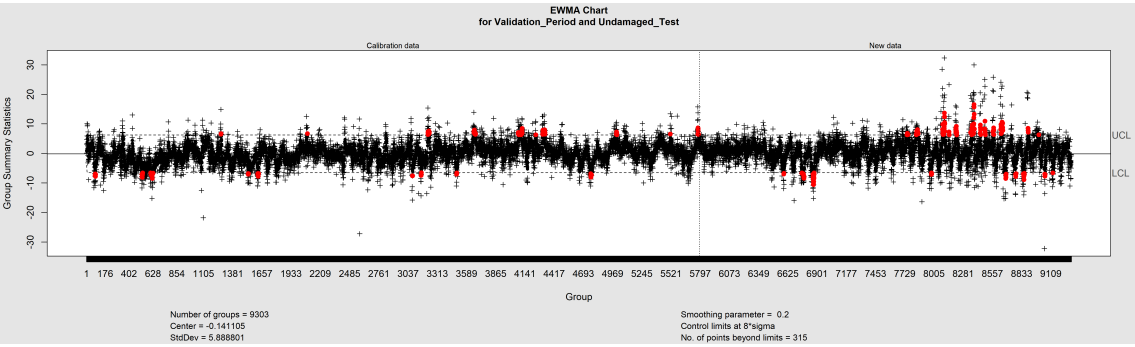


Figure 5.3: EWMA control chart of undamaged period data for ANN model.

Table 5.3 presents statistics for Hotelling T^2 control charts built for every model. These reflect the percentage of points out of control for each group of data. Validation group refers to train and validation data, upon which control charts are built and Testing group refers to test and prediction data and new unseen data from the undamaged situation; points out of control in these groups correspond to false positives. Damaged group refers to data from the damaged period and points out control correspond to true positives. Therefore, it is expected that control charts present low amount of percentage points over the UCL for Validation and Test groups, and a high amount of percentage points over the UCL for Damaged group.

Even though all models presented similar prediction error, control chart results clearly distinguish models apart.

As expected, all models present low percentage for Validation group, since this data is used for control chart tuning. However, Neural Networks, ANN and both CBLSTM models, present lower percentage than the remaining models.

The combination of Testing and Damaged groups statistics has the potential to reveal the ability of a control chart to successfully detect abnormal situations while not signaling normal situations as abnormal too.

Even though RF, SVR and both Ensemble control charts present high percentage of points out of control for data from the damaged period, they also reveal a high percentage of false positives

in testing data. This reflects an overall inability of the control chart to correctly identify regular situations. Ensemble models' results are more solid than RF and SVR, with significantly lower false positives while preserving a high amount of true positives; however, these results are not solid enough when compared to other models.

Neural Networks models present control charts with low amount of false positives, between 0.79 and 1.20 %, while maintaining a high number of true positives in data from Damaged period, between 40 and 60%.

Table 5.3: Hotelling T^2 control chart statistics for all models constructed

Model	RF	SVR	ANN	Averaged	Weighted	CBLSTMsingle	CBLSTMmulti
Validation	0,74%	0,71%	0,29%	0,55%	0,52%	0,14%	0,44%
Testing	31,80%	30,43%	1,20%	9,19%	9,25%	1,23%	0,79%
Damaged	91,47%	91,54%	40,26%	73,59%	73,37%	54,32%	59,27%

After this process, RF, SVR models are discarded. As Ensemble models reveal almost identical results, only Weighted Ensemble is taken into account on the next procedures. All Neural Networks models are considered for further steps.

Code for control charts elaboration can be found in Annex C.

5.3 Damage Detection

After abnormal situations are signaled, damage detection rules are necessary, in order to filter through such points to determine if these points are indicative of damage in the structure.

Two damage detection rules are applied and tested, being the first one the percentage of points out of control (PPOC) on a moving window, and the second one the application of CUSUM. As long as both rules indicate when the structure is eventually damaged, CUSUM additionally indicates the moment when significant change happens. Therefore, both PPOC and CUSUM are evaluated by the number of false positives and false negatives, and CUSUM is also evaluated considering the time it takes to first detect damage in the damaged situation.

Rules are applied to data from Testing and Prediction undamaged period alongside to damaged period.

5.3.1 Percentage Points Out of Control

Two aspects of Percentage Points Out of Control approach can be tuned, i.e. the range of the window and the percentage threshold. Therefore, different combinations are applied and tested for the amount of false positives and false negatives, in a process similar to control charts evaluation.

Three window sizes are tested: 1, 2 and 3 hours, corresponding to 4, 8 and 12 observations, respectively. Percentage thresholds vary from 50% to 75%.

Naturally, an increase on the percentage threshold should result in the decreasing of overall signaled points, both for undamaged and damaged period data, decreasing the number of false positives yet also decreasing the ability of the rule to generate true positives.

An increase on window size should result on a decrease of false positives on undamaged period data, as the signaled points out of control in this group are expected to be punctual and sporadic. This increase on the window size may not have a huge effect on the number of true positives on data from damaged group, as the signaled points out of control are expected to be recurrent. The increase of the window size also increases the time for damage detection after an initial point out of control is signaled, slowing the eventual response to a possibly damaged structure.

Results are presented in table 5.4.

From presented results it is confirmed that increasing the percentage threshold results in a decrease of signaled instances for both undamaged and damaged periods. It is also concluded that

Table 5.4: Percentage points out of control damage detection rule statistics for different window sizes and percentage threshold

Size	%Threshold	Data	CBLSTMmulti	CBLSTMsingle	ANN	Ensemble
4	50%	Test	0,64%	1,38%	0,60%	9,80%
		Damaged	67,10%	62,63%	48,29%	82,14%
4	75%	Test	0,25%	0,69%	0,14%	4,42%
		Damaged	53,71%	45,91%	29,59%	68,70%
8	50%	Test	0,35%	0,79%	0,23%	6,42%
		Damaged	64,71%	59,78%	43,23%	81,12%
8	62%	Test	0,25%	0,69%	0,00%	3,65%
		Damaged	56,29%	48,52%	31,27%	72,88%
8	75%	Test	0,10%	0,59%	0,00%	2,08%
		Damaged	48,69%	38,81%	22,14%	63,65%
12	50%	Test	0,30%	0,79%	0,00%	4,91%
		Damaged	63,80%	58,51%	40,60%	81,49%
12	66%	Test	0,00%	0,59%	0,00%	1,74%
		Damaged	51,51%	42,10%	24,24%	68,46%
12	75%	Test	0,00%	0,50%	0,00%	1,26%
		Damaged	46,19%	35,61%	19,00%	61,16%

increasing the window size reduces the percentage signals for both groups of data, contrary to the expectations, with a more significant reduction on undamaged period.

Regarding differences between models, Ensemble models clearly presents worse results compared to other models tested, as the percentage of false positives is up to 10 times higher. This is somewhat expected, as control chart results are also clearly worse for this algorithm.

Neural Networks leads to similar results for all three models, with a slight advantage for CBLSTM multi output.

Percentage of false positives is affected by the presence of many points out of control contained in a small time window. Increasing the window size reduces the number of false positives solely by increasing detection time until the number of signaled observations contained in said small time window is smaller than the number of signals necessary to surpass the threshold. Therefore, the main difference for the number of false positives in a set of parameters lies on the sole existence of damage detection, rather than the percentage signals.

All things considered, despite outperforming CBLSTM single output in percentage false positives, ANN fails to detect damage as well as both CBLSTM models, falling behind 50% for all combinations of parameters considered. Even though CBLSTM single output presents better results for control charts and prediction error, it is outperformed by CBLSTM multi output both in false positives and false negatives. For these reasons, it is concluded that CBLSTM multi output is the basis for the best overall damage detection, supported by a moving window of 12 observations, i.e, 3 hours, with a percentage signals threshold of 66%.

Even though only 51% of observations are identified as a possible occurrence of damage in damaged period, figure 5.4 reveals that damage is consistently identified throughout time. Therefore, such low amount of true positives is not a determining point to invalidate the model and discredit its damage detection ability.

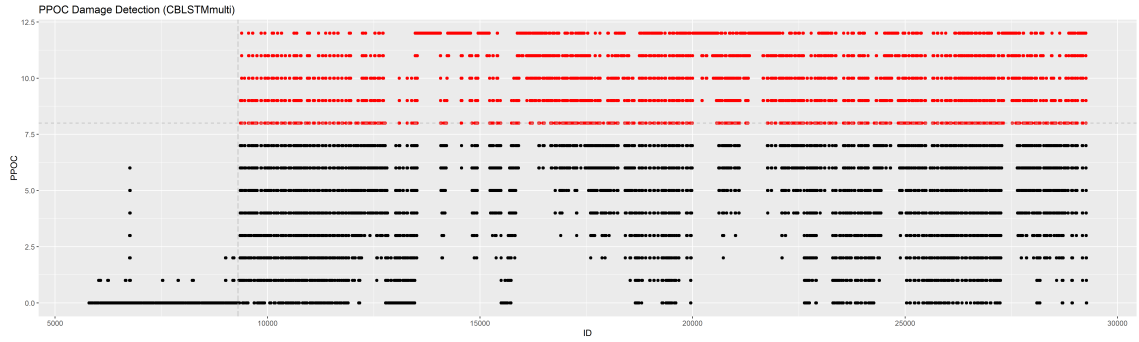


Figure 5.4: PPOC Damage Detection for CBLSTM multi output algorithm.

5.3.2 Cumulative Sum Control Chart

Cumulative Sum Control Chart (CUSUM) monitors change in a sequence. Applied to this project it is developed to detect change in the cumulative sum of points out of control. When such significant positive change occurs, CUSUM signals it as a damage situation.

The goal of this damage detection rule is not to detect when the structure is damaged but to detect when the damage starts. Therefore, the two metrics used to evaluate the rule are percentage of false positives and response time to damage inception.

Inputs are normalized, being the mean and standard deviation calculated with training and validation undamaged period.

Two aspects of CUSUM can be tuned: the threshold for damage detection, measured in standard deviations, and a weight applied to each observation. Both parameters optimize the sensitivity of the model, at the cost of increasing damage detection time. However, while increasing the threshold decreases overall sensitivity, a variation in the weight affects mostly singular cases of points out of control, reflected in the number of false positives.

In a first iteration, a binary sequence demonstrating whether a point is above or below the UCL of the Hotelling T^2 control chart is applied. To better tune the detection damage rule, different weights are applied and tuned. These are calculated as a percentage of the normalized value of a point out of control, i.e., $Z(1) = \frac{1-\bar{x}}{\sigma_X}$. If the weight were $Z(1)$, no damage would be detected; by having a weight of $0.5Z(1)$, each signaled point has its value reduced to half, while keeping a low impact on points in control, improving model sensitivity. Results are as follow on table 5.5. First Damaged reflects how many observations passed until damaged is detected. If 1^{st} Damaged is 2, it means a damage is detected on the second observation, i.e. 30 minutes, after damage is induced to the structure.

From observed results it is possible to conclude that the absence of a weight makes the damage detection rule too sensitive, where a single signaled point on the Hotelling control chart is not only enough to induce the model into detecting damage, but also enough to spread the effect for many following observations. Due to this aspect, damage detection model fails to correctly identify each situation, as most of the undamaged instants of time are wrongfully signaled as damaged.

The introduction of a weight is successful in reducing the amount of false positives to a satisfactory amount, while maintaining a low number of false negatives.

As for each model results, Ensemble model presents the worst damage detection statistics, as expected. Despite satisfactory, ANN results are shadowed by the great response CBLSTM models had to the introduction of a weight. CBLSTM multi is able to keep the number of false positives well below 1% for both weights, still maintaining a high value of true positives, while CBLSTM single was able to keep a fast reaction to damage induction, with a number of false positives just over 2%.

Response time to damage is slowed by the introduction of such heavy weights used to tune down model sensitivity. Additionally, by feeding a binary sequence to the algorithm, information is lost, as the actual value of residuals for each record is not represented.

In hopes of accelerating model's detection of damage, a second iteration is attempted. This time, the input for the damage detection rule is the Hotelling T^2 value for each record. Inspired by

Table 5.5: CUSUM fed by binary sequence damage detection rule statistics for different weights

Model	%Weight	Test	Damaged	1st Damaged
ANN	0%	62,27%	100,00%	1
CBLSTMmulti	0%	85,55%	100,00%	1
CBLSTMsingle	0%	74,46%	100,00%	1
Ensemble	0%	98,49%	100,00%	1
ANN	50%	1,37%	76,18%	10
CBLSTMmulti	50%	0,54%	91,46%	54
CBLSTMsingle	50%	2,17%	93,93%	10
Ensemble	50%	8,99%	99,60%	4
ANN	75%	0,14%	36,61%	74
CBLSTMmulti	75%	0,20%	71,81%	55
CBLSTMsingle	75%	1,08%	61,53%	55
Ensemble	75%	3,41%	93,10%	55

Cumulative observed-minus-expected plots (Grigg et al., 2003), the model is fed a value representative of the residues in the prediction of all six considered dependent variables. Therefore, more information is fed into the algorithm as the actual degree of prediction error is taken into account.

Following a similar process, weights are introduced to the algorithm as a percentage of the normalized value of the Hotelling Chart's UCL. Such value is representative of the Hotelling T^2 baseline value for anomaly detection. In this way, it is intended to make points just slightly out of control, like signaled points in undamaged period, less significant, while maintaining the importance of observations with a high degree of prediction error.

Results are as follow on table 5.6.

Results are relatively similar to ones with the binary sequence approach. However, it is clear that the adoption of the approach based on the residuals has its benefits. While response time did not see any improvement, CBLSTM models enabled to decrease the number of false positives, while increasing the number of true positives.

It is concluded that CUSUM fed by residuals values with a weight of $2 \times Z(UCL)$ presents the best combination of low false positives and high true positives. This approach also enables to keep damage detection time among the most observed value for all models and model setups, with 55 observations, corresponding to 13 hours and 45 minutes.

Response time to damage itself is considerably high; as models presented sensitivity issues, solved by the insertion of a weight, damage detection became more strict. Therefore, the benefit of considerably lowering false positives came with the cost of a slower response to damage.

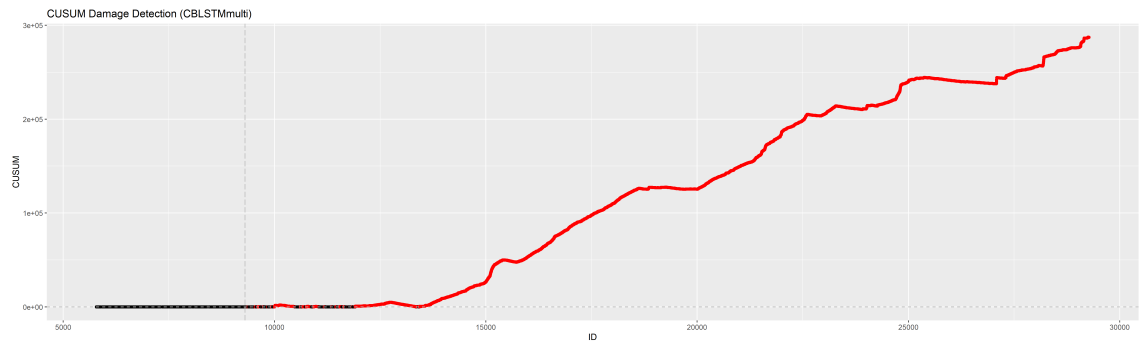


Figure 5.5: CUSUM Damage Detection for CBLSTM multi output algorithm.

Table 5.6: CUSUM fed by residuals value damage detection rule statistics for different weights

Model	%Weight	Test	Damaged	1st Damaged
ANN	0%	93,14%	100,00%	1
CBLSTMmulti	0%	97,53%	100,00%	1
CBLSTMsingle	0%	89,20%	100,00%	1
Ensemble	0%	98,66%	100,00%	1
ANN	100%	0,31%	95,66%	77
CBLSTMmulti	100%	0,69%	99,07%	55
CBLSTMsingle	100%	1,83%	98,79%	10
Ensemble	100%	14,83%	99,80%	3
ANN	200%	0,06%	87,45%	81
CBLSTMmulti	200%	0,10%	92,68%	55
CBLSTMsingle	200%	0,74%	92,45%	80
Ensemble	200%	2,99%	98,65%	10

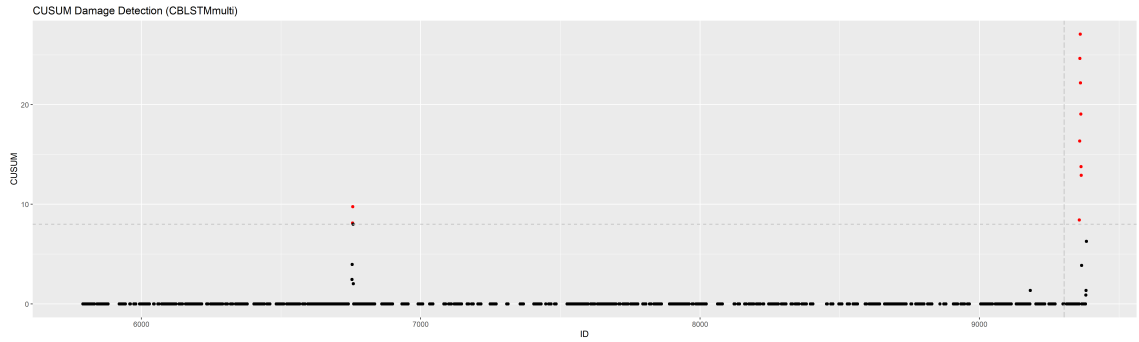


Figure 5.6: CUSUM Damage Detection for CBLSTM multi output algorithm, zoomed on undamaged period and early damage data.

Figure 5.5 plots damage detection of optimal CUSUM rule for CBSLTM multi output data. It is observed that the lowered sensitivity of the rule yields a slow response to damage inception. However, once damage is first detected, it is consistently signaled, rising to a continuously growing cumulative sum.

Zooming in on undamaged period and early damaged period, 5.6 indicates that all damaged observations identified are referent to the same spike in the cumulative sum, which is translated to one single damage identification in a real case scenario.

Comparing both models, PPOC presents excellent results on false positives, attaining 0 false positives in the most successful setup, while CUSUM's true positives, with over 90% in most cases, clearly outperform PPOC, with its low values, mostly under 60%. All in all, both models combined, fed by the information produced by CBLSTM multi output algorithm, demonstrate strong potential to correctly detect damage in the studied structure.

Chapter 6

Conclusion

This chapter presents a final analysis over all the procedures adopted and results outputted. Recalling the objective for this project, final considerations are made in order to establish the optimal combination of techniques to produce an accurate damage detection tool and its potential for application in real time to full size structures.

Regarding the dataset, despite its considerably large time window for information gathering and the overall extensive quantity of variables recorded, the initial presence of many outliers in undamaged phase and their removal conditioned the development of LSTM models quite severely. From the beginning of the project, LSTM models were expected to yield the more accurate predictions, due to its greater complexity and the ability to interpret contextual information. Said superiority, although noticed, revealed to be smaller than expected.

Regarding the remaining prediction algorithms, Random Forests and Support Vector Regression models, despite simpler, yielded a performance similar to the LSTM models. ANN models also presented a prediction error similar to LSTM models. Ensemble model successfully attained a lower generalization error than the corresponding single algorithms used for its construction, proving the usefulness of combining predictions from different models. At this point, it was impossible to rank the algorithms in what regards their potential for application on a real-life scenario, and consequently the residuals from all models were considered for building Control Charts.

Hotelling T^2 control charts did accurately distinguish the two states of the structure, i.e. undamaged and damaged, showing great potential for application on a real-life scenario.

Comparing the result attained from feeding Hotelling T^2 with the residuals from different predicting algorithms, the use RF and SVR models proved to be unsuccessful, resulting in a poor detection of abnormal situations. The combination of the remaining models - Ensemble, ANN and both LSTM - with Hotelling Charts resulted in a good capability of damage detection, being the combination LSTM-Hotelling the top performing, and Ensemble-Hotelling falling behind, likely influenced by SVR and RF models. However, all combinations proved to be satisfactory, as a low amount of false positives and false negatives was attained. Therefore, these four combinations were considered for the construction of timely damage detection rules.

Two rules were proposed, one based on percentage points out of control in a moving window (PPOC), are another based on the cumulative sum of control charts score (CUSUM). Overall, PPOC and CUSUM present themselves as complementary rules, providing a more secure and accurate damage detection system.

Regarding the application of said rules to the different combinations of Data Mining Algorithms/Hotelling Charts, the model based on Ensemble predictions struggled to accurately detect damage, as a consequence of a subpar anomaly detection. LSTM models yielded the best performance, as a confirmation of its superiority, already evidenced in previous stages.

The applied Recurrent Neural Networks, Long Short-Term Memory, single and multiple output, revealed to be the most successful algorithms in all stages of the process. Having the best prediction performance (CBLSTM single), best number of false positives (CBLSTM single) in abnormal situation detection, to overall best damage detection (CBLSTM multi), these algorithms proved to be promising tools to support Structure Health Monitoring, specially when applied to continuous data where different inputs can be studied.

Summing up, the results of this project highlight the potential of Convolutional Bi-directional Long Short-Term Memory algorithms for variable prediction, followed by anomaly signaling through Hotelling T^2 Control Charts, used to feed a Damage Detection system composed by the integration of both a Percentage Points out of Control on a Moving Window and a Cumulative Sum change detection rules.

The potential that the techniques studied in this project demonstrated for Structural Health Monitoring reiterates the goal of applying the damage detection tool on a full-size, real life structure, in order to make the better use of its utility.

Therefore, future works should aim for the application of the damage detecting tool developed on this project on a real life structure. Data should be retrieved from the structure for model training. The model should then predict the behavior of the structure in real time, for a live feedback on its behaviour. The use of Internet technologies for faster, widely available Data Mining process is recommended.

Bibliography

Aggarwal, C. C. (2018). *Neural Networks and Deep Learning*. Springer International Publishing, 1 edition.

Alipour, M., Harris, D. K., Barnes, L. E., Ozbulut, O. E., and Carroll, J. (2017). Load-capacity rating of bridge populations through machine learning: application of decision trees and random forests. *Journal of Bridge Engineering*, 22(10):04017076 (12 pp.).

Anaissi, A., Nguyen Lu Dang, K., Mustapha, S., Alamdari, M. M., Braytee, A., Yang, W., and Fang, C. (2017a). Adaptive one-class support vector machine for damage detection in structural health monitoring. In *Advances in Knowledge Discovery and Data Mining. 21st Pacific-Asia Conference, PAKDD 2017, 23-26 May 2017*, volume pt.I of *Advances in Knowledge Discovery and Data Mining. 21st Pacific-Asia Conference, PAKDD 2017. Proceedings: LNAI 10234*, pages 42–57. Springer International Publishing.

Anaissi, A., Nguyen Lu Dang, K., Rakotoarivelo, T., Alamdari, M. M., and Yang, W. (2017b). Self-advised incremental one-class support vector machines: An application in structural health monitoring. In *Neural Information Processing. 24th International Conference, ICONIP 2017, 14-18 Nov. 2017*, volume pt.I of *Neural Information Processing. 24th International Conference, ICONIP 2017. Proceedings: LNCS 10634*, pages 484–96. Springer International Publishing.

Awad, M. and Khanna, R. (2015). *Support Vector Regression*. Apress, Berkeley, CA.

Aygun, B. and Gungor, V. (2011). Wireless sensor networks for structure health monitoring: recent advances and future research directions. *Sensor Review*, 31(3):261 – 76.

Barthorpe, R. J. (2010). On model- and data-based approaches to structural health monitoring.

Biau, G. and Scornet, E. (2016). A random forest guided tour. *TEST*, 25(2):197–227.

Boukabache, H., Matmat, M., Escriba, C., and Fourniols, J. (2011). Sensors/actuators network development for aeronautics structure health monitoring. pages 4 pp. –, Piscataway, NJ, USA.

Brachman, R. J. and Anand, T. (1994). *The Process of Knowledge Discovery in Databases: A First Sketch*, volume WS-94-03 of *AAAI Workshop Series Technical Report*. Association for the Advancement of Artificial Intelligence.

Brachman, R. J. and Anand, T. (1996). *The Process of Knowledge Discovery in Databases: A Human-Centered Approach*, pages 37–58. AAAI Press, Menlo Park, California.

Breiman, L. (1996a). Bagging predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (1996b). Bias, variance, and arcing classifiers. Technical report.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Chapman Hall/CRC, Boca Raton.

Bühlmann, P. and Yu, B. (2002). Analyzing bagging. *Ann. Statist.*, 30(4):927–961.

Chang, P. C., Flatau, A., and Liu, S. C. (2003). Review paper: Health monitoring of civil infrastructure. *Structural Health Monitoring*, 2:257–267.

Cheng, Y., Hanif, A., and Li, Z. (2018). Development of a flexible capacitive sensor for concrete structure health monitoring. volume 10598, pages 105980N (9 pp.) –, USA.

Christmann, A. and Steinwart, I. (2008). *Support Vector Machines*. Springer, New York, NY, New York.

Cios, K. J., Swiniarski, R. W., Pedrycz, W., and Kurgan, L. A. (2007). *The Knowledge Discovery Process*, pages 9–24. Springer US, Boston, MA.

Cross, E. J., Worden, K., and Chen, Q. (2011). Cointegration: A novel approach for the removal of environmental trends in structural health monitoring data. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 467:2712–2732.

Da Silva, M. F. M. (2017). Machine learning algorithms for damage detection in structures under changing normal conditions.

Dragomirescu, D., Kraemer, M., Jatlaoui, M., Pons, P., Aubert, H., Thain, A., and Plana, R. (2010). 60ghz wireless nano-sensors network for structure health monitoring as enabler for safer, greener aircrafts. volume 7821, pages 782105 (10 pp.) –, USA.

Dunia, R. and Qin, S. J. (1998). Subspace approach to multidimensional fault identification and reconstruction. *En*, 44.

Efron, B. and Tibshirani, R. (1993). *An introduction to the bootstrap*. Chapman Hall, New York.

Fayyad, U. M., Piatetsky-Shapiro, G., and Smyth, P. (1996). *Advances in Knowledge Discovery and Data Mining*. MIT Press Ltd, Cambridge, Massachusetts.

Frawley, W. J., Piatetsky-Shapiro, G., and Matheus, C. J. (1992). Knowledge discovery in databases: An overview. *AI Magazine*, 13(3).

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139.

Ge, G. (2017). The application of the wireless sensor network on port wharf structure health monitoring. pages 701 – 7, Piscataway, NJ, USA.

Graves, A. (2012). *Long Short-Term Memory*, pages 37–45. Springer Berlin Heidelberg, Berlin, Heidelberg.

Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602 – 610. IJCNN 2005.

Grigg, O. A., Farewell, V. T., and Spiegelhalter, D. J. (2003). Use of risk-adjusted cusum and rsprtcharts for monitoring in medical contexts. *Statistical Methods in Medical Research*, 12(2):147–170. PMID: 12665208.

Hamasaki, T. (2013). Power management of autonomous wireless sensor node for structure health monitoring. *Analog Integrated Circuits and Signal Processing*, 75(2):217 – 24.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366.

Howard, J. and Bowles, M. (2012). The two most important algorithms in predictive modeling today. In *Strata Conference*.

Iliadis, L. and Maglogiannis, I. (2016). *Artificial Intelligence Applications and Innovations*. Springer, Cham.

Kesavan, A., John, S., and Herszberg, I. (2008). Strain-based structural health monitoring of complex composite structures. *Structural Health Monitoring*, 7:203–213.

Khazaeli, S., Ravandi, A. G., Banerji, S., and Bagchi, A. (2016). The application of data mining and cloud computing techniques in data-driven models for structural health monitoring. In *Health Monitoring of Structural and Biological Systems 2016, 21-24 March 2016*, volume 9805 of *Proc. SPIE (USA)*, page 98052M (18 pp.). SPIE.

Kim, J.-T., Ryu, Y.-S., Cho, H.-M., and Stubbs, N. (2003). Damage identification in beam-type structures: frequency-based method vs mode-shape-based method. *Engineering Structures*, 25(1):57–67.

Kleiner, A., Talwalkar, A., Sarkar, P., and Jordan, M. I. (2014). A scalable bootstrap for massive data. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(4):795–816.

Kromanis, R. and Kripakaran, P. (2013). Advanced engineering informatics support vector regression for anomaly detection from measurement histories. *Advanced Engineering Informatics*, 27:486–495.

Kumar, R. and Hossain, A. (2018). Experimental performance and study of low power strain gauge based wireless sensor node for structure health monitoring. *Wireless Personal Communications*, 101(3):1657 – 69.

Kurian, B. and Liyanapathirana, R. (2020). Machine learning techniques for structural health monitoring. In *13th International Conference on Damage Assessment of Structures, 9-10 July 2019*, Proceedings of the 13th International Conference on Damage Assessment of Structures. DAMAS 2019. Lecture Notes in Mechanical Engineering (LNME), pages 3–24. Springer.

Li, X., Yu, W., and Villegas, S. (2016). Structural health monitoring of building structures with online data mining methods. *IEEE Systems Journal*, 10(3):1291–1300.

Lopes, V., J., Park, G., Cudney, H. H., and Inman, D. J. (2000). Impedance-based structural health monitoring with artificial neural networks. *Journal of Intelligent Material Systems and Structures*, 11(3):206–14.

Manaswi, N. K. (2018). *Deep Learning with Applications Using Python*. Apress, Berkeley, CA, 1 edition.

Nunes Silva, I., Hernane Spatti, D., Andrade Flauzino, R., Helena Bartocci Liboni, L., and Franco dos Reis Alves, S. (2017). *Artificial Neural Networks*. Springer International Publishing, 1 edition.

Park, G. and Inman, D. J. (2007). Structural health monitoring using piezoelectric impedance measurements. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 365:373–392.

Phares, B., Lu, P., Wipf, T., Greimann, L., and Seo, J. (2013). Field validation of a statistical-based bridge damage-detection algorithm. *Journal of Bridge Engineering*, 18:1227–1238.

Posenato, D., Lanata, F., Inaudi, D., and Smith, I. F. (2008). Model-free data interpretation for continuous monitoring of complex structures. *Advanced Engineering Informatics*, 22:135–144.

Qifeng, Z., Yongpeng, N., Qingqing, Z., Linkai, L., and Jiayan, L. (2013). Structural damage detection method based on random forests and data fusion. *Structural Health Monitoring*, 12(1):48–58.

Reynders, E., Wursten, G., and de Roeck, G. (2014). Output-only structural health monitoring in changing environmental conditions by means of nonlinear system identification. *Structural Health Monitoring*, 13:82–93.

Rodrigues, M. (2020). *Structural Health Monitoring*. Thesis.

Roiger, R. J. (2017). *Data Mining: A Tutorial-Based Primer, Second Edition*. Chapman Hall/CRC Data Mining and Knowledge Discovery Series. Taylor Francis Group, LLC, Department of Computer Science and Engineering, Minneapolis, Minnesota, USA.

Satoru, S., Keizo, M., and Shinji, K. (2014). Structure health monitoring system using mems-applied vibration sensor. *Fuji Electric Review*, 60(1):54 – 8.

Schölkopf, B. and Smola, A. J. (2018). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press.

Slišković, D., Grbić, R., and Hocenski, (2012). Multivariate statistical process monitoring.

Sousa Tomé, E., Pimentel, M., and Figueiras, J. (2020). Damage detection under environmental and operational effects using cointegration analysis – application to experimental data from a cable-stayed bridge. *Mechanical Systems and Signal Processing*, 135:4–8.

Tan, P.-N., Steinbach, M., and Kumar, V. (2014). *Introduction to Data Mining*. Pearson Education Limited, Edinburgh Gate, Harlow, Essex CM20 2JE.

Tibaduiza, D. A., Mujica, L. E., and Rodellar, J. (2011). Structural health monitoring based on principal component analysis: damage detection, localization and classification. *Advances in Dynamics, Control, Monitoring and Applications, Universitat Politècnica de Catalunya, Departament de Matemàtica Aplicada*, 3:8–17.

Tomé, E. S., Pimentel, M., and Figueiras, J. (2019). Shm based damage detection using cointegration and linear multivariate data analysis: performance comparison based on a real case study.

Usama, F., Gregory, P.-S., and Padhraic, S. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, 17(3).

Vapnik, V. (1995). *The Nature of Statistical Learning*. Information Science and Statistics. Springer-Verlag New York, New York.

Wager, S., Hastie, T., and Efron, B. (2014). Confidence intervals for random forests: The jackknife and the infinitesimal jackknife. *Journal of Machine Learning Research*, 15(48):1625–1651.

Wipf, T. J., Phares, B., Doornink, J. D., Greimann, L., and Wood, D. L. (2007). Evaluation of steel bridges , volumes i ii.

Worden, K., Cross, E., and Barton, E. (2012). Damage detection on the npl footbridge under changing environmental conditions. *Proceedings of the 6th European Workshop - Structural Health Monitoring 2012, EWSHM 2012*, 2:1124–1131.

Yan, A. M., Kerschen, G., De Boe, P., and Golinval, J. C. (2005). Structural damage diagnosis under varying environmental conditions - part i: A linear analysis. *Mechanical Systems and Signal Processing*, 19:847–864.

Yi, T.-H., Li, H.-N., Song, G., and Zhang, X.-D. (2015). Optimal sensor placement for health monitoring of high-rise structure using adaptive monkey algorithm. *Structural Control and Health Monitoring*, 22(4):667 – 81.

Zi-yan, W., Feng-juan, D., and Hai-feng, Y. (2006). Research on optimal sensor placement methods for structure health monitoring. *Journal of Zhengzhou University Engineering Science*, 27(4):92 – 6.

Appendix A

Data Acquisition and Processing Code

A.1 Data Acquisition

```
if (!require("pacman")) install.packages("pacman")
pacman::p_load(rio,caret, randomForest, Metrics,taRifx,
                        ggplot2,tidyr,reshape2,e1071, tweenr,utils,nnet,
                        lubridate,tidyverse,imputeTS,zoo,
                        MASS,qcc,MSQC,evobiR)

data_ud <- rio::import('DadosUD.csv')
data_ud$Estado <- 'UD'
data_d <- rio::import('DadosD.csv')
data_d$Estado <- 'D'


names(data_ud)[1] <- 'ID'
data_ud$Time <- as.POSIXct(data_ud$Time,tz="GMT")

names(data_d)[1] <- 'ID'
data_d$Time <- as.POSIXct(data_d$Time,tz="GMT")
data_d$ID <- data_d$ID+9303

data <- rbind(data_ud,data_d)

var_ind <- c(3,4,5,15,16,17,18)
var_dep <- c(6,7,8,9,10,11,12,13,14)
```

```
feats <- colnames(data)

data_train_var <- data_ud[1:3747,]
data_val_var <- data_ud[3748:5790,]
data_test_var <- data_ud[5791:7184,]
data_pred_var <- data_ud[7185:9303,]
data_model_var <- data_ud[1:5790,]

set.seed(13)
fold_train <- sample(10, nrow(data_train_var), replace=TRUE)
fold_val <- sample(10, nrow(data_val_var),replace=TRUE)

random_model <- sample(1:nrow(data_model_var))
```

A.2 Data Processing for LSTM Models

```
CNN_split_ind <- function(x, n_steps, n_seq){
  split_x <- array(dim= c((nrow(x)-(n_steps*n_seq)+1),n_seq,n_steps,(ncol(x))))
  for (row in 1:nrow(split_x)){
    for (feature in 1:(ncol(x))){
      for(seq in 1:n_seq){
        split_x[row,seq,,feature] <- x[(row+((seq-1)*n_steps)):
                        (row+(n_steps-1)+((seq-1)*n_steps)),feature]
      }
    }
  }
  return(split_x)
}



lstm_data <- data
second(lstm_data$Time) <- 0

minute(lstm_data$Time[minute(lstm_data$Time)>=0 & minute(lstm_data$Time)<15]) <- 0
minute(lstm_data$Time[minute(lstm_data$Time)>=15 & minute(lstm_data$Time)<30]) <- 15
minute(lstm_data$Time[minute(lstm_data$Time)>=30 & minute(lstm_data$Time)<45]) <- 30
minute(lstm_data$Time[minute(lstm_data$Time)>=45 & minute(lstm_data$Time)<60]) <- 45

lstm_data <- lstm_data[!duplicated(lstm_data$Time),]

missing <- data.frame(as.numeric(difftime(lstm_data$Time,lag(lstm_data$Time))))
missing[1,2] <- 0
missing[which(missing[,1]<=15),1] <- 0
missing[which(missing[,1]>15),1] <- 1

missing[1,3] <- 1
for(i in 2:nrow(missing)){
  if(missing[i,1]==1){
    missing[i,2] <- 0
  }else{
    missing[i,2] <- lag(missing[,2])[i]+1
  }
}
```

```
preblanks <- which(lead(missing[,3])==0)



cap <- n_steps*n_seq
        preblanks2 <- preblanks[which(missing[preblanks,3]>=cap)]

        groups <- data.frame(preblanks2,missing[preblanks2,3])
        #separar undamaged de damaged
        if(cap<=14){
        groups <- rbind(groups[groups[,1]<9303,],c(9303,14),groups[groups[,1]>9303,])
        }
        groups[which(groups[,1]==10598),2] <- 1295
        #groups_train <- groups[groups[,1]<=3747,]
        #groups_val <- groups[groups[,1]>=3748 & groups[,1]<=5790,]
        #groups_test <- groups[groups[,1]>=5791 & groups[,1]<=7184,]
        #groups_pred <- groups[groups[,1]>=7185 & groups[,1]<=9303,]
        #groups_model <- groups[groups[,1]<=5790,]

        n_features <- length(var_ind)




        groups_data_X <- array(dim=c(1,n_seq,n_steps,n_features))
        groups_data_vars_Y <- data.frame(lstm_data[1,var_dep])
        groups_ID_Time <- data.frame(lstm_data[1,1:2])

        for(i in 1:nrow(groups)){
          aux <- lstm_data[(groups[i,1]-groups[i,2]+1):groups[i,1],var_ind]
          old_data <- groups_data_X
          new_data <- CNN_split_ind(aux,n_steps,n_seq)
          groups_data_X <- array(dim=c((dim(old_data)[1]+dim(new_data)[1]),
                    n_seq,n_steps,n_features))
          groups_data_X[1:dim(old_data)[1],,,] <- old_data
          groups_data_X[(dim(old_data)[1]+1):dim(groups_data_X)[1],,,] <- new_data
          groups_data_vars_Y <- rbind(groups_data_vars_Y,
                    lstm_data[(groups[i,1]-groups[i,2]+(n_steps*n_seq)):groups[i,1],var_dep])
          groups_ID_Time <- rbind(groups_ID_Time,
```

```
        lstm_data[(groups[i,1]-groups[i,2]+(n_steps*n_seq)):groups[i,1],1:2])
}

old_data <- groups_data_X
groups_data_X <- array(dim=c((dim(old_data)[1]-1),n_seq,n_steps,n_features))
groups_data_X[,,,] <- old_data[-c(1),,,]
groups_data_vars_Y <- groups_data_vars_Y[-c(1),]
groups_ID_Time <- groups_ID_Time[-c(1),]

train <- length(which(groups_ID_Time$ID<=3747))
val <- length(which(groups_ID_Time<=5790))
test <- length(which(groups_ID_Time<=7184))
pred <- length(which(groups_ID_Time<=9303))


groups_data_train_X <- array(dim=c((train),n_seq,n_steps,n_features))
groups_data_train_X[,,,] <- groups_data_X[1:train,,,]
groups_data_train_vars_Y <- groups_data_vars_Y[1:train,]

groups_data_val_X <- array(dim=c((val-train),n_seq,n_steps,n_features))
groups_data_val_X[,,,] <- groups_data_X[(train+1):val,,,]
groups_data_val_vars_Y <- groups_data_vars_Y[(train+1):val,]

groups_data_test_X <- array(dim=c((-val+test),n_seq,n_steps,n_features))
groups_data_test_X[,,,] <- groups_data_X[(val+1):test,,,]
groups_data_test_vars_Y <- groups_data_vars_Y[(val+1):test,]

groups_data_pred_X <- array(dim=c((-test+pred),n_seq,n_steps,n_features))
groups_data_pred_X[,,,] <- groups_data_X[(test+1):pred,,,]
groups_data_pred_vars_Y <- groups_data_vars_Y[(test+1):pred,]

groups_data_model_X <- array(dim=c((+val),n_seq,n_steps,n_features))
groups_data_model_X[,,,] <- groups_data_X[1:val,,,]
groups_data_model_vars_Y <- groups_data_vars_Y[1:val,]




groups_data_train_Y <- array(data=groups_data_train_vars_Y[,var],
```

```
        dim=c(nrow(groups_data_train_vars_Y),1))
groups_data_val_Y <- array(data=groups_data_val_vars_Y[,var],
        dim=c(nrow(groups_data_val_vars_Y),1))
groups_data_test_Y <- array(data=groups_data_test_vars_Y[,var],
        dim=c(nrow(groups_data_test_vars_Y),1))
groups_data_pred_Y <- array(data=groups_data_pred_vars_Y[,var],
        dim=c(nrow(groups_data_pred_vars_Y),1))
groups_data_model_Y <- array(data=groups_data_model_vars_Y[,var],
        dim=c(nrow(groups_data_model_vars_Y),1))
groups_data_Y <- array(data=groups_data_vars_Y[,var],
        dim=c(nrow(groups_data_vars_Y),1))
```

Appendix B

Algorithms Code

B.1 Random Forests

```
rf_performance <- data.frame(matrix(nrow=4,ncol=9))
colnames(rf_performance) <- feats[var_dep]
rownames(rf_performance) <- c("MAPE_val","MSE_val","MAPE_test","MSE_test")

rf_predictions <- data.frame(data$ID,data$Time)



for (var in 1:length(var_dep)){

  vars<- c(var_ind,var_dep[var])

  data_train <- data_train_var[,vars]
  data_val <- data_val_var[,vars]
  data_test <- data_test_var[,vars]
  data_pred <- data_pred_var[,vars]
  data_model <- data_model_var[,vars]

  formula <- as.formula(paste(feats[var_dep[var]],"~.",sep=""))

  rf_tuning <- data.frame()
  for (i in c(1:7)){
    predictions <- data.frame()
    true_value <- data.frame()
    for (j in c(1:10)){
      fit <- randomForest(formula,data=data_train[fold_train==j,],
```

```
                                                    mtry=i,ntree=100)
      predict <- predict(fit, data_val[fold_val==j,])
      predictions <- rbind(predictions, data.frame(predict))
      true_value <- c(true_value, data_val[fold_val==j,8])
    }
    n_var <- i
    MAPE <- Metrics::mape(unlist(true_value),predictions$predict)
    MSE <- Metrics::mse(unlist(true_value),predictions$predict)
    rf_tuning <- rbind(rf_tuning, data.frame(n_var,MAPE,MSE))
  }

  n_var <- rf_tuning[which.min(rf_tuning$MAPE),'n_var']
  rf_performance[1,var] <- rf_tuning$MAPE[which.min(rf_tuning$MAPE)]
  rf_performance[2,var] <- rf_tuning$MSE[which.min(rf_tuning$MAPE)]

  true_value <- data.frame()
  predictions <- data.frame()

  rf_model <- randomForest::randomForest(formula,data=data_model[random_model,],
                                                                mtry=i,ntree=100)
  predictions <- predict(rf_model,data)
  predictions <- data.frame(predictions)
  rf_predictions <- cbind(rf_predictions,predictions$predict)

  rf_performance[3,var] <- Metrics::mape(data_test[,8],predictions$predict[5791:7184])
  rf_performance[4,var] <- Metrics::mse(data_test[,8],predictions$predict[5791:7184])
}
```

B.2 Support Vector Machines

```
svr_performance <- data.frame(matrix(nrow=4,ncol=9))
colnames(svr_performance) <- feats[var_dep]
rownames(svr_performance) <- c("MAPE_val","MSE_val","MAPE_test","MSE_test")

svr_predictions <- data.frame(data$ID,data$Time)

for (var in 1:length(var_dep)){

  vars<- c(var_ind,var_dep[var])

  data_train <- data_train_var[,vars]
  data_val <- data_val_var[,vars]
  data_test <- data_test_var[,vars]
  data_pred <- data_pred_var[,vars]
  data_model <- data_model_var[,vars]

  formula <- as.formula(paste(feats[var_dep[var]],"~.",sep=""))

  svr_tuning <- data.frame()
  svr_tuning_predictions <- data.frame()

  for (cost in 2^(seq(2,9,by=2))){
    for (gamma in seq(0.1,1,0.1)){
      svr_tuning_predictions <- data.frame()
      for (k in 1:10){
      model <- e1071::svm(formula,data_train[fold_train==k,],scale=T,
                    cost=cost, gamma=gamma, kernel="linear", type="eps-regression")
      predictions <- predict(model,data_val[fold_val==k,])
      predictions <- data.frame(predictions)
      svr_tuning_predictions <- rbind(svr_tuning_predictions,
                data.frame(true=data_val[fold_val==k,8],pred=predictions$predict))
      }
    MSE <- Metrics::mse(svr_tuning_predictions$true,svr_tuning_predictions$pred)
    MAPE <- Metrics::mape(svr_tuning_predictions$true,svr_tuning_predictions$pred)

    svr_tuning <- rbind(svr_tuning,data.frame(cost,gamma,MSE,MAPE))
    }
```

```
  }

  svr_model <- e1071::svm(formula,data_model[random_model,],
                    cost=cost, gamma=gamma,scale=T)
  predictions <- predict(svr_model,data)
  predictions <- data.frame(predictions)

  svr_predictions <- cbind(svr_predictions,predictions$predict)

  svr_performance[3,var] <- Metrics::mape(data_test[,8],predictions$predict[5791:7184])
  svr_performance[4,var] <- Metrics::mse(data_test[,8],predictions$predict[5791:7184])

}
```

B.3 Artificial Neural Networks

```
ann_performance <- data.frame(matrix(nrow=4,ncol=9))
colnames(ann_performance) <- feats[var_dep]
rownames(ann_performance) <- c("MAPE_val","MSE_val","MAPE_test","MSE_test")

ann_predictions <- data.frame(data$ID,data$Time)

for (var in 1:length(var_dep)){

  vars<- c(var_ind,var_dep[var])

  data_train <- data_train_var[,vars]
  data_val <- data_val_var[,vars]
  data_test <- data_test_var[,vars]
  data_pred <- data_pred_var[,vars]
  data_model <- data_model_var[,vars]

  f <- paste(colnames(data)[var_dep[var]],"~",paste(colnames(data_ud[,var_ind]),
        collapse=" + "),sep="")

  ann_tuning <- data.frame()

  for (size in (seq(40,110,10))){
    for (decay in 10^seq(-5, 0, length = 10)){
      ann_tuning_predictions <- data.frame()
      for (k in 1:10){

        model <- nnet::nnet(formula=f.,x=data_train[fold_train==k,-8],
                        y=data_train[fold_train==k,8],size=size,decay=decay,
                        linout=T,trace = FALSE, maxit=500, MaxNWts=7000)
        predictions <- predict(model,data_val[fold_val==k,])
        predictions <- data.frame(predictions)
        ann_tuning_predictions <- rbind(ann_tuning_predictions,
                        data.frame(true=data_val[fold_val==k,8],pred=predictions$prediction))
      }
      MSE <- Metrics::mse(ann_tuning_predictions$true,ann_tuning_predictions$pred)
      MAPE <- Metrics::mape(ann_tuning_predictions$true,ann_tuning_predictions$pred)
```

```
      ann_tuning <- rbind(ann_tuning,data.frame(size,decay,MSE,MAPE))
    }
  }

  size <- ann_tuning$size[which.min(ann_tuning$MAPE)]
  decay <- ann_tuning$decay[which.min(ann_tuning$MAPE)]
  ann_performance[1,var] <- ann_tuning$MAPE[which.min(ann_tuning$MAPE)]
  ann_performance[2,var] <- ann_tuning$MSE[which.min(ann_tuning$MAPE)]

  ann_model <- nnet::nnet(formula=f,x=data_model[random_model,-8],
            y=data_model[random_model,8], size=size,decay=decay,
            linout=T,trace = FALSE, maxit=100, MaxNWts=7000)
  predictions <- predict(ann_model,data[,vars])
  predictions <- data.frame(predictions)
  ann_predictions <- cbind(ann_predictions,predictions$predict)
  ann_performance[3,var] <- Metrics::mape(data_test[,8],predictions$predict[5791:7184])
  ann_performance[4,var] <- Metrics::mse(data_test[,8],predictions$predict[5791:7184])

}
```

B.4 Ensemble Models

```
#Ensemble Average
real_values <- data[,c(1,2,var_dep)]
averaged_predictions <- data.frame(data$ID,data$Time)
for(var in 3:11){

  aux <- data.frame(rf_predictions[,var],ann_predictions[,var],svr_predictions[,var])
  average <- data.frame(apply(aux,1,mean))
  averaged_predictions <- data.frame(averaged_predictions,average)
}

averaged_performance <- data.frame(row.names=c('MSE','MAPE'))
for( var in 3:11){

  MSE <- mse(real_values[5791:7184,var],averaged_predictions[5791:7184,var])
  MAPE <- mape(real_values[5791:7184,var],averaged_predictions[5791:7184,var])
  averaged_performance <- data.frame(averaged_performance, c(MSE,MAPE))
}

#Ensemble Weighted Average
real_values <- data[,c(1,2,var_dep)]
weighted_predictions <- data.frame(data$ID,data$Time)
weights <- 1/c(mean(as.numeric(rf_performance[3,-c(4,5,7)])),
                              mean(as.numeric(ann_performance[3,-c(4,5,7)])),
                              mean(as.numeric(svr_performance[3,-c(4,5,7)])))

for(var in 3:11){

  aux <- data.frame(rf_predictions[,var]*weights[1]/sum(weights),
                                    ann_predictions[,var]*weights[2]/sum(weights),
                                    svr_predictions[,var]*weights[3]/sum(weights))
  average <- data.frame(apply(aux,1,sum))
  weighted_predictions <- data.frame(weighted_predictions,average)
}

weighted_performance <- data.frame(row.names=c('MSE','MAPE'))
for( var in 3:11){
```

```
  MSE <- mse(real_values[5791:7184,var],weighted_predictions[5791:7184,var])
  MAPE <- mape(real_values[5791:7184,var],weighted_predictions[5791:7184,var])
  weighted_performance <- data.frame(weighted_performance, c(MSE,MAPE))
}
```

B.5 LSTM Single Output

```
library(keras)
install_keras()

ks <- 1+floor((n_steps-1)/4)

model <- keras_model_sequential() %>%
  time_distributed(layer_conv_1d(filters=15, kernel_size=ks,activation='relu'),
                                 input_shape=c(n_seq,n_steps,n_features))%>%
  time_distributed(layer_max_pooling_1d(pool_size=1))%>%
  time_distributed(layer_conv_1d(filters=40, kernel_size=ks,activation='relu'))%>%
  time_distributed(layer_max_pooling_1d(pool_size=1))%>%
  time_distributed(layer_conv_1d(filters=40, kernel_size=ks,activation='relu'))%>%
  time_distributed(layer_max_pooling_1d(pool_size=1))%>%
  time_distributed(layer_conv_1d(filters=40, kernel_size=ks,activation='relu'))%>%
  time_distributed(layer_max_pooling_1d(pool_size=1))%>%
  time_distributed(layer_flatten())%>%
  bidirectional(layer_lstm(units=100, activation='relu',return_sequences=TRUE))%>%
  bidirectional(layer_lstm(units=50, activation='relu',return_sequences=TRUE))%>%
  bidirectional(layer_lstm(units=25, activation='relu'))%>%
  layer_dense(units=1)%>%
  compile(loss='mse',optimizer='adam',metrics=list("mean_absolute_percentage_error"))


model %>% summary()

CBLSTM7_performance  <- data.frame(matrix(nrow=2,ncol=9))
CBLSTM7_predictions <- data.frame(groups_ID_Time)
for (var in 1:length(var_dep)){

  #for(var in 1:length(var_dep)){
  groups_data_train_Y <- array(data=groups_data_train_vars_Y[,var],
                               dim=c(nrow(groups_data_train_vars_Y),1))
  groups_data_val_Y <- array(data=groups_data_val_vars_Y[,var],
                               dim=c(nrow(groups_data_val_vars_Y),1))
  groups_data_test_Y <- array(data=groups_data_test_vars_Y[,var],
                               dim=c(nrow(groups_data_test_vars_Y),1))
  groups_data_pred_Y <- array(data=groups_data_pred_vars_Y[,var],
```

```
                                dim=c(nrow(groups_data_pred_vars_Y),1))
  groups_data_model_Y <- array(data=groups_data_model_vars_Y[,var],
                                dim=c(nrow(groups_data_model_vars_Y),1))
  groups_data_Y <- array(data=groups_data_vars_Y[,var],
                                dim=c(nrow(groups_data_vars_Y),1))

  model %>% fit(groups_data_model_X,groups_data_model_Y, epochs=75)
  Y <-  model %>% predict(groups_data_test_X)
  CBLSTM7_performance[1,var] <- Metrics::mape(as.vector(groups_data_test_Y),as.vector(Y))
  CBLSTM7_performance[2,var] <- Metrics::mse(as.vector(groups_data_test_Y),as.vector(Y))

  Y <-  model %>% predict(groups_data_X)
  CBLSTM7_predictions <- cbind(CBLSTM7_predictions,Y)
}
```

B.6 LSTM Multi Output

```
ks <- 1+floor((n_steps-1)/4)
model <- keras_model_sequential() %>%
  time_distributed(layer_conv_1d(filters=15, kernel_size=ks,activation='relu'),
                         input_shape=c(n_seq,n_steps,n_features))%>%
  time_distributed(layer_max_pooling_1d(pool_size=1))%>%
  time_distributed(layer_conv_1d(filters=40, kernel_size=ks,activation='relu'))%>%
  time_distributed(layer_max_pooling_1d(pool_size=1))%>%
  time_distributed(layer_conv_1d(filters=40, kernel_size=ks,activation='relu'))%>%
  time_distributed(layer_max_pooling_1d(pool_size=1))%>%
  time_distributed(layer_conv_1d(filters=40, kernel_size=ks,activation='relu'))%>%
  time_distributed(layer_max_pooling_1d(pool_size=1))%>%
  time_distributed(layer_flatten())%>%
  bidirectional(layer_lstm(units=100, activation='relu',return_sequences=TRUE))%>%
  bidirectional(layer_lstm(units=50, activation='relu',return_sequences=TRUE))%>%
  bidirectional(layer_lstm(units=25, activation='relu'))%>%
  layer_dense(units=6)%>%
  compile(loss='mse',optimizer='adam',metrics=list("mean_absolute_percentage_error"))

model %>% summary()


groups_data_model_Y<- array(dim = c(nrow(groups_data_model_vars_Y),6,1))
groups_data_test_Y <- array(dim = c(nrow(groups_data_test_vars_Y),6,1))
groups_data_Y<- array(dim = c(nrow(groups_data_vars_Y),6,1))

groups_data_model_vars_Y <- groups_data_model_vars_Y[,c(1,2,3,6,8,9)]
groups_data_test_vars_Y <- groups_data_test_vars_Y[,c(1,2,3,6,8,9)]
groups_data_vars_Y <- groups_data_vars_Y[,c(1,2,3,6,8,9)]

for(var in 1:6){
groups_data_test_Y[,var,1] <- groups_data_test_vars_Y[,var]
groups_data_model_Y[,var,1] <- groups_data_model_vars_Y[,var]
groups_data_Y[,var,1] <- groups_data_vars_Y[,var]
}
model %>% fit(groups_data_model_X,groups_data_model_Y, epochs=150)

Y <-  model %>% predict(groups_data_test_X)
```

```
Y <- data.frame(Y)
for (var in 1:6){
  CBLSTM6_performance[1,var] <- Metrics::mape(groups_data_test_vars_Y[,var],Y[,var])
  CBLSTM6_performance[2,var] <- Metrics::mse(groups_data_test_vars_Y[,var],Y[,var])
}

Y <-  model %>% predict(groups_data_X)
CBLSTM6_predictions <- data.frame(Y)
```

Appendix C

Control Charts Code

The present annex reveals the code used for control chart elaboration. The present example refers to ANN model; code used for other models presents only slight adjustments, specially for LSTM models.

```
real_values <- data[,c(1,2,var_dep)]
ann_diff <- real_values[,-c(1,2,6,7,9)]-ann_predictions[,-c(1,2,6,7,9)]

Validation_Period <- ann_diff[1:5790,]
Undamaged_Test <- ann_diff[5791:9303,]
Undamaged_Period <- ann_diff[1:9303,]
Damaged_Period <- ann_diff[9304:(nrow(ann_diff)),]



rob <- cov.rob(Validation_Period)

ann_cc0 <- mqcc(Validation_Period, type = 'T2.single',
                confidence.level = 0.9999999999999999, plot=TRUE, center = rob$center,
                cov = rob$cov)
ann_cc1 <- mqcc(Validation_Period, type = 'T2.single',
                confidence.level = 0.9999999999999999,newdata=Undamaged_Test,
                plot=TRUE, center = rob$center,
                cov = rob$cov)
ann_cc2 <- mqcc(Undamaged_Period, type = 'T2.single',
                confidence.level = 0.9999999999999999,newdata = Damaged_Period,
                plot=TRUE, center = rob$center,
                cov = rob$cov)

ann_cc00 <- ewma(Validation_Period,lambda=0.2, nsigmas=8, plot=T)
ann_cc11 <- ewma(Validation_Period,lambda=0.2,nsigmas=8,newdata=Undamaged_Test, plot=T)
```

```
ann_cc22 <- ewma(Validation_Period,lambda=0.2,nsigmas=8,newdata = Damaged_Period, plot=T)


ann_hot_val <- mqcc(Validation_Period, type = 'T2.single', confidence.level =
                0.9999999999999999, plot=TRUE, center = rob$center, cov = rob$cov)
ann_hot_ud <- mqcc(Undamaged_Test, type = 'T2.single', confidence.level =
                0.9999999999999999, plot=TRUE, center = rob$center, cov = rob$cov)
ann_hot_d <- mqcc(Damaged_Period, type = 'T2.single', confidence.level =
                0.9999999999999999, plot=TRUE, center = rob$center, cov = rob$cov)
ann_stats <- data.frame(row.names=c('val','undamaged','damaged'))
ann_stats$UCL <- ann_hot_val$limits[1,2]
ann_stats <- cbind(ann_stats,groups=c(length(ann_hot_val$statistics),
                length(ann_hot_ud$statistics),
                length(ann_hot_d$statistics)))
ann_stats <- cbind(ann_stats,beyond=c(length(which(ann_hot_val$statistics>ann_stats[1,1])),
                length(which(ann_hot_ud$statistics>ann_stats[2,1])),
                length(which(ann_hot_d$statistics>ann_stats[3,1]))))
ann_stats$percentage <- ann_stats$beyond/ann_stats$groups
```

Appendix D

Damage Detection Rules Code

D.1 Percentage Points Out of Control on a Moving Window

```
ws <-12
cap <- 8

#CBLSTMsingle
CBLSTMsingle_damage_detection <- data.frame(groups_ID_Time[(val+1):pred,])
CBLSTMsingle_damage_detection$OU <- CBLSTMsingle_hot_ud$statistics
CBLSTMsingle_damage_detection$OU[CBLSTMsingle_damage_detection$OU<=
                                   CBLSTMsingle_hot_ud$limits[1,2]] <- 0
CBLSTMsingle_damage_detection$OU[CBLSTMsingle_damage_detection$OU>
                                   CBLSTMsingle_hot_ud$limits[1,2]] <- 1
CBLSTMsingle_damage_detection$Category <- 'Undamaged Test'
CBLSTMsingle_damage_detection$sum <-
  c(rep(NA,ws-1),rollapply(CBLSTMsingle_damage_detection$OU,ws,sum))

aux <- data.frame(groups_ID_Time[(pred+1):nrow(CBLSTMsingle_diff),])
aux$OU <- CBLSTMsingle_hot_d$statistics
aux$OU[aux$OU<=CBLSTMsingle_hot_ud$limits[1,2]] <- 0
aux$OU[aux$OU>CBLSTMsingle_hot_ud$limits[1,2]] <- 1
aux$Category <- 'Damaged'
aux$sum <- c(rep(NA,ws-1),rollapply(aux$OU,ws,sum))


CBLSTMsingle_damage_detection <- rbind(CBLSTMsingle_damage_detection,aux)

CBLSTMsingle_damage_detection$DD[CBLSTMsingle_damage_detection$sum<cap] <- 0
```

```
CBLSTMsingle_damage_detection$DD[CBLSTMsingle_damage_detection$sum>=cap] <- 1
```

D.2 Cumulative Sum Change Detection

```
#CBLSTMsingle
aux <- CBLSTMsingle_hot_val$statistics
ucl <- CBLSTMsingle_hot_val$limits[1,2]
aux[which(aux<ucl)] <- 0
aux[which(aux>=ucl)] <- 1
CBLSTMsingle_damage_detection$X <- c(as.vector(CBLSTMsingle_hot_ud$statistics),
                                     as.vector(CBLSTMsingle_hot_d$statistics))
m <- mean(CBLSTMsingle_hot_val$statistics)
sd <- sd(CBLSTMsingle_hot_val$statistics)
#m <- mean(aux)
#sd <- sd(aux)
#w <- (1-m)/sd*0.75
w <- ((ucl-m)/sd)*0.75
CBLSTMsingle_damage_detection$Z <- (CBLSTMsingle_damage_detection$X-m)/sd
CBLSTMsingle_damage_detection$Sh <- 0

for(i in 2:nrow(CBLSTMsingle_damage_detection)){

  CBLSTMsingle_damage_detection$Sh[i] <- max(0,CBLSTMsingle_damage_detection$Sh[i-1]+
                                               CBLSTMsingle_damage_detection$Z[i]-w)

}
CBLSTMsingle_damage_detection$CUSUM <- 0
CBLSTMsingle_damage_detection$CUSUM[which(CBLSTMsingle_damage_detection$Sh>=8)] <- 1
#plot(CBLSTMsingle_damage_detection£CUSUM)n£Sh>=8)] <- 1


CBLSTMsingle_damage_detection$ID[which(CBLSTMsingle_damage_detection$Sh>=8)]
sum(CBLSTMsingle_damage_detection$CUSUM[1:2028])/(2028)
sum(CBLSTMsingle_damage_detection$CUSUM[2029:nrow(CBLSTMsingle_damage_detection)])/
  (nrow(CBLSTMsingle_damage_detection)-2028)
```

Appendix E

Predictions and Real Values Plots

E.1 Random Forests Models

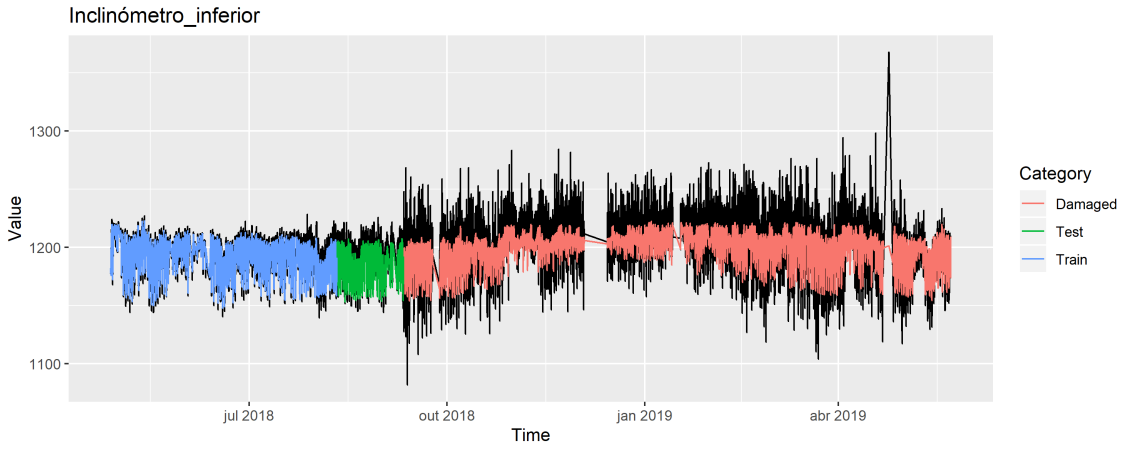


Figure E.1: Random Forests predictions for Lower Inclinometer

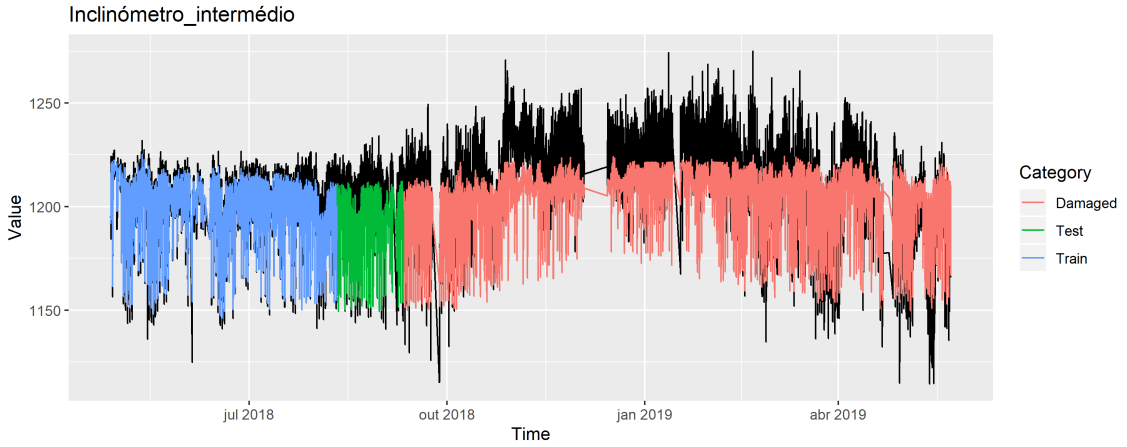


Figure E.2: Random Forests predictions for Middle Inclinometer

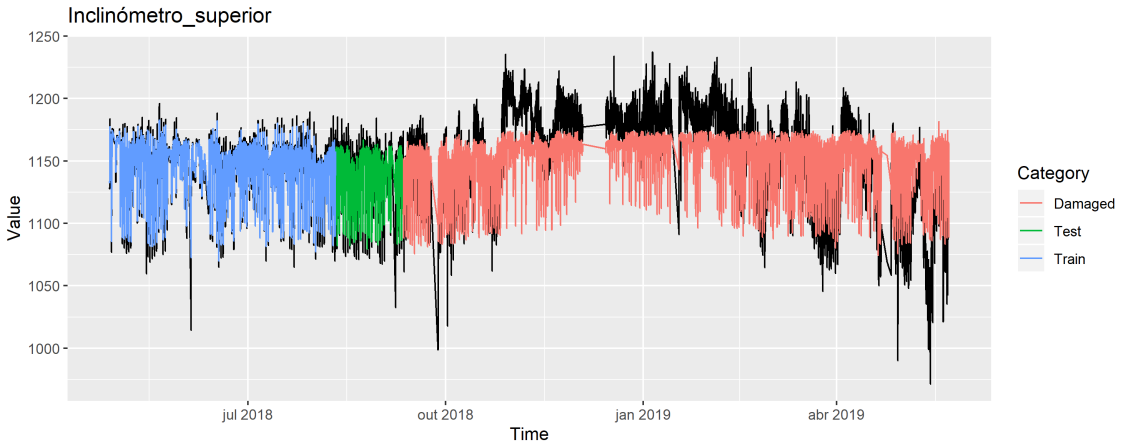


Figure E.3: Random Forests predictions for Upper Inclinometer

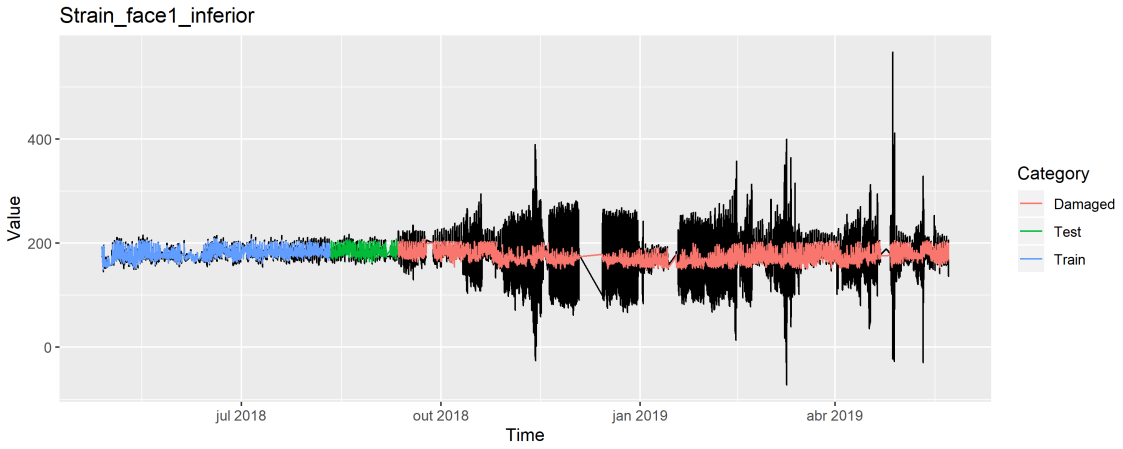


Figure E.4: Random Forests predictions for Lower F1 Strain

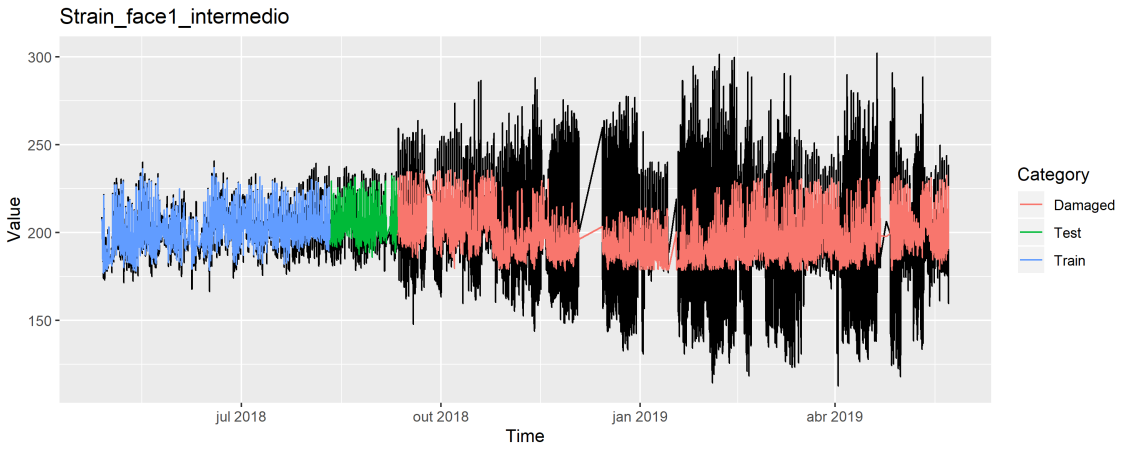


Figure E.5: Random Forests predictions for Middle F1 Strain

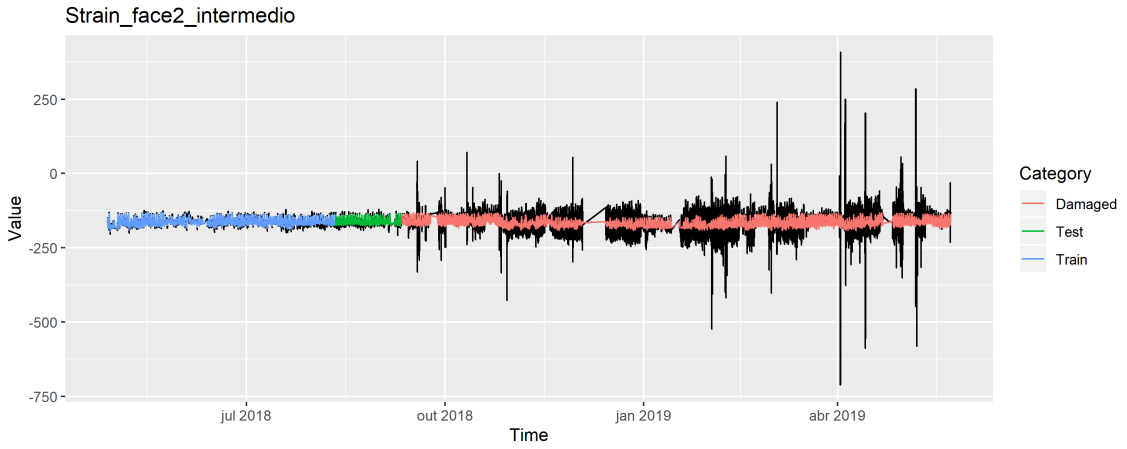


Figure E.6: Random Forests predictions for Middle F2 Strain

E.2 Support Vector Regression

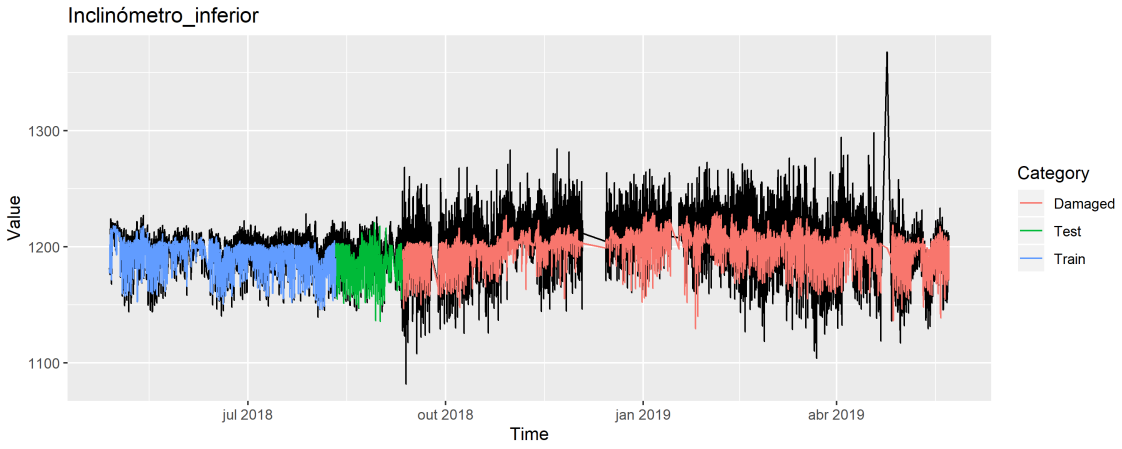


Figure E.7: Support Vector Regression predictions for Lower Inclinometer

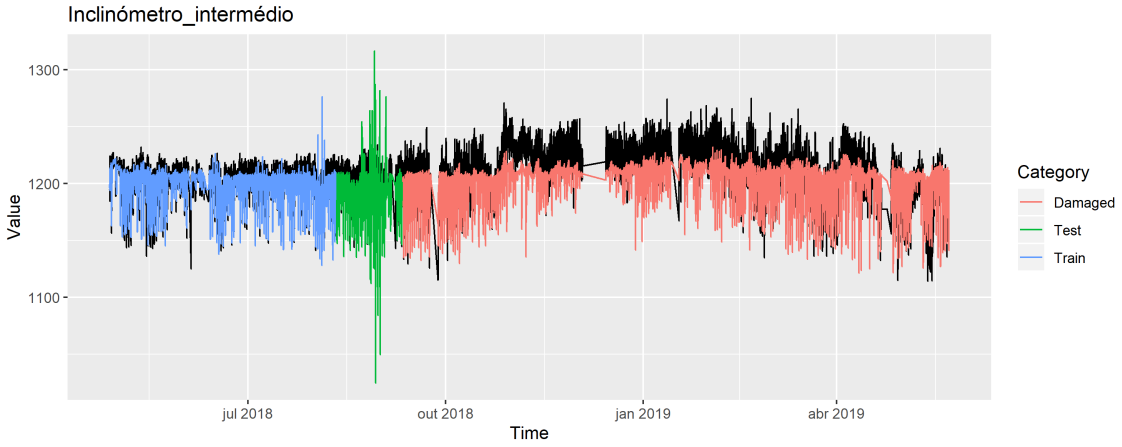


Figure E.8: Support Vector Regression predictions for Middle Inclinometer

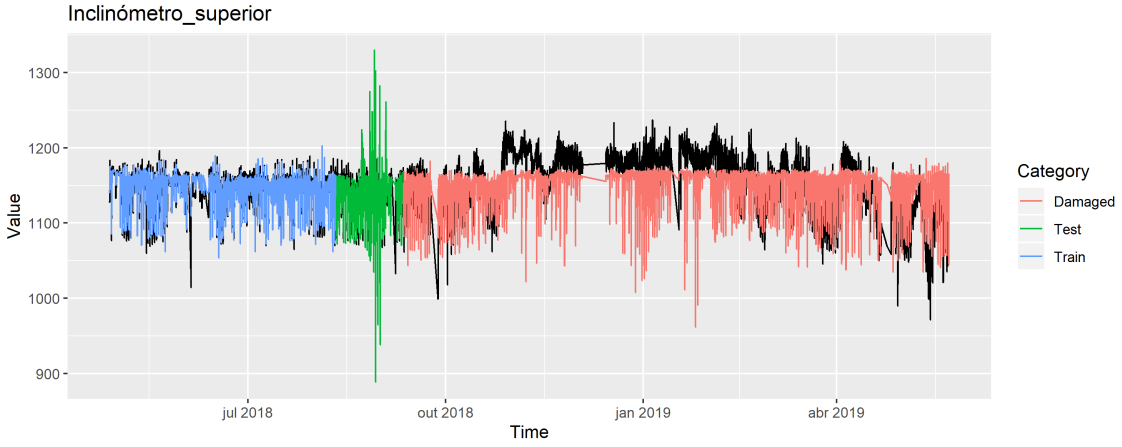


Figure E.9: Support Vector Regression predictions for Upper Inclinometer

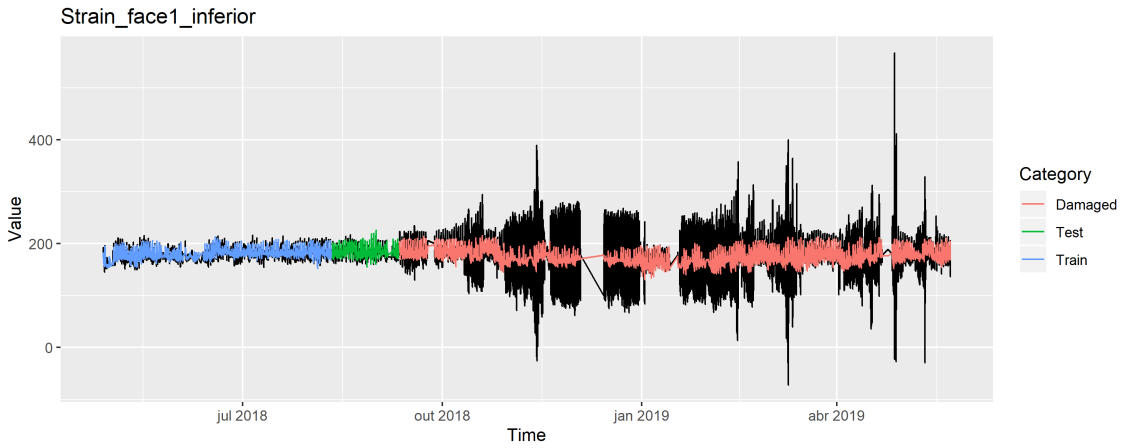


Figure E.10: Support Vector Regression predictions for Lower F1 Strain

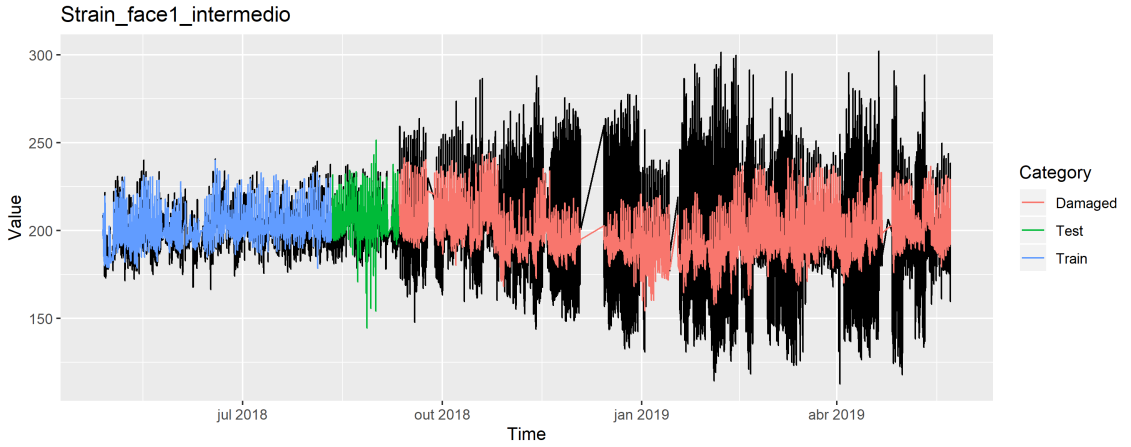


Figure E.11: Support Vector Regression predictions for Middle F1 Strain

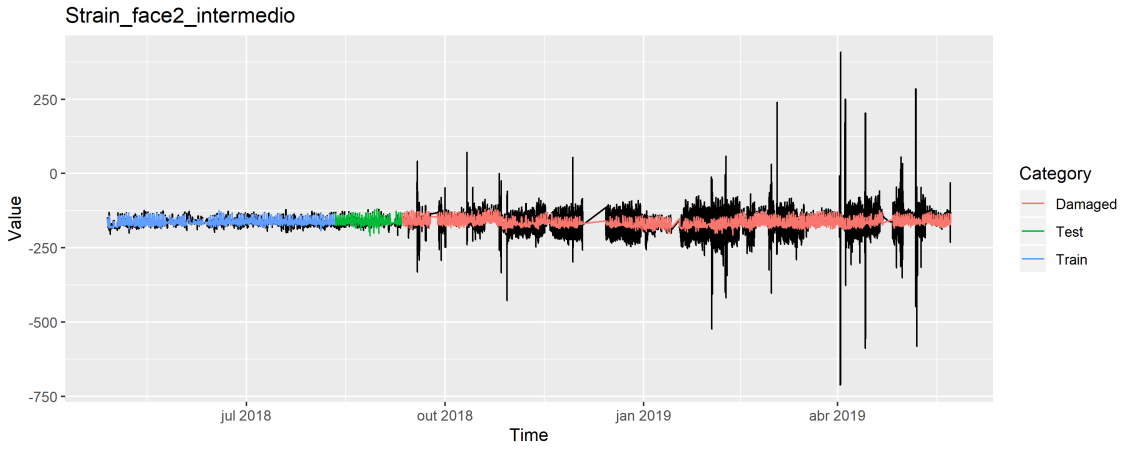


Figure E.12: Support Vector Regression predictions for Middle F2 Strain

E.3 Artificial Neural Networks Models

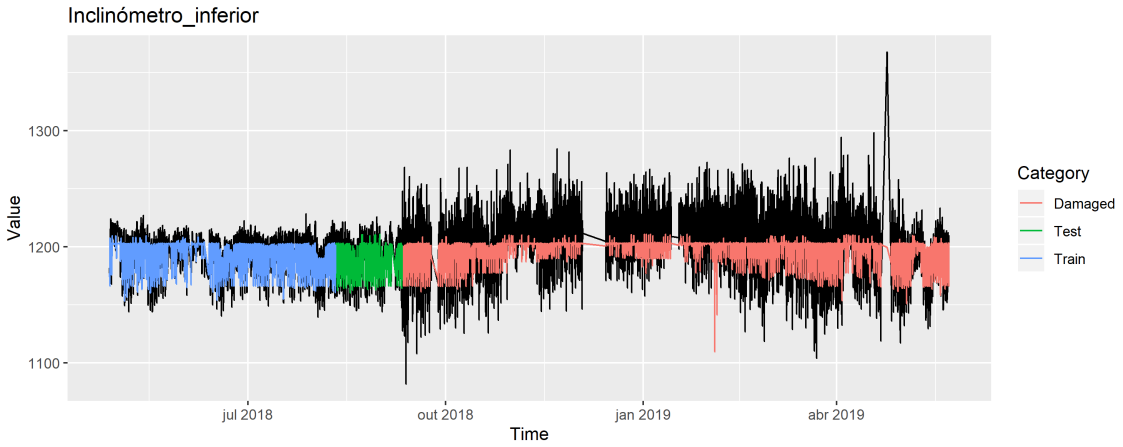


Figure E.13: Artificial Neural Networks predictions for Lower Inclinometer

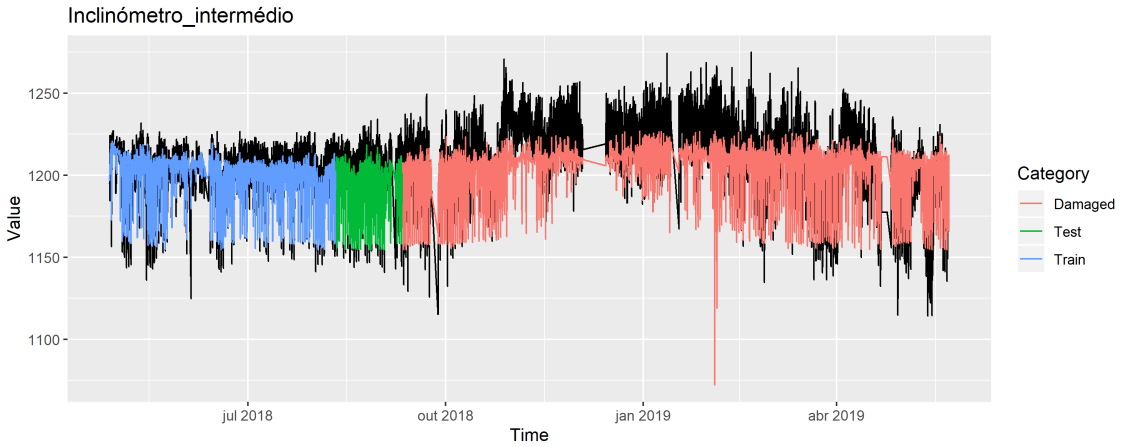


Figure E.14: Artificial Neural Networks predictions for Middle Inclinometer

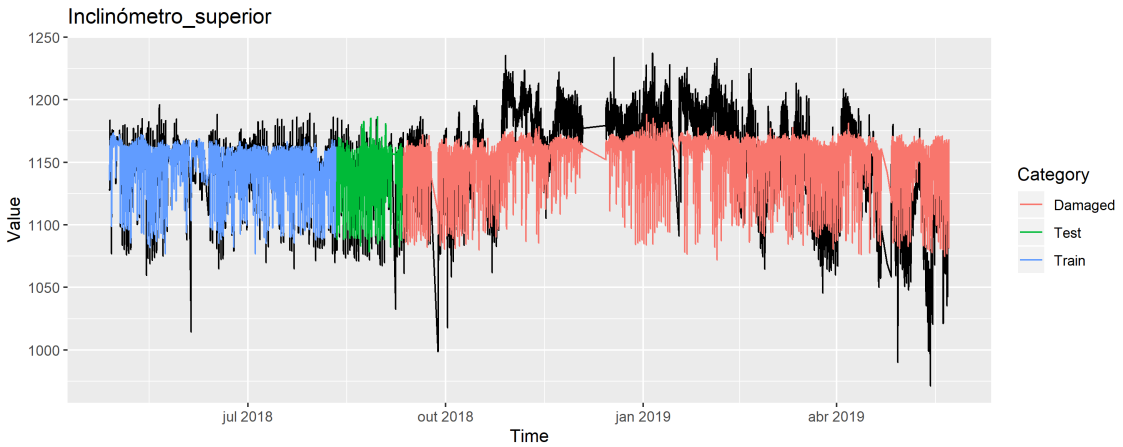


Figure E.15: Artificial Neural Networks predictions for Upper Inclinometer

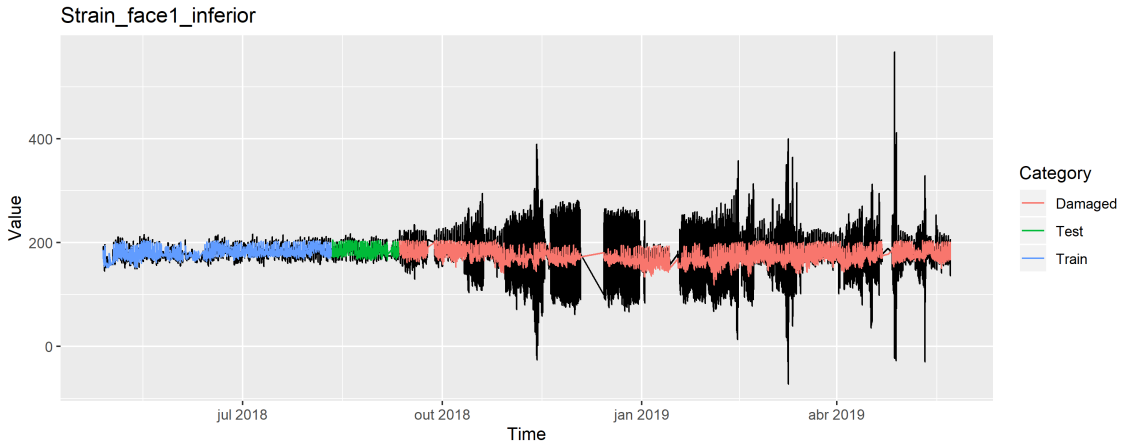


Figure E.16: Artificial Neural Networks predictions for Lower F1 Strain

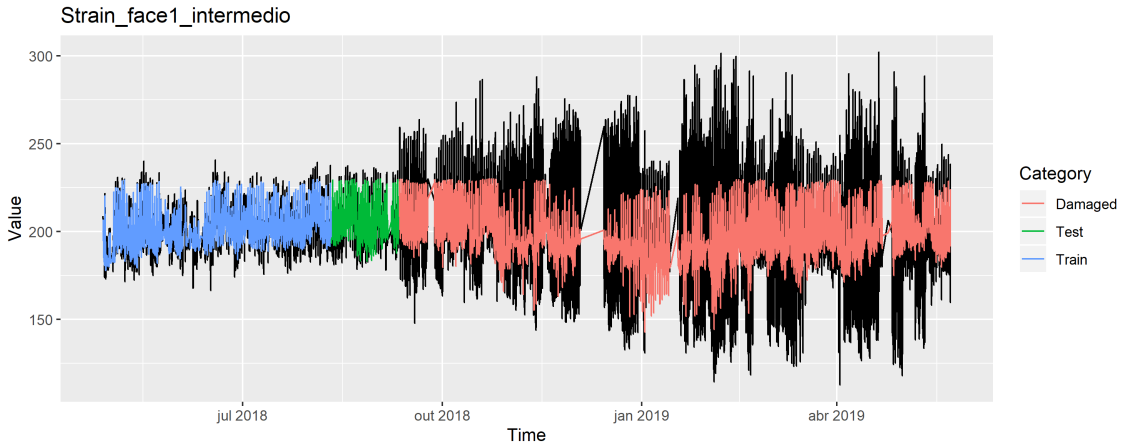


Figure E.17: Artificial Neural Networks predictions for Middle F1 Strain

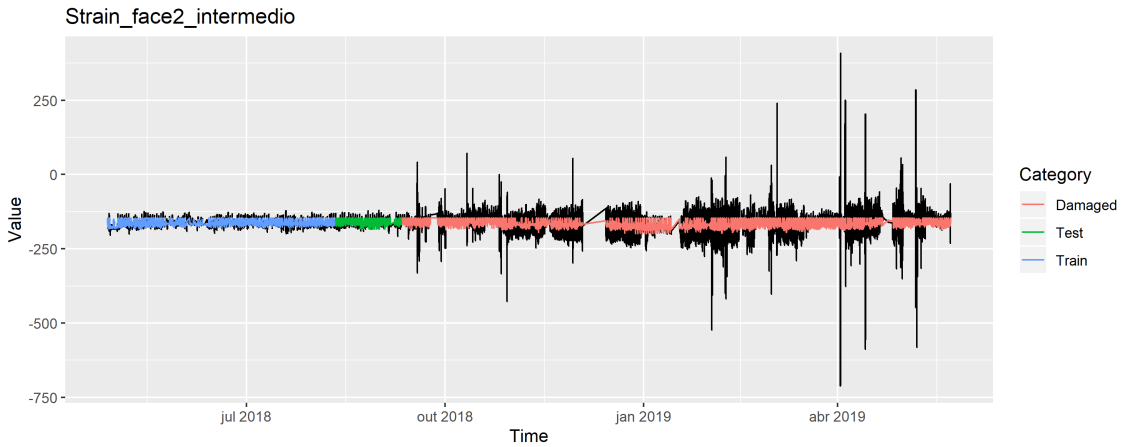


Figure E.18: Artificial Neural Networks predictions for Middle F2 Strain

E.4 CBLSTM Single Output Models

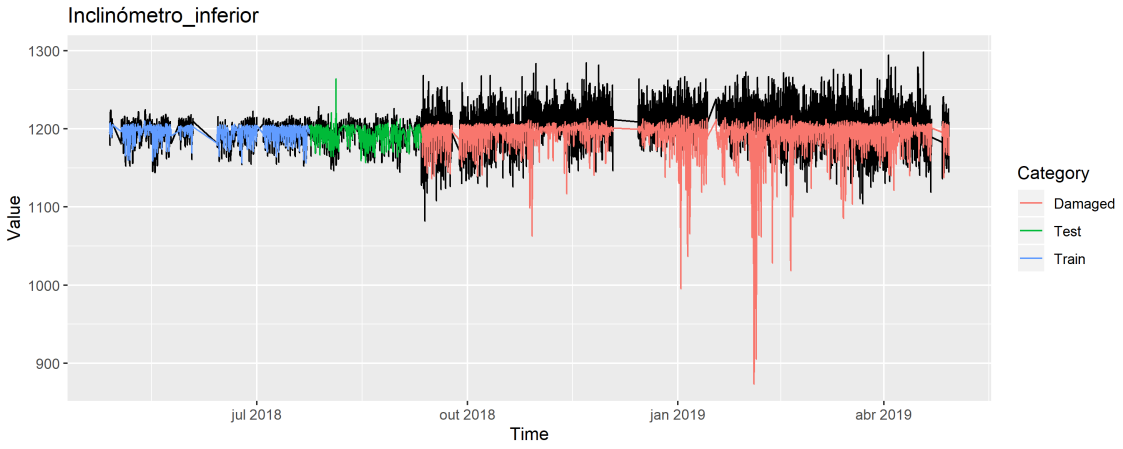


Figure E.19: CBLSTM Single Output predictions for Lower Inclinometer

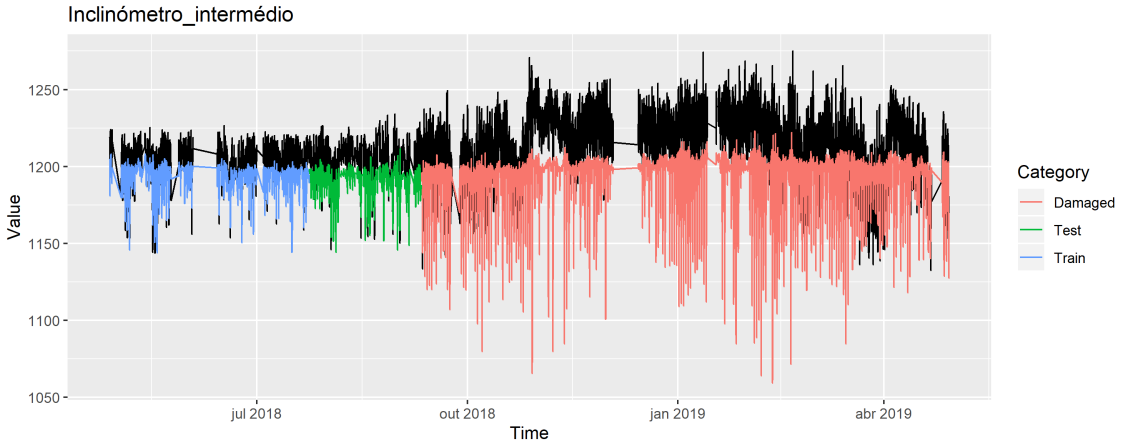


Figure E.20: CBLSTM Single Output predictions for Middle Inclinometer

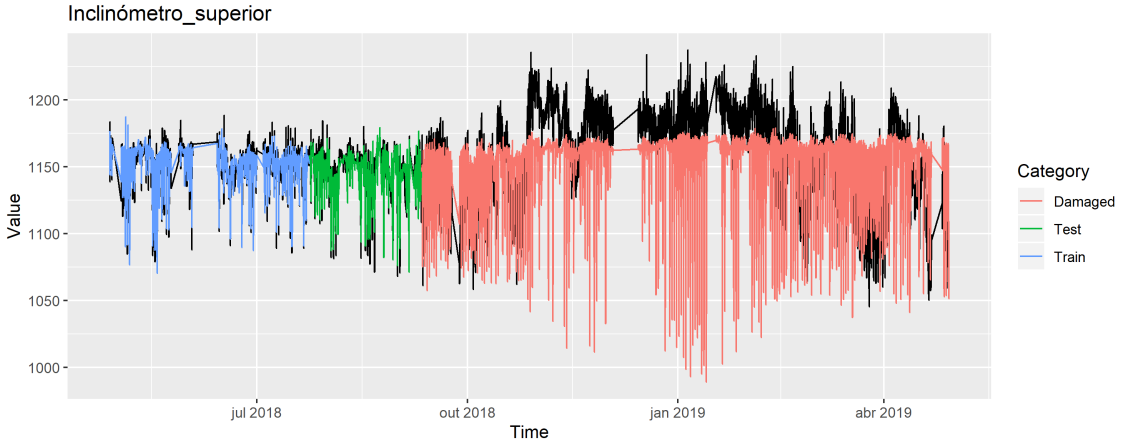


Figure E.21: CBLSTM Single Output predictions for Upper Inclinometer

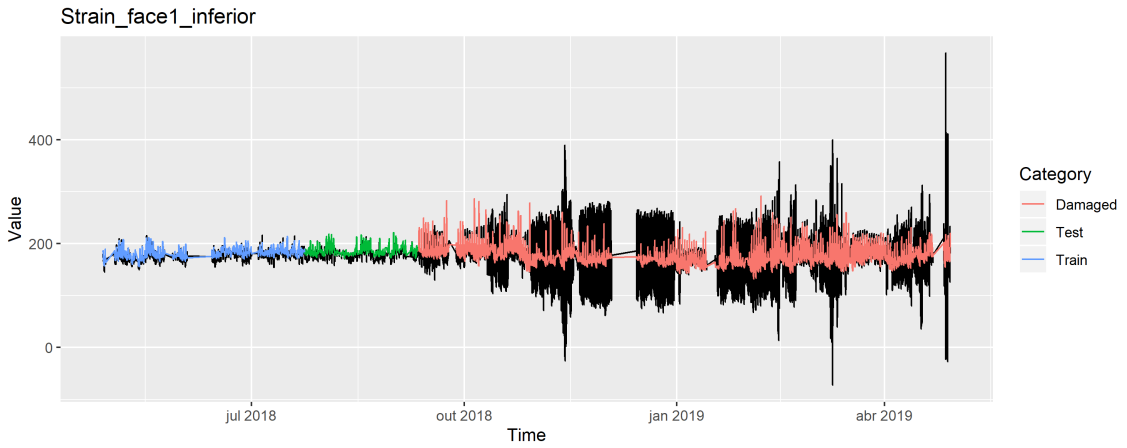


Figure E.22: CBLSTM Single Output predictions for Lower F1 Strain

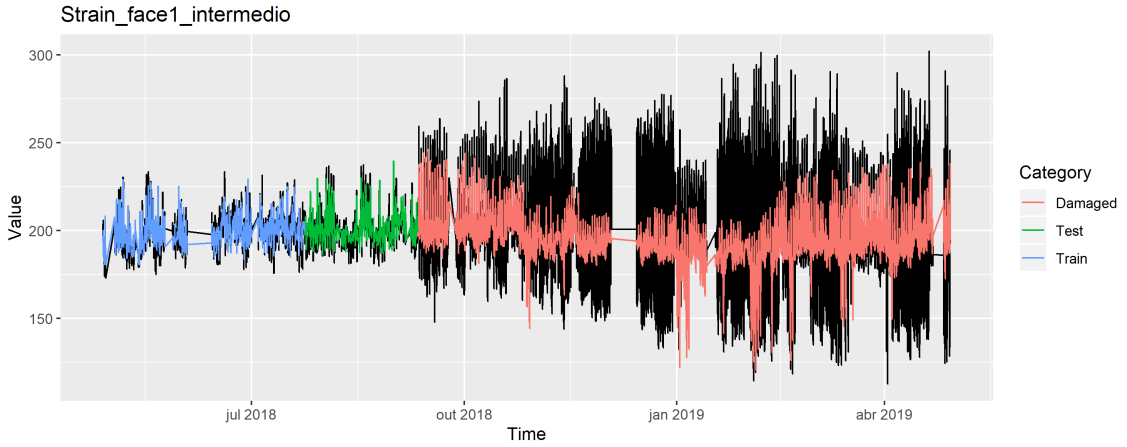


Figure E.23: CBLSTM Single Output predictions for Middle F1 Strain

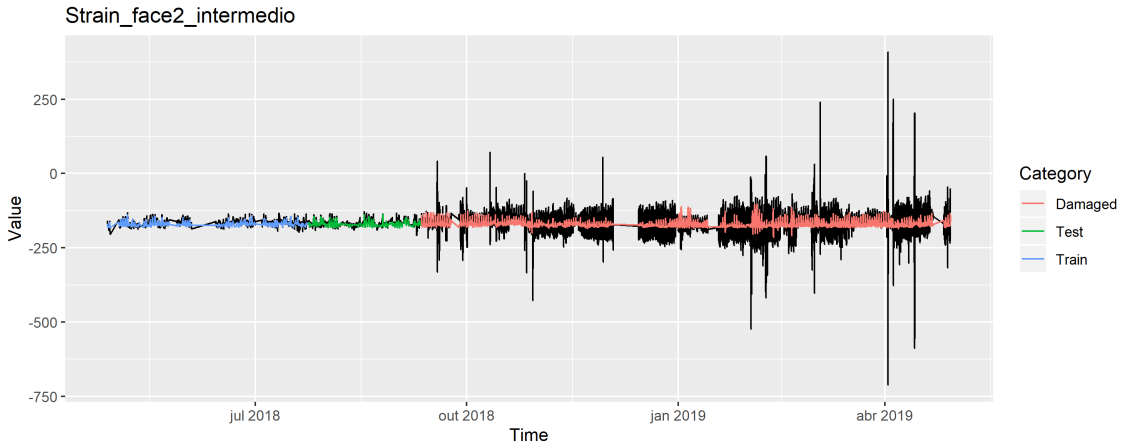


Figure E.24: CBLSTM Single Output predictions for Middle F2 Strain

E.5 CBLSTM Multiple Output Models

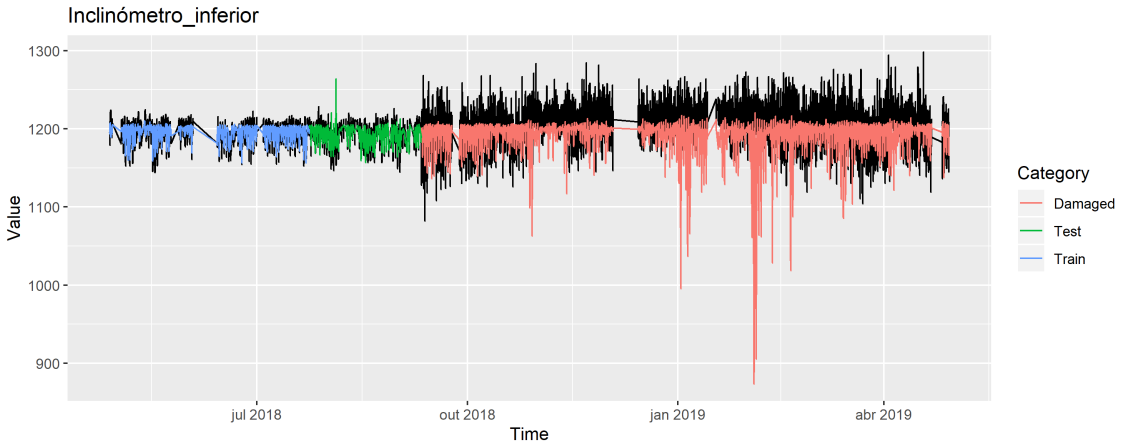


Figure E.25: CBLSTM Multiple Output predictions for Lower Inclinometer

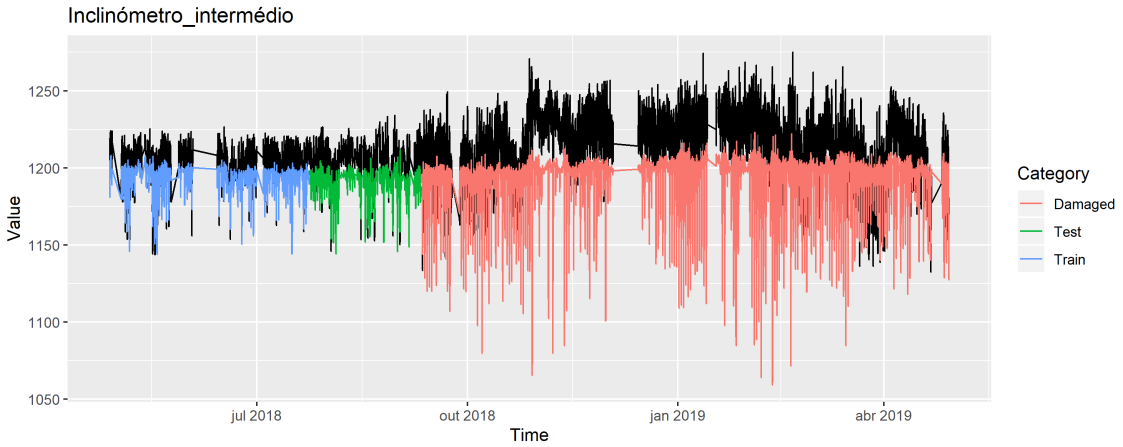


Figure E.26: CBLSTM Multiple Output predictions for Middle Inclinometer

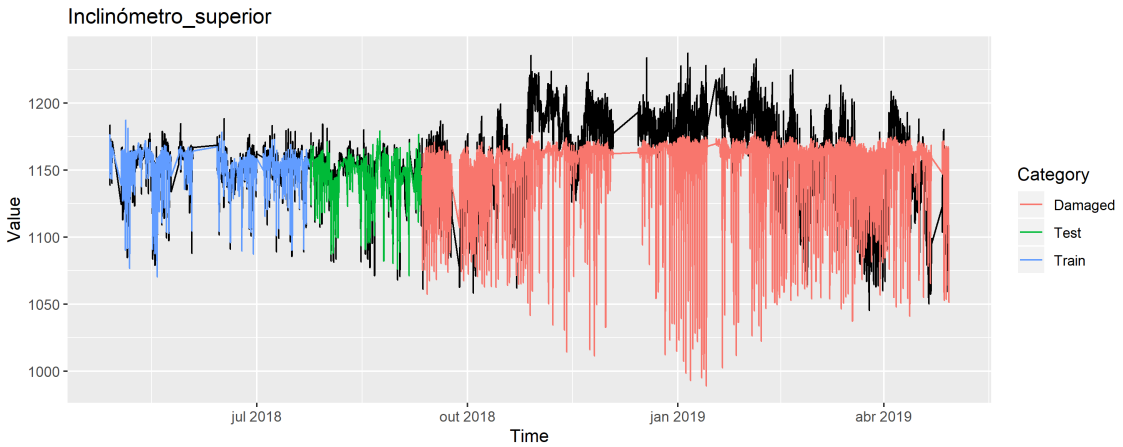


Figure E.27: CBLSTM Multiple Output predictions for Upper Inclinometer

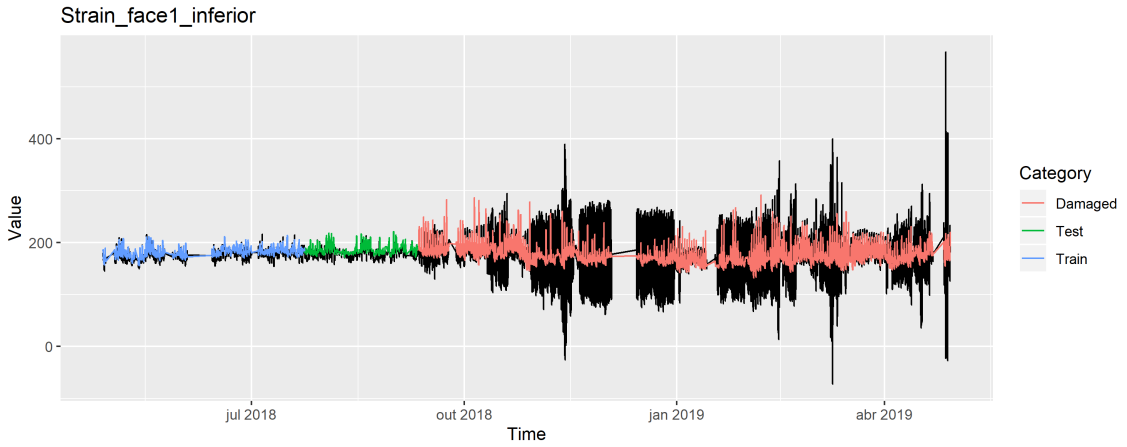


Figure E.28: CBLSTM Multiple Output predictions for Lower F1 Strain

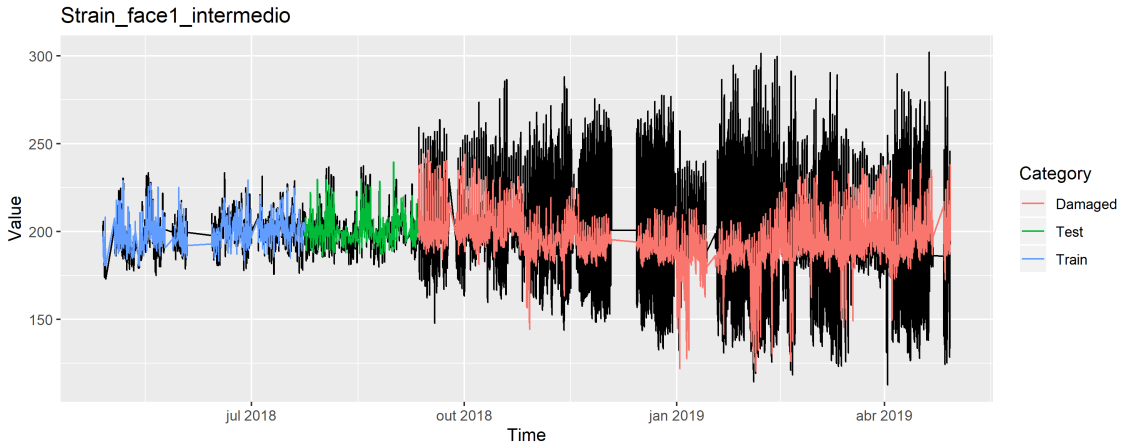


Figure E.29: CBLSTM Multiple Output predictions for Middle F1 Strain

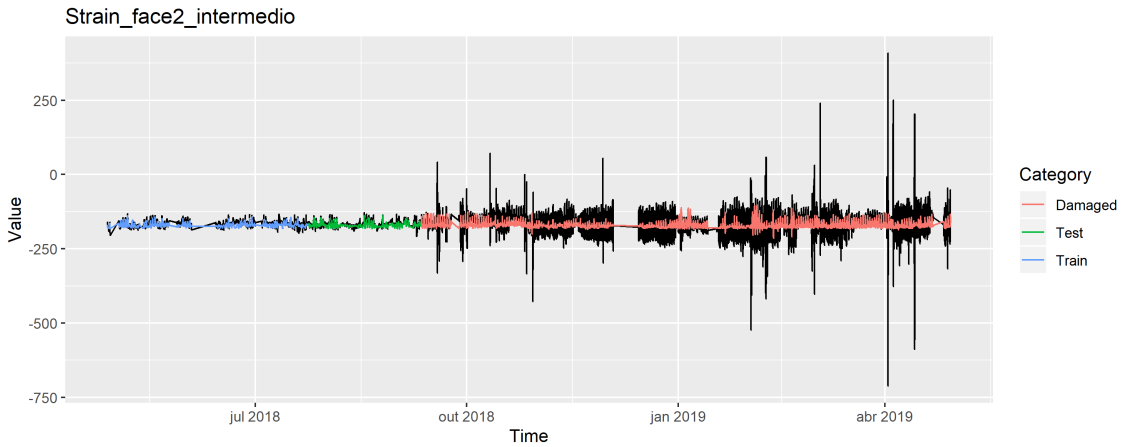


Figure E.30: CBLSTM Multiple Output predictions for Middle F2 Strain

Appendix F

Control Charts Visualization

F.1 Random Forests

F.1.1 Hotelling T^2

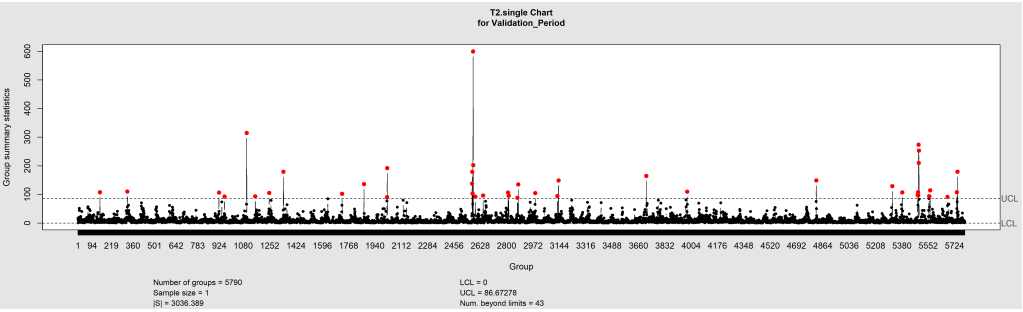


Figure F.1: Hotelling T^2 Control Chart for Random Forests Validating Data

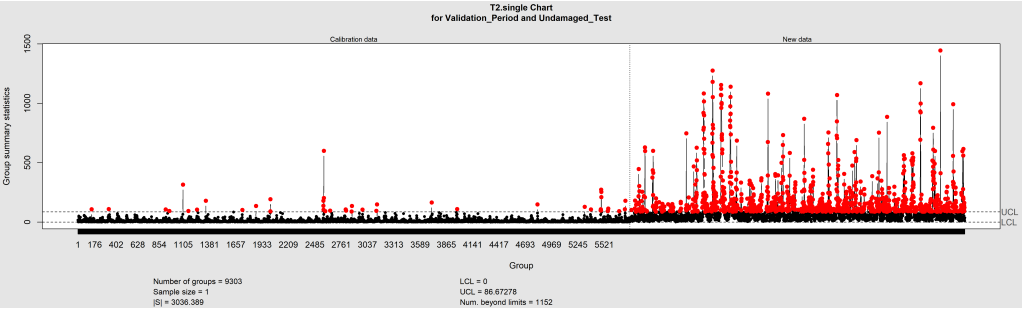


Figure F.2: Hotelling T^2 Control Chart for Random Forests Validating and Testing Data

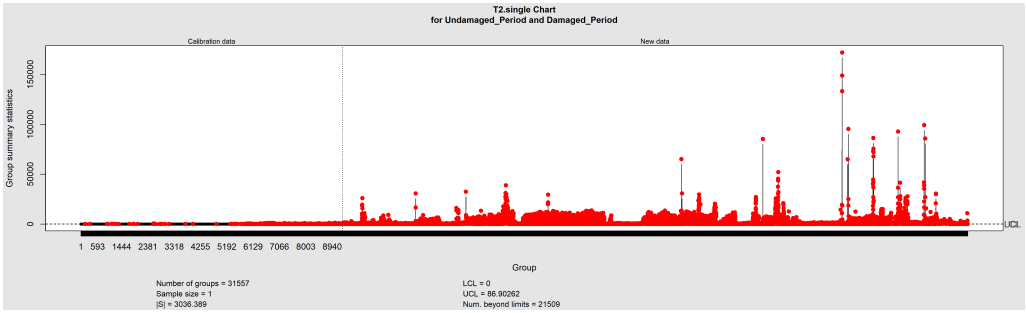


Figure F.3: Hotelling T^2 Control Chart for Random Forests Data

F.1.2 EWMA

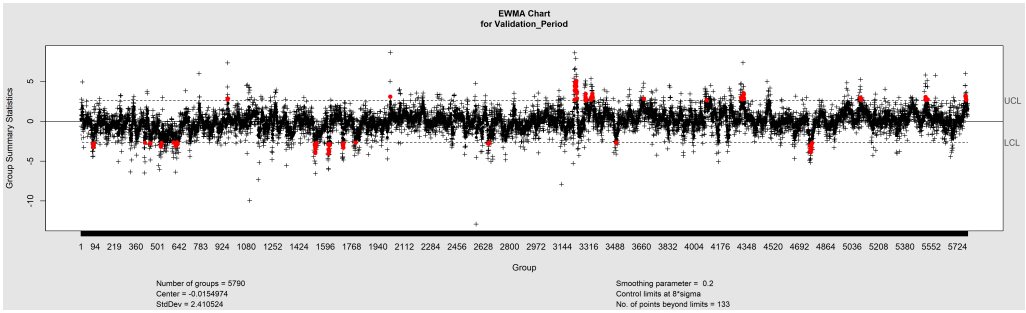


Figure F.4: EWMA Control Chart for Random Forests Validating Data

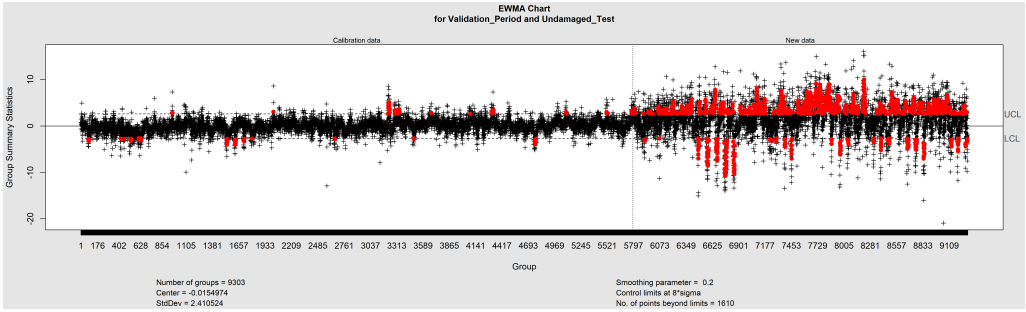


Figure F.5: EWMA Control Chart for Random Forests Validating and Testing Data

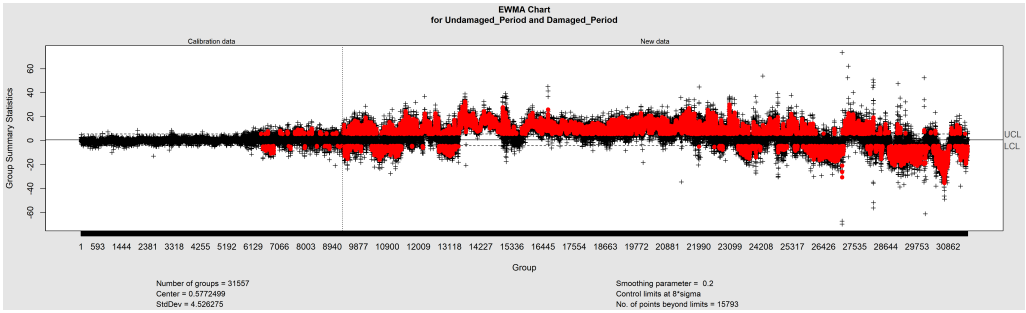


Figure F.6: EWMA Control Chart for Random Forests Data

F.2 Support Vector Regression

F.2.1 Hotelling T^2

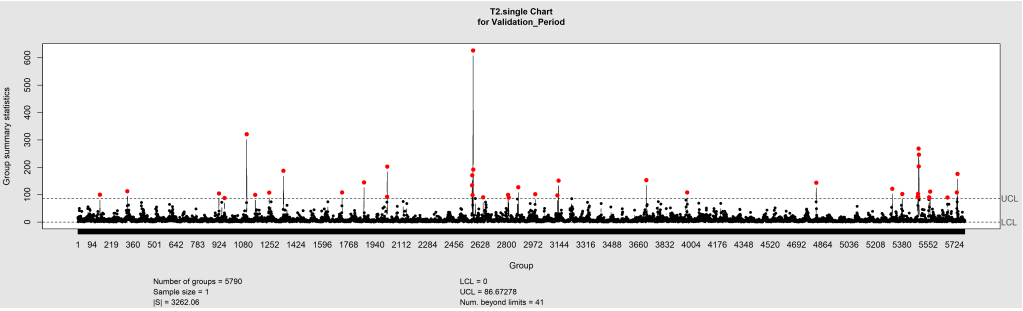


Figure F.7: Hotelling T^2 Control Chart for Support Vector Regression Validating Data

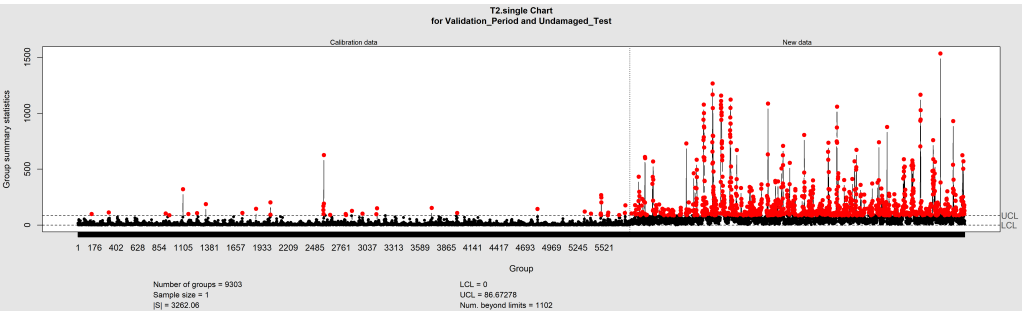


Figure F.8: Hotelling T^2 Control Chart for Support Vector Regression Validating and Testing Data

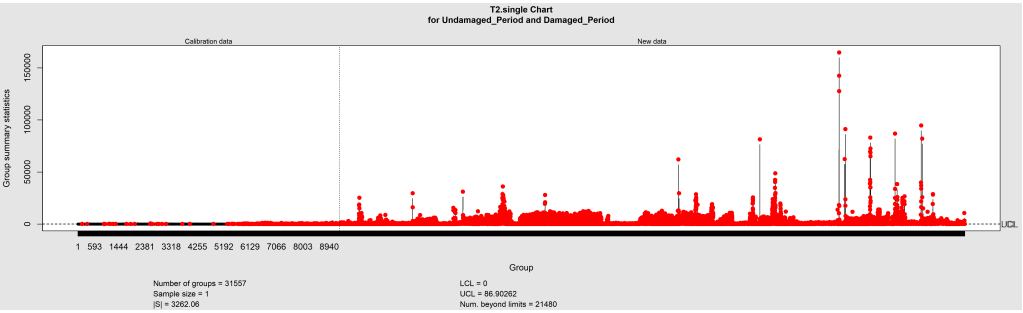


Figure F.9: Hotelling T^2 Control Chart for Support Vector Regression Data

F.2.2 EWMA

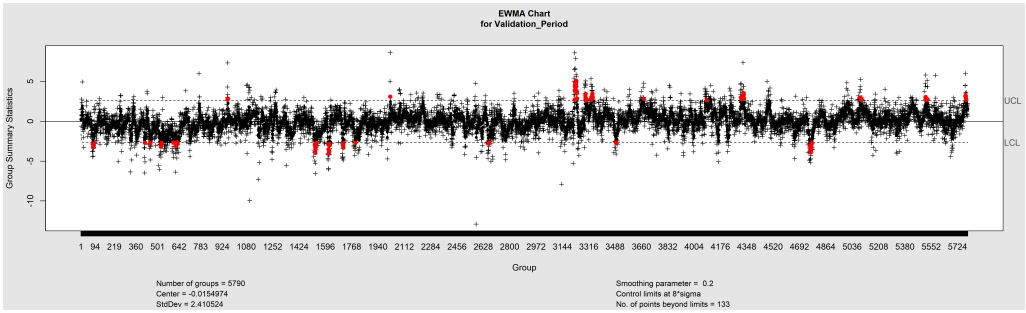


Figure F.10: EWMA Control Chart for Support Vector Regression Validating Data

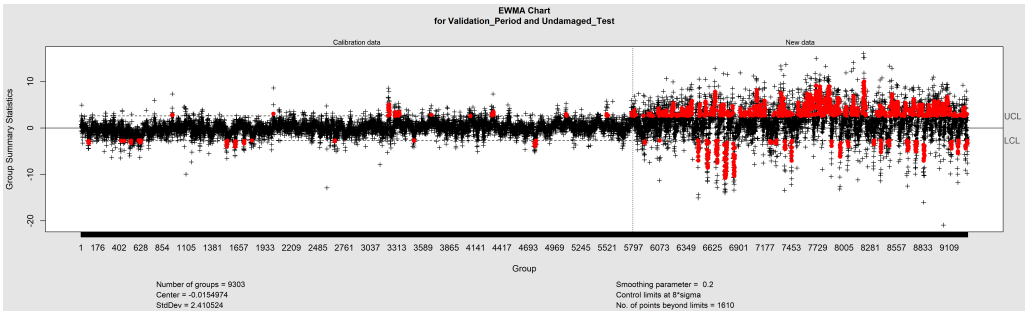


Figure F.11: EWMA Control Chart for Support Vector Regression Validating and Testing Data

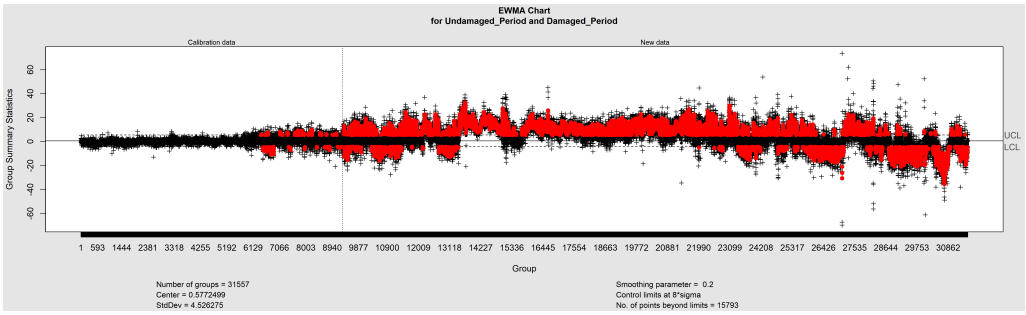


Figure F.12: EWMA Control Chart for Support Vector Regression Data

F.3 Artificial Neural Networks

F.3.1 Hotelling T^2

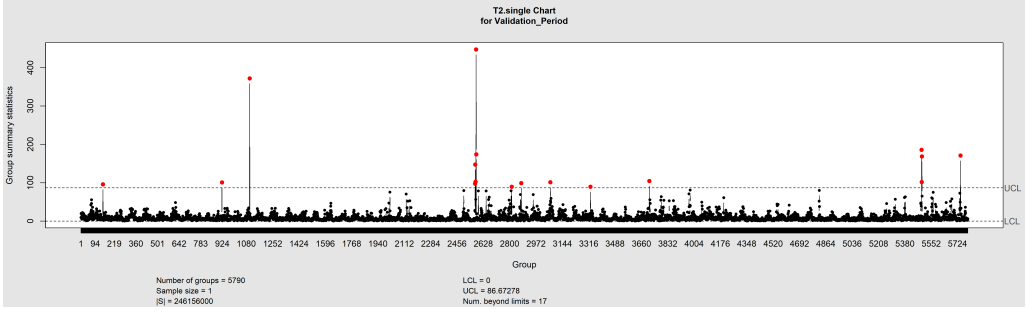


Figure F.13: Hotelling T^2 Control Chart for Artificial Neural Networks Validating Data

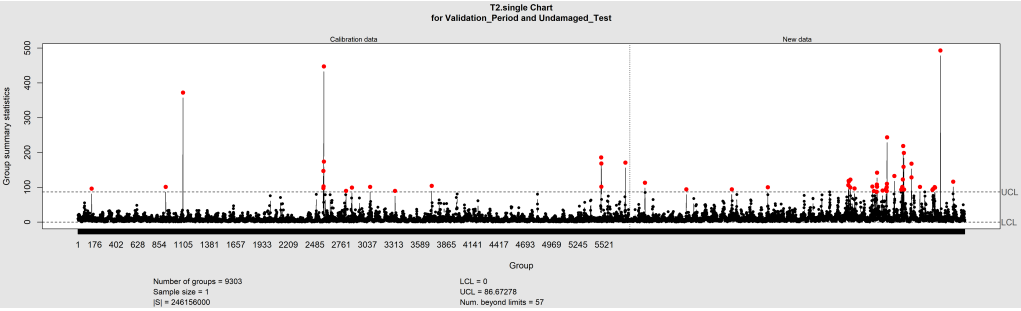


Figure F.14: Hotelling T^2 Control Chart for Artificial Neural Networks Validating and Testing Data

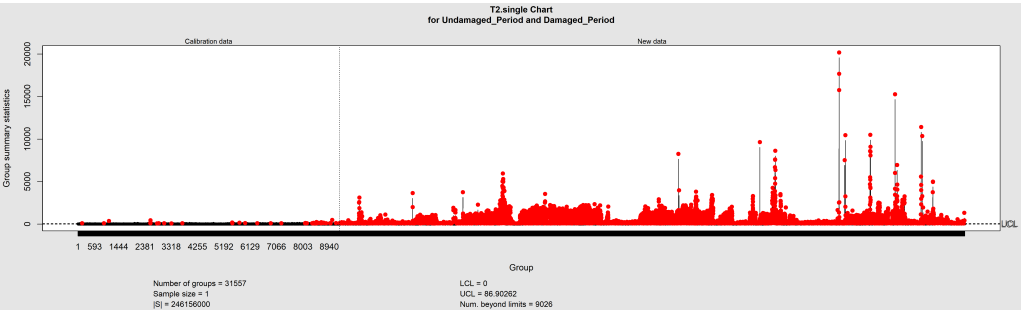


Figure F.15: Hotelling T^2 Control Chart for Artificial Neural Networks Data

F.3.2 EWMA

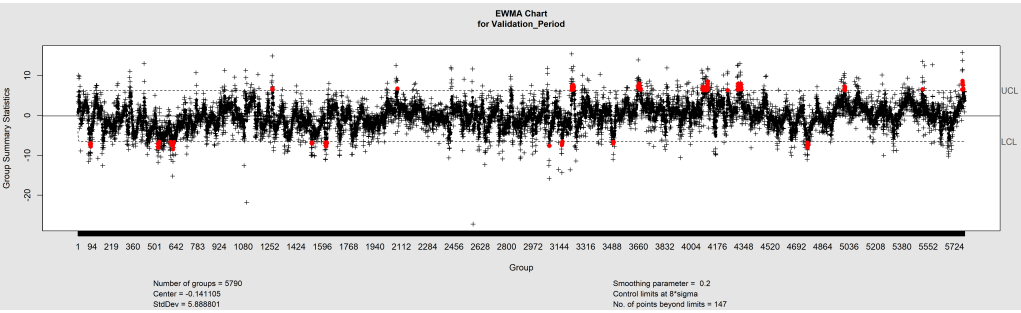


Figure F.16: EWMA Control Chart for Artificial Neural Networks Validating Data

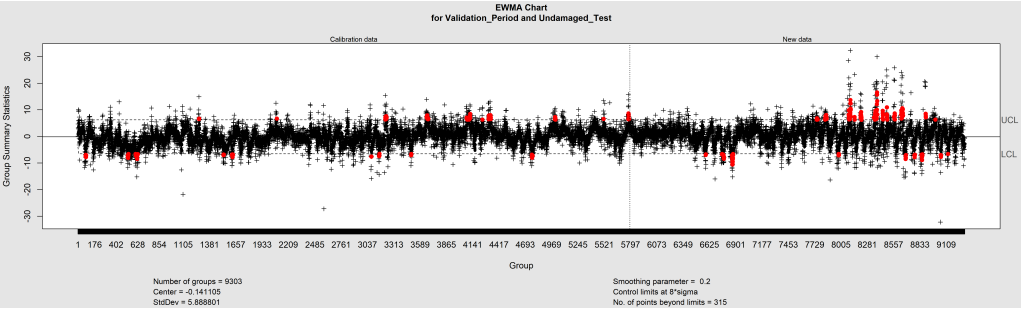


Figure F.17: EWMA Control Chart for Artificial Neural Networks Validating and Testing Data

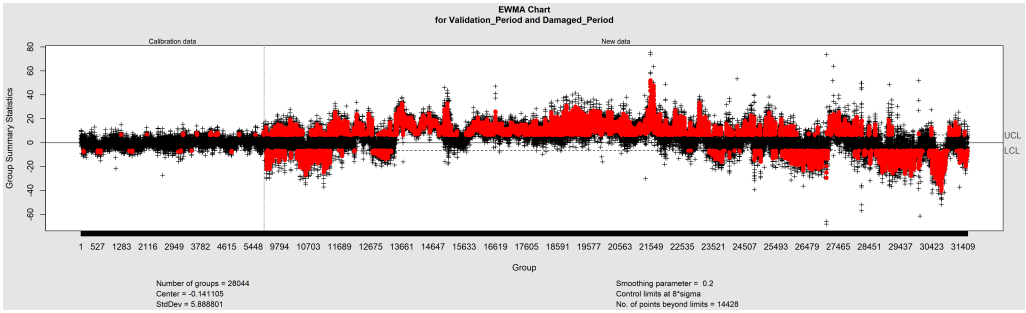


Figure F.18: EWMA Control Chart for Artificial Neural Networks Data

F.4 Averaged Ensemble

F.4.1 Hotelling T^2

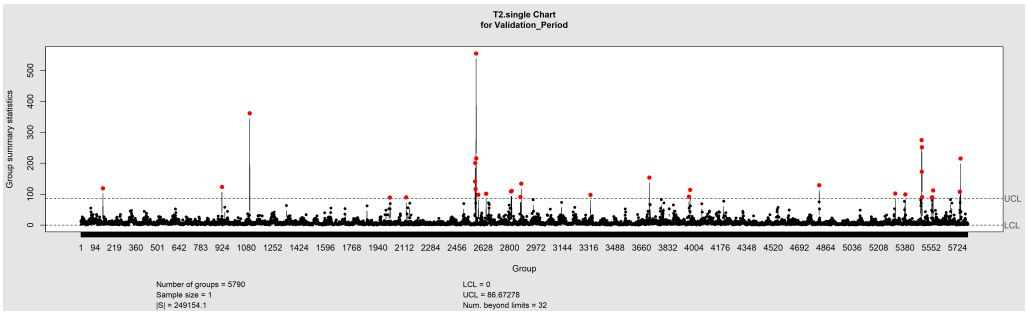


Figure F.19: Hotelling T^2 Control Chart for Averaged Ensemble Validating Data

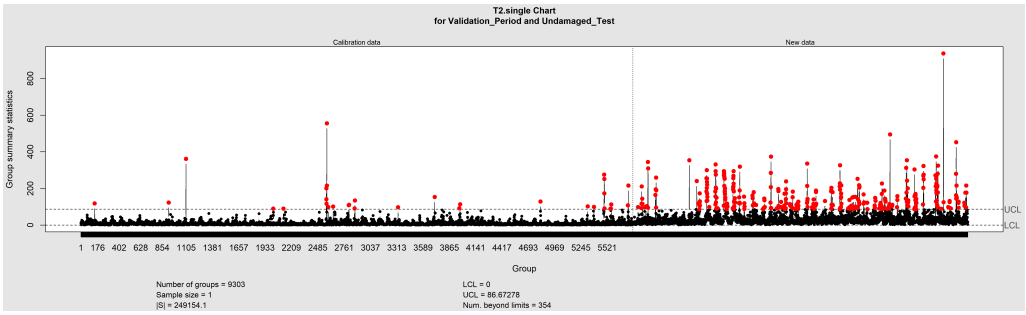


Figure F.20: Hotelling T^2 Control Chart for Averaged Ensemble Validating and Testing Data

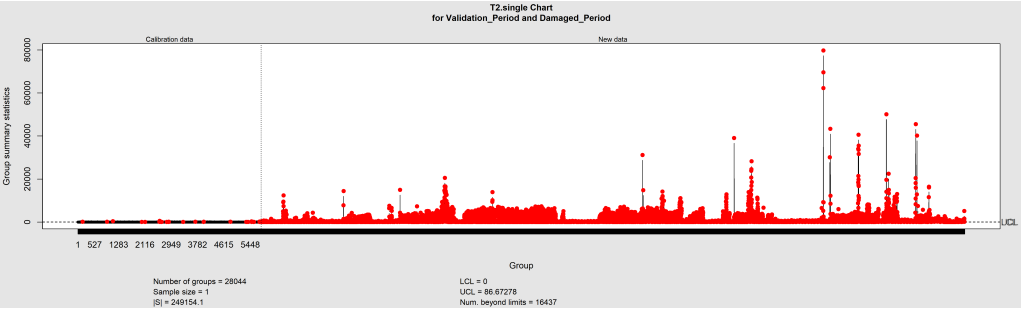


Figure F.21: Hotelling T^2 Control Chart for Averaged Ensemble Data

F.5 Weighted Ensemble

F.5.1 Hotelling T^2

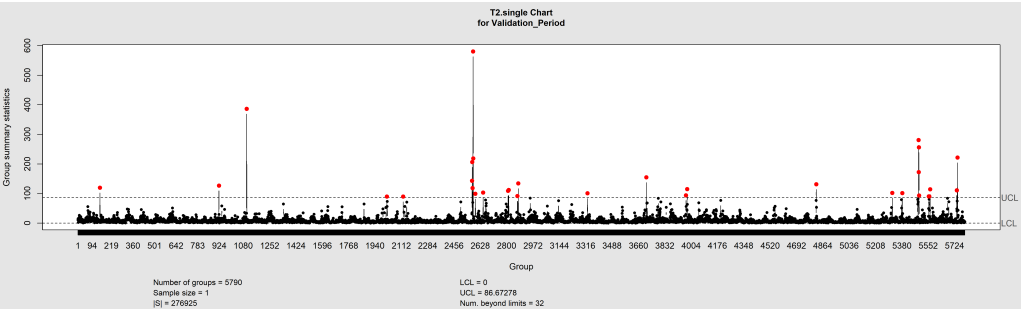


Figure F.22: Hotelling T^2 Control Chart for Weighted Ensemble Validating Data

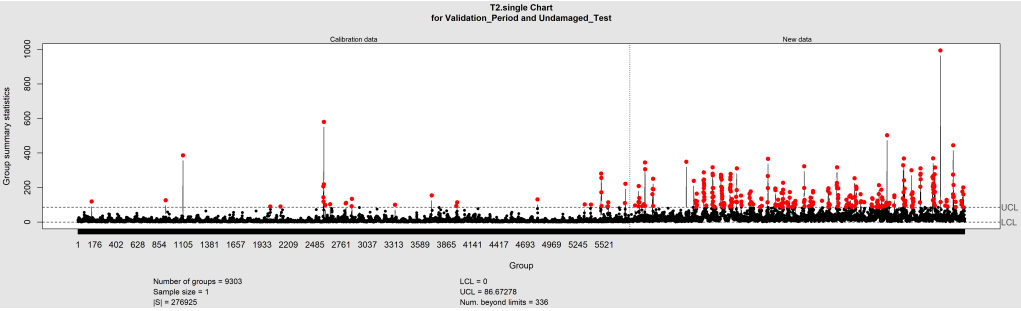


Figure F.23: Hotelling T^2 Control Chart for Weighted Ensemble Validating and Testing Data

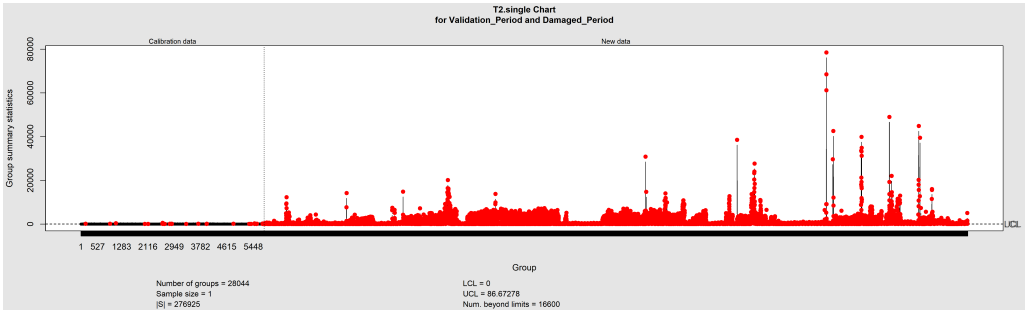


Figure F.24: Hotelling T^2 Control Chart for Weighted Ensemble Data

F.6 CBLSTM Single Output

F.6.1 Hotelling T^2

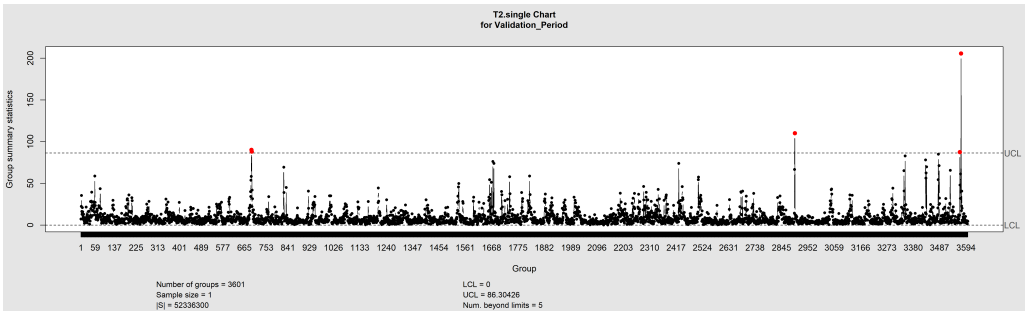


Figure F.25: Hotelling T^2 Control Chart for CBLSTM single output Validating Data

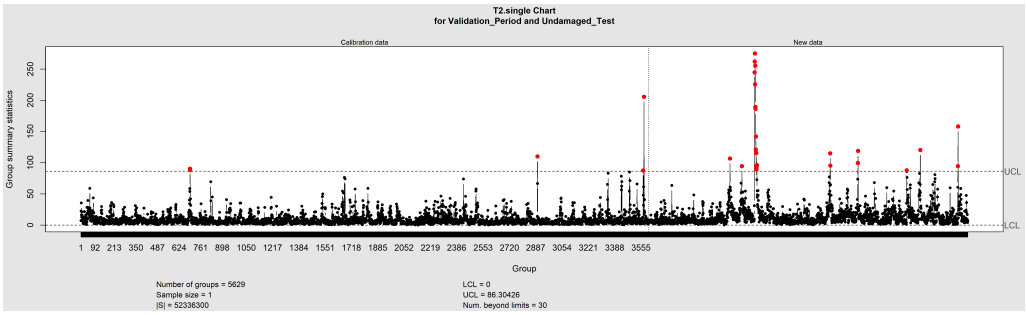


Figure F.26: Hotelling T^2 Control Chart for CBLSTM single output Validating and Testing Data

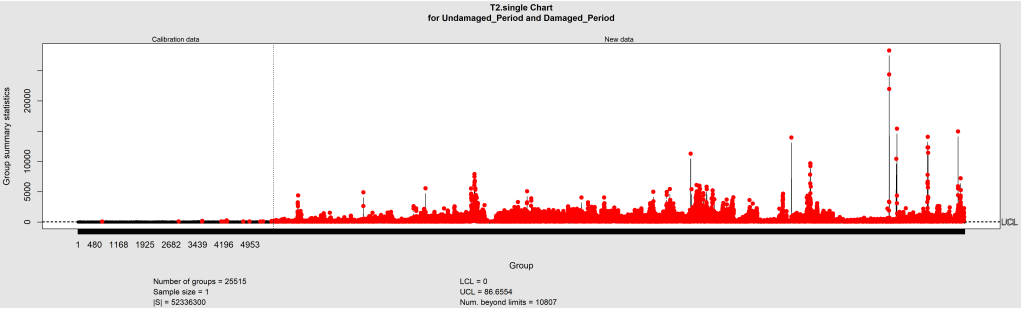


Figure F.27: Hotelling T^2 Control Chart for CBLSTM single output Data

F.7 CBLSTM Multiple Output

F.7.1 Hotelling T^2

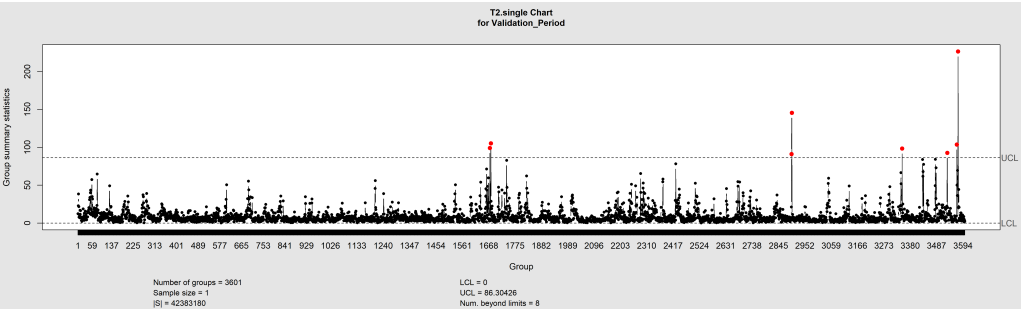


Figure F.28: Hotelling T^2 Control Chart for CBLSTM multiple output Validating Data

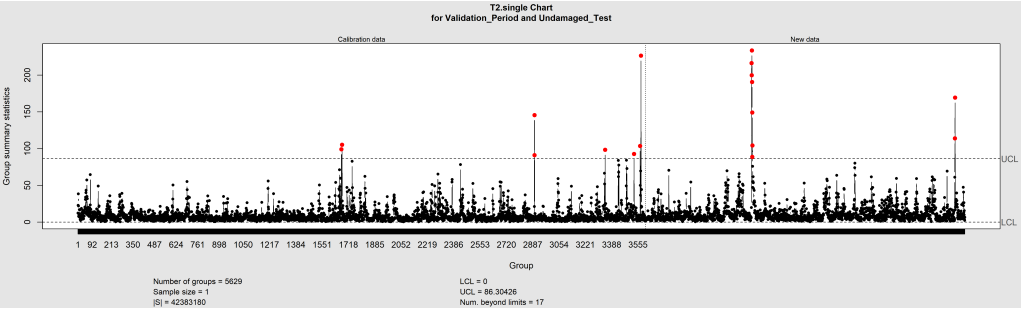


Figure F.29: Hotelling T^2 Control Chart for CBLSTM multiple output Validating and Testing Data

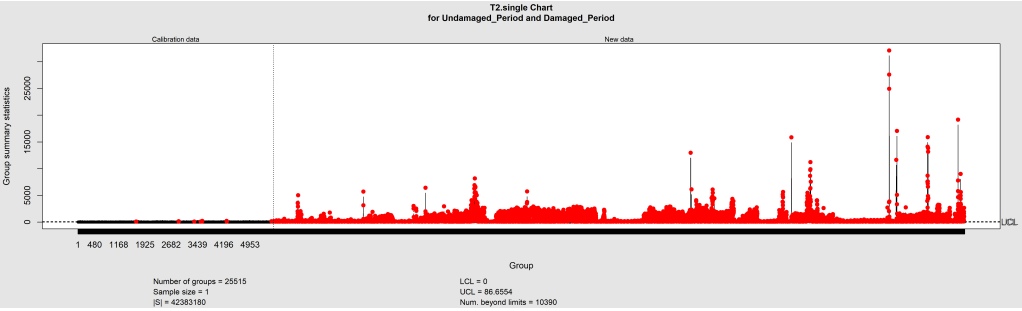


Figure F.30: Hotelling T^2 Control Chart for CBLSTM multiple output Data