U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Master Thesis in Mechanical Engineering

Lamb Waves signal analysis for damage detection in structural adhesive joints

Author:
Vasco Filipe Ferreira Loreiro

Supervisors:
Prof. António Mendes Lopes, Ph.D.
Prof. Lucas Filipe Martins da Silva, Ph.D.
Eng. Gabriel Martins Franco Ramalho

Integrated Masters in Mechanical Engineering - Specialization in Automation

September, 2021

Agradecimentos

Em primeiro lugar, expresso a minha gratidão aos orientadores deste projeto, Prof. António Mendes Lopes, Prof. Lucas da Silva e Eng. Gabriel Ramalho, pela oportunidade de fazer parte deste excelente grupo de trabalho, pela disponibilidade e partilha de informação constante ao longo de todo o semestre.

À minha família, em especial aos meus pais, Cristina e Jorge, e avós, Pompílio e Leonor, agradeço profundamente todo o apoio que sempre me deram, a compreensão e suporte em todo o meu percurso académico, e a educação que me ensinou a ser o homem que sou hoje. Farei sempre o que puder para que se orgulhem de mim.

À Inês Viana, por ser a melhor pessoa que poderia ter ao meu lado, que me inspira todos os dias a ser a melhor versão de mim próprio.

Uma palavra de apreço a todos os meus amigos e aqueles que me acompanharam durante todo o percurso, especialmente ao Rafael Carvalho, à AliveTeam e ao poderoso Fábio Pereira.

Abstract

In recent years the industry has grown in many different ways, with new materials and fabrication methods being developed. This, allied with the availability and lowering cost of sensors has sparked the widespread use of Non-Destructive Testing, which has various advantages over the more classic testing such as tensile, fracture or creep testing. There is also a large advancement in the development of Structural Health Monitoring methods, that use different techniques, specifically Lamb Waves to ascertain the integrity of engineering structures through the analysis of sampled responses to mechanical impulses.

The purpose of this project is to inspect the structural integrity of adhesive joints, as their relevance in the industry expands, specially in the aeronautical and automotive, and with the greater inclusion of composite materials in mechanical design. Methods utilizing time series sensor data for damage detection have shown great promise in classifying the extent of damage present in plate structures when used in union with machine learning algorithms. Despite this success, there is still a lack of robust methods for choosing features that optimize the learning process to classify any damage.

This project aims to assemble a multi-class classification pipeline that takes the raw sensor data, obtained from finite element simulations, extracts features containing meaningful information regarding the data, preprocesses this information, selecting the features according to their significance relative to the classes, and uses them in machine learning algorithms, that will predict the damage class for each instance of testing. The project was developed with data obtained from simulated numerical models, because training and testing machine learning algorithms requires large volumes of data, therefore it would be impracticable to use experimental data. Nevertheless, these models are in every way prepared to process real sensor data, as long as the information is presented in the proper format.

After a brief introduction to adhesives, adhesive joints and their properties, and an initial visualization of the sensor signals and some manually extracted features for context, a powerful time series specialized feature extraction method is implemented, from which over 75 different prominent features and their variations are extracted. Then, by utilizing hypothesis tests, some are selected as relevant for the classification of each damage class, and the Benjamini-Hochberg procedure is applied for the removal of false positives. After this selection stage, the features are visualized with dimensionality reduction techniques, namely Multidimensional Scaling, among others, and are inserted into supervised machine learning algorithms, such as the Random Forest and Gradient Boosting classifiers, where not only is it possible to achieve good classification metrics using all features, but also reveal and isolate which features allow the best differentiation of each damage class.

This methodology accounts for robustness by utilizing different layers of selection and classification, validating the feature relevance in relation to the appropriate set of classes. As such, different damage types and ranges can be detected in this pipeline, as long as the classes are properly defined.

Resumo

Nos últimos anos a indústria tem vindo a evoluir em vários aspetos, com o desenvolvimento de novos materiais e processos de fabrico. Aliada à maior disponibilidade e decrescente custo de sensores, esta evolução potenciou a disseminação de Testes Não-Destrutivos, que possuem diversas vantagens sobre testes mecânicos mais convencionais, tais como ensaios de tração, fratura ou fluência. Existem também grandes progressos no desenvolvimento de métodos de Monitorização de Integridade Estrutural, que utilizam diferentes técnicas, especificamente Lamb Waves, entre outras, para verificar a integridade de estruturas através da análise do sinal amostrado de resposta a impulsos mecânicos.

O propósito deste projeto é explorar Lamb Waves para monitorização e inspeção da integridade estrutural de juntas adesivas, em virtude da sua crescente relevância na indústria, sobretudo aeronáutica e automóvel, e com a ascendente inclusão de materiais compósitos como solução de projeto mecânico. Métodos que utilizam séries temporais de sensores para deteção de danos mostram grande potencial em classificar a extensão dos danos presentes em estruturas em placa, quando usados em união com algoritmos de *machine learning*. Apesar deste sucesso, há ainda uma falta de métodos robustos para seleção de características (*features*) que otimizem o processo de aprendizagem dos modelos para classificar os danos.

Este projeto tem como objetivo construir uma progressão de classificação *multi-class*, que inicia com os dados dos sensores em bruto, extrai *features* que contenham informação significativa sobre os dados, efetua uma fase de pré-processamento em que são selecionadas *features* que sejam relevantes para a triagem das classes, e utiliza-as em algoritmos de *machine learning* supervisionados, que efetuam a previsão das classes de dano para cada iteração de teste. Os dados usados para o desenvolvimento do projeto são obtidos por modelos numéricos de simulação, pois treinar e testar algoritmos de *machine learning* requer um extenso volume de dados, tornando impraticável o uso de dados experimentais. Não obstante, estes modelos estão em todos os aspetos preparados para processar dados de sensores reais, logo que estes sejam introduzidos com o formato adequado.

Após uma breve introdução a adesivos, juntas adesivas e as suas propriedades, e a vizualização inicial dos sinais dos sensores e algumas *features* extraídas manualmente para contexto, é implementada uma poderosa ferramenta de extração de *features* especializada em séries temporais, da qual resultam acima de 75 *features* diferentes e as suas variações para cada sensor. Estas passam por uma série de testes de hipótese, onde algumas são selecionadas como relevantes para a previsão de cada classe, e é-lhes aplicado o procedimento de Benjamini-Hochberg, para a remoção de falsos positivos. Após esta fase de seleção, as *features* são visualizadas com técnicas de redução de dimensionalidade, nomeadamente *Multidimensional Scaling*, entre outras, e são então inseridas em vários algoritmos de *machine learning* supervisionados tais como os classificadores *Random Forest* e *Gradient Boosting*, onde não só é possível atingir boas métricas de classificação, mas também revelar e isolar quais as *features* que melhor diferenciam cada classe de dano.

Esta metodologia apresenta robustez através da utilização de diferentes camadas de seleção e classificação, validando a relevância das *features* em relação ao conjunto de classes apropriado. Deste modo, diferentes tipos e intervalos de dano podem ser detetados nesta progressão, logo que as classes sejam adequadamente definidas.

Contents

List of Figures

List of Tables

List of acronyms

AJPU	Advanced Joining Processes Unit's
ISO	International Organization for Standardization
BSI	British Standards Institution
ASTM	American Society for Testing and Materials
NDT	Non-Destructive Testing
NDE	Non-Destructive Evaluation
SHM	Structural Health Monitoring
LW	Lamb Waves
IDE	Integrated Development Environment
PZT	Piezoelectric / Lead Zirconate Titanate
ML	Machine Learning
DS	Data Science
FEM	Finite Element Method
FT	Fourier Transform
DFT	Discrete Fourier Transform
FFT	Fast Fourier Transform
STFT	Short-Time Fourier Transform
CWT	Continuous Wavelet Transform
HHT	Hilbert-Huang Transform
MDS	Multidimensional Scaling
PCoA	Principal Coordinates Analysis
PCA	Principal Components Analysis
SNE	Stochastic Neighbor Embedding
t-SNE	t-distributed Stochastic Neighbor Embedding
API	Application Programming Interface
FDR	False Discovery Rate
AI	Artificial Intelligence
KNN	k-Nearest Neighbors

1 Introduction

1.1 Background and motivation

This project has emerged as the continuation of the Advanced Joining Processes Unit's (AJPU at INEGI) long-standing research into the Structural Health Monitoring (SHM) of surfaces and adhesive joints with the use of Non-Destructive Testing (NDT) methods, specifically with the deployment of ultrasonic testing based on Lamb Waves (LW).

Research on LW - an elastic disturbance that propagates on thin plate structures with shallow to no curvature - and their relevance to this application has been carried out throughout the last few years, in a collective effort to increase the reliability, integrity and durability of adhesive bonded structures [18], which are present and growing in popularity in different industries, such as the aeronautical and automotive, due to their potential for reduction of weight and cost in relation to mechanical fasteners [19]. As component reliability and performance is paramount in these fields, the ability to identify and locate defects without originating any damage to the structure or component under testing is definitely worthy [20].

LW are a prime candidate for NDT, since they propagate over long distances with slight attenuation and have the capability of interacting with various types of material discontinuities and defects [21]. However, the intrinsic nature of LW propagation makes the interpretation of their characteristics more difficult, since they invariably excite more than one propagation mode at any given testing frequency [20], and interaction with defects present in the medium results in very complicated time-based response signals arising from the sensors. Hence, the opportunity to apply Machine Learning (ML) to the time series signals should be explored, by extracting meaningful features and inputting the information into classification algorithms, in order to train them to predict damage.

1.2 Objectives

As it was mentioned, this project is based on previous work developed by AJPU, namely the development of finite element models with `ABAQUS` that simulate the propagation of LW in designated media with different defects and damage types. The wave signals are generated by a single piezoelectric actuator's impulse, and whose resulting responses - small mechanical vibrations - are measured with piezoelectric sensors with specified positions. There are two main objectives to this project:

- Applying dimensionality reduction techniques in an automated manner, putting in place an automatic feature extraction method that takes the raw time series, resulting from the simulated LW response signals, and outputs a vector of features;
- With the extracted features, training and testing supervised machine learning (ML) classification algorithms, that are designed to detect/predict which damage class the signal originated from.

The ultimate goal is to assemble a ML pipeline from start to finish, designed to receive raw data from sensors, either from a testing setup or a structure, and classify what type of damage they have, if any. The data used for the development of the project is obtained from simulated numerical models, since training and testing ML algorithms requires large volumes of data, and therefore it would be impracticable to use experimental data. Nevertheless, these models are in every way prepared to operate with real sensor data, as long they are properly preprocessed and presented in the same format.

1.3 Research methodology

The first part of the thesis, being a theoretical introduction to the subject of adhesives and adhesive joints, followed a typical approach, with literature research and the review of the state of the art on the specific topics of SHM, NDT and LW, and even some topics on Data Science (DS) and ML that support the practical procedures ahead. As for the implementation of the feature extraction and classification models, it was very much a practical endeavor, using the `Python` open-source programming language on the `Spyder` integrated development environment (IDE), and with a hands-on approach from beginning to end, understanding the libraries and the inner workings of the IDE to program the different parts originally planned. All of this information can be found online, through the official documentation of the `Python` libraries used, which are very informative and easy to absorb, and so that was the main source of information to the development of the programming project itself. When in comes to the specific methods carried out throughout the project, their statistic and scientific concepts were researched as they came up, and so they are explained opportunely whenever they appear.

1.4 Thesis outline

This thesis is divided into six parts. The present introductory chapter contains a brief description of the problem addressed and the tools used to approach it, as well as the background and motivation of this project and its main objectives.

Chapter 2 contains a review on adhesive joints, SHM, NDT and LW.

Chapter 3 comprehends a review of the state of the art and use of DS methods and concepts such as ML algorithms, Dimensionality Reduction techniques and the details associated with handling large amounts of data, as well as some representation of the signal data that will be processed.

Chapter 4 is the start of the practical part of the project, reporting the beginning of the construction and implementation of the ML pipeline that processes the data, namely the automated feature extraction and selection part. It also contains the theoretical support of what is being done.

Chapter 5 presents the back-end of the pipeline, where the preprocessed and selected features are inserted into several supervised ML algorithms that then predict the different

damage classes. Also contains some theory behind the models.

Chapter 7 presents the main conclusions that resulted from the project's development and final results, and indicates some suggestions for further work on these topics, not only what could be done to improve the results obtained, but also what lies ahead in terms of expanding the scope of the project's application.

2 Strucutural Adhesives

2.1 History and introduction to adhesive joints

"When a plate of gold shall be bonded with a plate of silver or joined thereto, it is necessary to beware of three things, of dust, of wind, and of moisture; for if any come between the gold and silver they may not be joined together; and therefore it is necessary to bond these two metals together in a full clean place and quiet." [22]

The quote presented comes from a massive medieval encyclopedia, *De Proprietatibus Rerum* - "The Properties of Things" - compiled around 1250 A.D., by Bartholomaeus Angelicus, and it clearly shows that, even then, there was a concern about the effectiveness of the adhesion phenomenon and the knowledge of some basic elements that undermine the perfect conditions for a successful adhesive bond.

Adhesives have been known and used for centuries, the sensation of stickiness is among the commonplace experiences of humanity, a phenomenon as natural as resin oozing from a pine branch, that immediately presents itself as useful to join a tree branch to a stone point and form a spear [23]. Until the beginning of the 20^{th} century, adhesives obtained from natural products, such as skins, fish, milk and plants, or natural materials that intuitively present themselves as sticky. Nowadays, there are very few products anywhere that do not use adhesives in some aspect, since they are found in all types of industries, from automotive to construction, electronics and others. But its biggest advocate is the aeronautic industry, that continuously expands its use, as more composite materials are introduced [1].

An adhesive can be more formally defined as a material which, when applied to surfaces, can join them together and resist separation. Adhesive is, therefore, the general term which covers materials like glue, cement, paste, and others [4]. As for adhesion as a phenomenon, it is commonly described as a state where two bodies are stuck together, in a simple and straightforward manner. Accepting the ASTM D 907 definition that incorporates more technical concepts, it is "the state in which two surfaces are held together by interfacial forces, which may consist of valence or interlocking forces or both" [23], and that state is in other words the attraction itself that results from intermolecular and interatomic forces established between two surfaces at the interface [5].

Adhesives based on synthetic polymers, such as epoxy resins, have steadily been considered as design solutions in different industries, and were approached as a viable alternative to methods such as bolting, riveting, welding and brazing, more traditionally used, thanks to their ability to adhere to most materials easily. This process eventually led to the development of this type of material, making them stronger and more reliable, bringing forth the concept of structural adhesives, able to resist substantial loads and actively contribute to the stiffness and strength of the structure [4]. Apart from structural applications, adhesives can also be considered as sealants, for instance in all sorts of liquid

containers, as well as be used to attach surface coatings, like ceramic tiles [4].

An adhesive joint is the finished connection of two surfaces, made of similar or different materials, through the use of adhesives. The materials that are bonded by the adhesive, before bonding, are called substrates, and after bonding they are referred to as adherends, a term first coined by de Bruyne in 1939 [4]. The region linking the adherend and the adhesive is the interphase, whose chemical and physical characteristics critically influence the mechanical properties of the adhesive bond itself. Not to be confused with the interface also known as the boundary layer, that resides within the interphase (where various interfaces connecting different materials can exist), which is the plane defined by the contact between the surface and the two materials, where during the formation stages of the bond, the intimate molecular contact is created. There is also the possibility of applying a primer, a substance usually applied on the substrates in the surface preparation steps, to protect them, to improve adhesion in general [1].

Adhesive joints provide a wide variety of advantages when compared to conventional mechanical fasteners [1]:

- More uniform stress distribution along the bonded area - this leads to a good resistance to dynamic solicitation as well as load transmission and higher stiffness, as illustrated on Figure 1;
- Reduction of the weight on the structure and consequently, the cost;
- The ability to bond dissimilar materials, with different coefficients of thermal expansion;
- The ability to bond thin sheets of material - one of the major applications of adhesives;
- Good damping properties, leading to high fatigue strength and good dynamic performance;
- The possibility of automated adhesive application;
- Regular and seamless contours and finishes, due to the absence of drilled holes and welding marks;
- Intimate contact of the bonded surfaces, protecting them from corrosion.

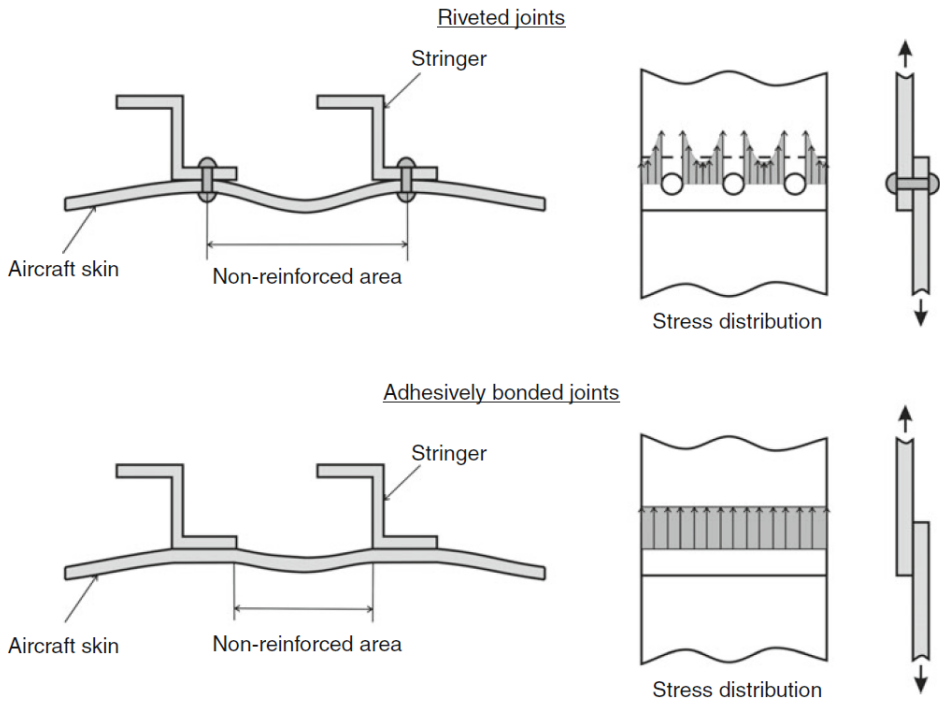
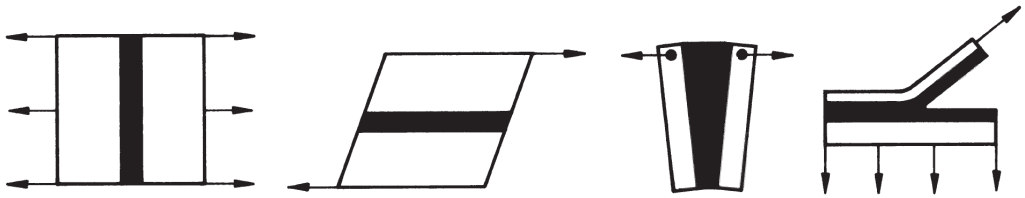


Figure 1: Advantage of adhesively bonded joints in terms of stiffness (left) and stress distribution (right), compared to riveted joints. [1].

However, some disadvantages should also be considered [1, 4]:

- The need to avoid peel and cleavage stress because they create a load concentration in a small area, resulting in low strength in that area;
- Limited resistance to extreme environmental conditions, such as high temperature and humidity, due to the polymeric nature of the adhesive;
- The necessity of fixing tools to maintain the substrates in position, since the bonding is usually not instantaneous;
- The requirement of temperature for the hardening of a wide variety of adhesives;
- The necessity of extremely careful surface preparation to ensure good adhesion;
- Permanent solution, since one adhesive bond is not capable of being dismantled and re-assembled;
- Designing adhesive joints tends to be a complex task, since there are no simple rules such as in the use of mechanical fasteners;
- Difficult quality control - that could change with NDT techniques.



(a) Normal. (b) Shear. (c) Cleavage. (d) Peel.

Figure 2: Types of stress on adhesive joints. Adapted from [2].

Different mechanical loads applied to the adhesive joint will cause different types of stress, the usual ones being shown in Figure 2, of which the most common is **shear**, because it is the best suited for adhesives having their highest performance, as opposed to **cleavage** and **peel**, that should be avoided at all costs. *"Peel is the hated enemy of the joint designer"* [4]. This can be achieved by altering the loading system, by changing the adhesive joint's configuration and geometry, improving the load-transfer and minimizing stress concentrations and peel. Figure 3 shows some of the most frequently employed joints.

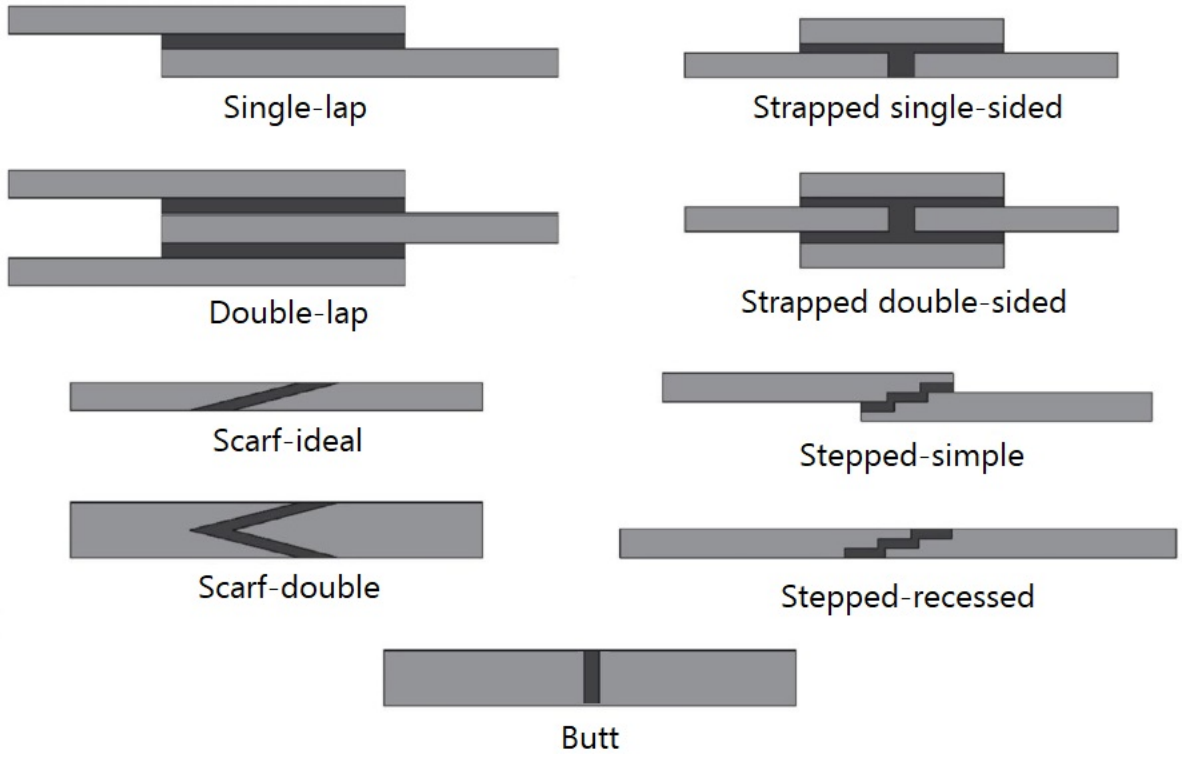


Figure 3: Common engineering adhesive joints. Adapted from [3].

So it can be easily assessed that, if the choice to use an adhesive as a project solution is correct, the main problems that arise from its use are related with the detail of preparation and complex design needed, as well and the overall time and conditions indispensable to a strong bond. The continuous evolution of automation technologies can be of great service

to combat these problems, as well as a way to tackle the difficulties in quality control through NDT, which in one of the main points addressed later in this report.

2.2 Classification of adhesives

The primary purpose of the adhesive is to be wetting and spread on the surfaces of the substrates, fill the gap between them and subsequently form a permanent, coherent bond, which implies that the first stage requires the adhesive to be applied in the liquid phase, that then solidifies with certain mechanical characteristics. The main type of classification of adhesives categorizes them based on the way that this phase transition happens [23], but they are also defined and grouped by their chemical nature, among other criteria, such as the nature and the functionality of the polymer base, or even the functional type, that is the design purpose of the adhesive that determines the properties required to be successful, since these can be adjusted even when formulated from the same type of polymer.

Methods of phase change/ Setting mode

With the exception of pressure-sensitive adhesives which do not change phase during application, adhesives are applied as low-viscosity liquids that wet the adherend surfaces. These liquid state adhesives are usually obtained by three methods, that lead to different ways of forming the solid adhesive bond: dispersing/dissolving the adhesive material in a solvent, that then solidifies as the solvent evaporates; heating of the solid adhesive, that then solidifies by cooling down; starting with and adhesive material as a liquid monomer, that then polymerizes by chemical reaction [1].

These activation methods are the base of this classification scheme, that divides the adhesives as follows [1, 23]:

- **Solvent-based adhesives / Loss of solvent** - The polymer or polymer blend is eventually dissolved or dispersed in a solvent carrier. This solvent could be water, amino-resins, starch or protein glues (among others), but it is more usual to use organic solvents. An added advantage of an organic solvent is that it dries faster. On the other hand, such solvents are more expensive than water, are generally inflammable, irritant/toxic, and the release of solvents to the atmosphere, as well as health and safety considerations have been accelerating a movement away from these solutions;

- **Hot melt adhesives / Cooling** - The primarily mechanical mechanical bond achieved with this types of adhesives results from solidification, since they usually are solid below 79°C. They are heated above this temperature, to the range of 149-188°C when preparing for application, where they become viscous fluids and easily wet the substrates, that have to withstand these high temperatures. Upon

cooling, they solidify rapidly, and as they are mainly constituted of a thermoplastic polymer (with possible added extenders/fillers), the melting/resolidification process is repeatable, which allows a reworking of the bonded parts, if needed;

– **Chemical Reaction** - In this case, there is a chemical change rather than a physical one. Such reactive adhesives may be single-part reactive liquids and rapidly convert to solids when exposed to designated energy sources, or two-part systems, that require the reactants to be stored separately and be mixed shortly before application. They will either require heating (Heat Activated adhesives) or exposure to an electron beam (EB) radiation, or either ultraviolet (UV), visible or microwave radiation (Radiation-Cured adhesives) to perform the reaction. These UV/EB-cured adhesives have many advantages, like good heat, chemical and abrasion resistance, dimensional stability, adhesion to a large spectrum of substrates, their rapid cure at room temperature, that increases production rates, and the controllable depth of curing penetration, a major advantage over thermal curing. However, EB and UV equipment and maintenance are expensive.

Chemical/Polymer type

Even though the categorization of adhesives by hardening mechanism is more informative, a classification based on the chemical nature of the main polymer is often used, due to the fact that such grouping is simple and requires little explanation. It is achieved by ordering the materials by class of polymer first (**thermoset, thermoplastic** or **elastomeric**), sometimes with a separate class of natural polymers, and then subdividing these classes into more specific polymer families [1, 4, 23]:

– **Thermosets** - Thermosets are available in liquid, paste and solid forms, and are converted to low molecular weight liquids (if not already) in early application stages. They are subsequently cured by heat and/or chemical catalysts, turning them into three-dimensional structures with a very high molecular weight. They do not solvate easily, have low creep and survive fairly well in environments heated above their cure temperature. They have outstanding mechanical strength, and through modifications by addition of a second polymer they can be made to meet certain requirements, and so evidently, with this properties and versatility, they are a prime candidate to structural adhesive utilization. Epoxy adhesives are the most important of this bunch: as thermosetting, cross-linked resins, they are strong and brittle, but through a variety of procedures they can be made more flexible without loss of tensile strength, bonding a vast span of substrates, in diverse cure temperatures and humidity conditions;

– **Thermoplastics** - Thermoplastics are available in liquid and solid forms. Frequently blended with other polymers in order to obtain specific properties such as

tack or water resistance, their applications are wide-ranging, from packaging, book binding and shoe-making to remoistenable adhesives (stamps), specific structural metal bonds or even printed circuits. They and are capable of being softened and hardened repeatedly with heat cycles around a transition temperature, particular to each adhesive. Some widely used thermoplastic adhesives are Polyvinyl Acetate, Polyimides and aromatic Polyamides, and also include Polyesters and Cyanoacrylates;

- **Elastomers** - Elastomers are polymers with rubberlike properties, where the "elastic" portion of the word refers to the capacity to return to original dimension after a mechanical load is removed. Polymer cross-linking, referred to as vulcanization when applied to rubbers, was discovered in 1839 by Goodyear [24], and since then it has become one of the most important topics in polymer technology, since the degree of cross-linking achieved in the adhesive bond formation has a dominant effect on the stiffness of the polymer.An important elastomer adhesive is the Polyurethane, that can be used for applications requiring high toughness and resistance to tear and abrasion;

- **Natural Adhesives** - These are polymers obtained from natural resources in many forms: starch from carbohydrates, not recommended for humid service environments, protein based glues from animals, bitumen or resin, natural rubbers and gums

Function

The end purpose of a material has a great effect on its required properties, and fortunately, same polymeric material can be engineered to produce adhesive joints with different properties. Adhesives typically have high tensile and shear strengths, while **Sealants**, for example silicones, substitute this mechanical strength for impermeability, flexibility, the capability to resist vibration or bond materials with different coefficients of thermal expansion, in pursuit of optimal characteristics for filling gaps and cavities among two substrates. They can also be used to protect electronic components [1].

Primers have the job of protecting the substrates' surface and improving strength in structural bonding, and are fundamentally adhesives in solution, usually sprayed on the substrate. They protect the adherend from chemical reactions, attacks or contamination between the surface preparation procedures and the bonding phase, reason why the primer is usually based on the same chemicals as the adhesive that is being used [1].

Hot-Melt adhesives are one of the most versatile types, suitable for most substrate materials and commonly found in applications not requiring the transfer of high loads, namely industrial and assembly bonding such as packaging, book bonding, and general household usage. Their rapid setting leads to very high production speeds, and their

reduced space and maintenance requirements, in addition to indefinite shelf life, make them a very economic solution [1, 23].

There are some applications that require specific properties: **High-Temperature Adhesives** find their place in extreme environments, for instance advanced aircraft, satellites, missiles, space vehicles, where they withstand steady temperatures of 260°C, reaching much higher temperatures for short periods of time (760°C), and exposure to abrasive and degrading substances like aircraft fuels, hydraulic fluids, etc. They are quite expensive, require high cure temperatures and sometimes complicated cure schedules; as for **Conductive Adhesives**, usually based on epoxy polymers, have as the focal requirement electrical or thermal conductivity, which use metallic or metal coated fillers for this purpose, which end up raising the cost of this adhesive materials, as well as changing some of the mechanical properties [1, 4, 23].

Finally, **Structural Adhesives** - Their main task is to transfer loads, either static or dynamic, withstanding stresses that can be quite substantial between their adherends, providing stiffness for the whole structure. In practical terms, this means bond strengths of approximately 6-8 MPa, and lap shear strengths up to 70 MPa [23]. In the formation of structural adhesives, hardening is almost always achieved by chemical reaction which involves either Chain (addition) polymerization or Step (condensation) polymerization, and cross-linking in certain adhesives. Typically cross-linked thermosetting resins are used, even though some other types can be used, for example Acrylics, that are thermoplastics [23].

Structural adhesives find themselves in suitable applications such as components of aircraft fuselage, construction, machinery and the automotive industry, or even the aerospace industry, where their light weight, reliability and good fatigue resistance/vibration damping are of prime importance.

The principal disadvantage that prevents this technology from reaching further levels is the lack of an universal method to detect damage and predict long-term behaviour, and the pursuit of that is in itself the motivation for this whole project [1, 4, 23].

At this point, it needs to be said that it is not a trivial task to classify adhesives into perfectly separate categories, and that is because there are cross-category characteristics in display, clearly illustrated with some of the above listed examples: Epoxies, for instance, are mentioned as one of the principal candidates for structural purposes, as thermosetting, cross-linked resins; however, they also appear as a group of Elastomers, which have nitrile rubber added, and are widely used in films and tapes. The term “Hot-melt” can designate both a type of setting mode and a function.

Nevertheless, it is important to establish this classification framework in order to facilitate the selection of the the most suitable adhesive material to each design application.

2.3 Problems associated with adhesive joints

Even if a certain adhesive is the perfect choice for a given application, its durability is a function of the entire bonding system, that is, adhesion defects and other problems can still arise and lead the joint to failures of different types when subjected to mechanical stress, be it because of the lack of proper surface preparation, absence of a mandatory primer or other environmental disturbances, such as high humidity.

Structural bonds, in particular, are expected to undergo some (possibly various) form(s) of loading for a significant part of their service life, and so an understanding of the bond failure and failure modes is critically important. It can occur in a number of places within a bonded joint or structure, as illustrated in Figure 4 [1, 23]:

- Cohesively within the adherends;
- Interfacially between the adherend and an altered layer resulting from any pre-treatment;
- Cohesively within an altered layer resulting from pre-treatment;
- Interfacially between pre-treatment and a primer, if present;
- Cohesively within a primer;
- Interfacially between a primer and the adhesive;
- Cohesively within the adhesive.

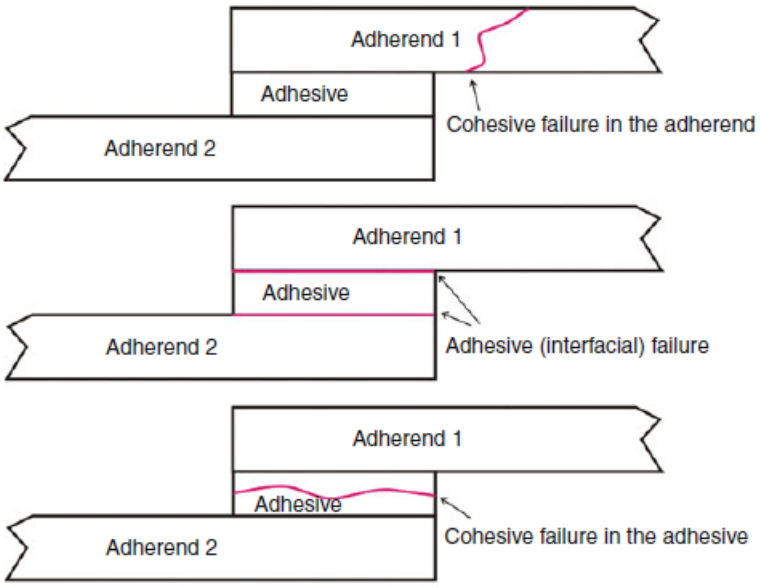


Figure 4: Types of failure within a bonded joint. Adapted from [1].

Mixed failure modes resulting from a combination of the failures mentioned above are commonly observed. Clearly, the first option should not occur with thick metal adherends, or well designed structural parts, but this failure has nothing to do with the adhesive joint itself. The other options occur when any of the adhesive layers is compromised cohesively, or suffer interfacial disconnections from one another [23].

In the case of structural joints, cohesive failure of the adhesive is often observed. As it has been mentioned throughout this chapter, structural adhesives are cured in order to develop cross-linking among the polymers during the assembly of the joint, joining the groups of atoms into large, three-dimensional structures, as shown in Figure 5 c), accounting for a higher strength. If this hardening process fails or is somehow incomplete or inadequate, zones with insufficient even voided of adhesion will appear in the joint.

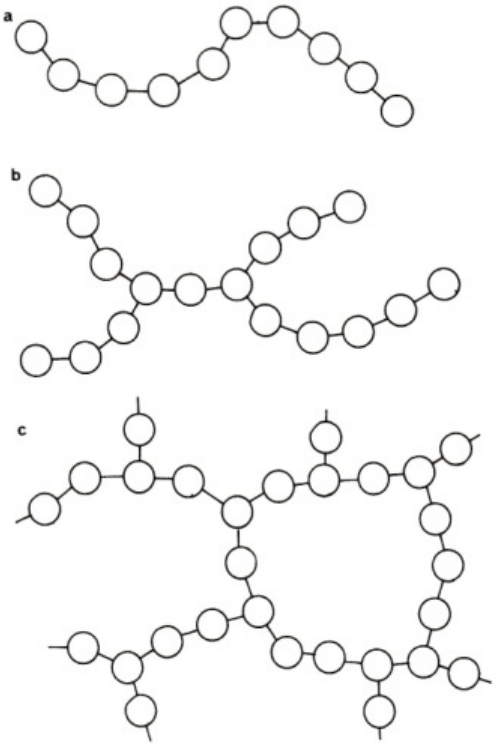


Figure 5: Polymer structures: (**a**) Linear, (**b**) Branched, (**c**) Cross-linked. Adapted from [4].

Internal defects, referred to as "bad/poor adhesion", are very hard to assess, these voids are not visible from the outside, and so advanced testing mechanisms should be put in place to ensure that the adhesive structure is sound and reliable, adhesive joints usually fail by the initiation and propagation of flaws, and if the flaw occurs next to a void, there is a high stress concentration that will certainly lead to failure [1, 23].

There is standardized testing for adhesives and surface treatments, sanctioned by organizations such as ISO, BSI and ASTM, put in place to understand the role that

stress plays in the mode of failure, in addition to other tests concerning the material properties (density, viscosity, etc.) or even curing/setting behaviour [23]. Conventional failure strength tests are carried out in different joint specimens that are destroyed in order to determine mechanical properties of the tested materials: tension and compression testing is useful to determine the Young's modulus and tensile stress-strain curve, one of the staring points of any mechanical design project [1].

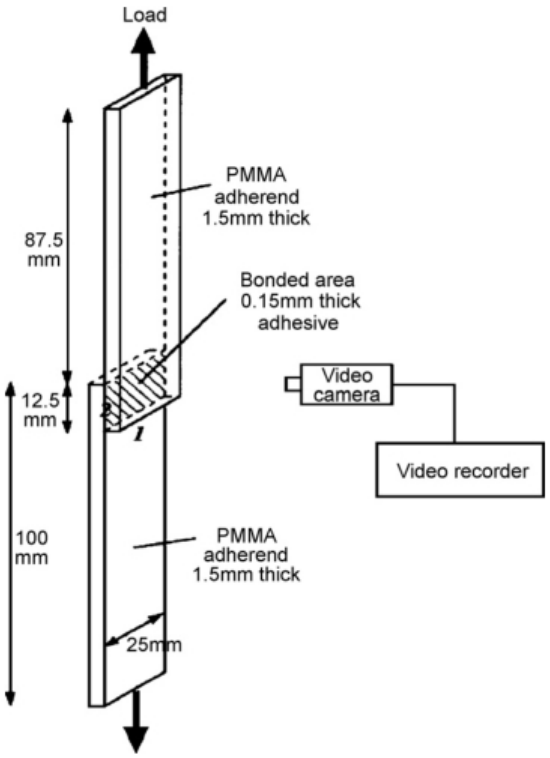


Figure 6: Schematic of tensile lap shear test. [5].

Shear testing is evidently one of the most common in adhesives, and there are many varieties, for instance lap shear tests, represented on Figure 6, thick adherend shear tests, pull out tests, and also torsion tests, which are very accurate because the stress concentrations are lower, but is not that common in laboratories [1]. Peel testing is also an interesting class of fracture testing, and there is a multitude of tests designed to determine the peel strength, very dependant on temperature and pulling rate, and even more specialized tests, like tack tests or impact tests, executed through special instruments, such as the Split Hopkinson Pressure Bar, that help to learn how adhesives react when exposed to loads abruptly [1, 5, 23].

Yet, even though all of these are relevant to gain knowledge about the mechanism of adhesion, they all sacrifice the specimens and give us no assurance that any given structural adhesive bond put in place will be complete and ready to operate at full capacity without failing, for its whole service life. For that, the solution is a NDT approach.

2.4 Non-Destructive Testing

Regrettably, there is still no way to determine the overall shear strength of structural adhesive bonds through NDT, however, techniques that depend upon mechanical, physical or chemical parameters, like the bond area, interfacial stiffness, elastic modulus or the joint's response to some specific impulses, which correlate directly to the strength of the joint, can indicate "isolated phenomena", referring to voids, cracks, porosity, second phase material and so on, which will cause unwanted stress concentrations adversely affecting the short and long-term strength of the joint, both cohesively on the polymeric adhesive or adhesive failure at the interface [1,4,23]. Figure 7 illustrates some of the most common adhesive bond defects.

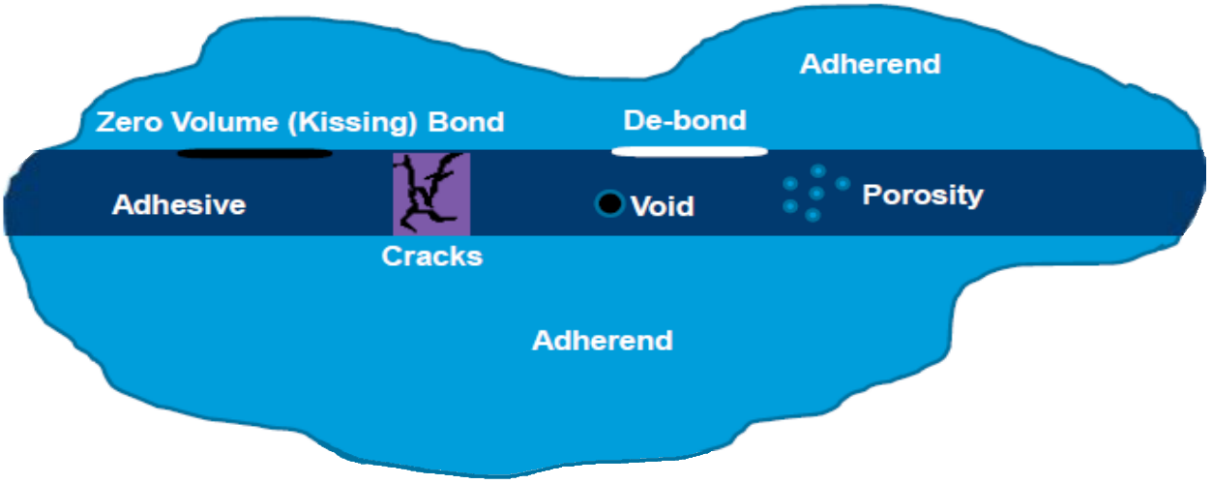


Figure 7: Adhesive bond-line possible defects. From [6].

As to whether any of these defects are critical, it depends on their extent, position, and the nature of the applied stress, but it is definitely important to notice them as early as possible, in order to avoid severe damage.

There are two main regimes in which NDT is carried out: during manufacturing, mainly related to surface analysis and quality control, and in-service use - SHM.

Before bonding, surface inspection is advised to properly prepare the adherend surface and enhance its adhesive properties, starting by removing excess amounts of water vapour, hydrocarbons or other contaminants. The wettability is tested, by evaluating the behaviour of a water droplet: if the surface is clean, then the water will spread over a large area, otherwise, if contaminated, the water will remain as droplets [4]. There are a number of advanced surface characterization and analysis techniques, employed to investigate the surface quality, namely the roughness, polarity, chemical composition and surface free energy: [4,5]

- Time-of-Flight secondary ion mass spectrometry (ToF-SIMS);
- X-ray photoelectron spectroscopy (XPS);

- Atomic force microscopy (AFM);
- Scanning electron microscopy (SEM);
- Optical contact angle analysis;
- Attenuated total reflectance infrared spectroscopy (ATR-IR);
- Fokker contamination tester.

Nevertheless, the vast majority of NDT techniques associated with adhesive bonds takes place after the joint has been made, either right after manufacturing or during service, and they are mainly trying to identify voids or any of the aforementioned bond-line defects [4].

The particular thing about SHM is that usually there is no need for any external operator, as sensors/transducers are permanently attached to the examined structure, collecting data continuously during the structure's working hours (passive SHM), and also being available to be plugged into proper instrumentation and carry out specific tests, where the structure is actuated with some kind of disturbance or agitation and the system response is monitored through the sensors (active SHM). It is of paramount importance that the sensing devices are of high quality and reliability, great accuracy and sensitivity, as they play a vital role in these procedures [25].

Apart from this network of sensors, on-board data and computing facilities are required to implement these techniques, along with algorithms that make use of that data, comparing it to stored data from the pristine structure to calculate a damage index, informing about existence, localization and type [26].

The subject of SHM is very broad, and it can include applications with different levels of detail in the systems. According to Cawley, SHM techniques can be classified based on purpose [27]:

- **Machine condition monitoring** - Not strictly concerned with structural health, it uses passive SHM measurements and is routinely applied in the industry for reliability's sake, typically when it comes to rotating machinery, with many standards available;
- **Global monitoring of large structures** - In practice, Structural identification, the development of numerical models for large structures, such as bridges, to assure their sustained integrity, also using passive SHM;
- **Large area damage monitoring** - A full volume coverage of a region of a large structure, searching for broad damage using active SHM methods through a limited number of sensors. There is often a trade off between sensor area coverage and sensitivity, so full coverage of a large structure would require multiple of these systems to be deployed;

– **Localised damage detection** - Focused on small areas of the structure or known damage hot-spots, scanning for cracks and corrosion with active SHM techniques is a growing solution for specialized commercial applications, specially when it comes to adhesive bonded structures;

There are a lot of NDT tests and techniques, but the majority of them can be classified under the following types [4, 28]:

– Visual inspection;

– Ultrasonic testing;

– Magnetic particle testing;

– Eddy currents testing;

– Thermography / Thermal methods;

– Penetration testing;

– Radiography / X-Ray;

– Optical holography;

– Acoustic emission testing;

– Guided waves methods.

A lot can be said about all of these types of techniques: their properties and physical phenomena involved, their strong points and most advantageous and typical applications and so on; however, in the context of this project, the focus is Guided Waves, specifically LW, which will be discussed next. It should be noted that no single method of detecting internal structural defects is universally applicable. The type of testing environment (i.e post-manufacture or SHM), paired with the properties of the structure/adhesive bond to be tested and the size of the defect that is being sought after should determine what technique to apply [29].

2.5 Lamb Waves

It was 1916 when Horace Lamb, F.R.S. published the article *"On Waves in an Elastic Plate"*, in which he presents his considerations on the problem of two-dimensional waves in a solid bounded by parallel planes, first approached by Lord Rayleigh, F.R.S in 1889 [30]. In fact, both Rayleigh and Lamb mention each other on different papers on the subject of vibrations around this time [31], but in this paper in 1916, Lamb presents the equations that characterize the propagation of this type of waves, that ended up with his name.

LW techniques have proven capabilities to provide information about damage type, severity and location ever since they were first used to detect damage in 1960 by Worlton of the General Eletric Company. Since then, they have been employed in a variety of fashions: from research conducted at NASA that demonstrated the possibility of using LW to detect delamination in composite beams, to different groups at Imperial College, working to optimize the generation of directional LW, among many others [32].

Assuming an infinite solid medium, elastic waves can propagate in two basic modes: pressure (P) waves and shear (S) waves. Yet, if the medium is bounded, wave reflections occur at the boundary, giving way for more complicated wave patterns. Guided Waves are particularly interesting because they remain contained in a wave guide and can travel very large distances with little amplitude attenuation, and are excellent for damage detection due to the full cross-section interrogation of the material, assuring us that the wave will interact with any possible defect. The sensitivity to different defects will however depend on mode type and the location of the defect in the thickness of the structure. Examples of Guided Waves are of course LW, and others, such as Love Waves, traveling in layered materials, or Rayleigh Waves, constraint to the surface [33, 34]. These are actually the typical seismic waves that propagate on the surface of the earth during an earthquake.

LW in specific are a form of elastic perturbation that propagates in a solid thin plate with parallel free boundaries, but can also occur on shell-like structures with shallow curvature, and they are made up of a superposition of longitudinal and shear modes, whose characteristics vary with entry angle, excitation and structural geometry [35]. They have two fundamental propagation modes: Symmetric (Sn) and Anti-symmetric (An), as shown in Figure 8.

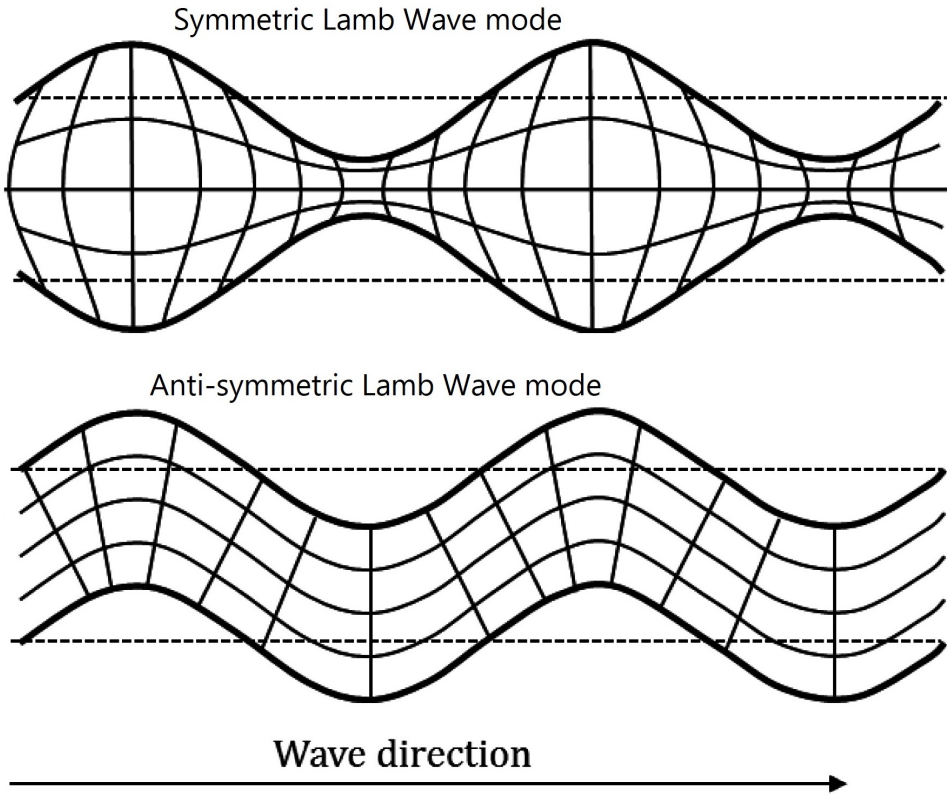


Figure 8: Symmetric and Anti-symmetric LW modes. From [7].

This symmetry or anti-symmetry happens with respect to the plate's mid-plane. Considering a plate with stress-free upper and lower surface, the outline of the equations for a LW propagation, following Giurgiutiu, et al. and Su, et al., is presented [33, 35, 36]. It can start by the equation of motion for an isotropic elastic medium, that describes the displacement field by satisfying Navier's displacement equation:

$$\mu\nabla^2\mathbf{u} + (\lambda+\mu)\nabla\nabla \bullet \mathbf{u} = \rho\frac{\partial^2\mathbf{u}}{\partial t^2} \tag{1}$$

where λ and μ are the Lamé constants, that is two material-dependent quantities that arise from the study of elastic stress-strain relationships, ρ is the mass density, and $\mathbf{u}$ is the displacement vector, given by:

$$\mathbf{u} = \nabla\Phi + \nabla\times\Psi \tag{2}$$

where Φ and Ψ are the potential functions.

The wave equations can be written as a function of this potential functions, the mass density, the Lamé constants, and the wavespeeds, both the pressure (L) wavespeed, given

by $c_L^2 = (\lambda + 2\mu)/\rho$, and the shear (T) wavespeed, given by $c_T^2 = \mu/\rho$:

$$\begin{aligned} \frac{\partial^2\Phi}{\partial x^2} + \frac{\partial^2\Phi}{\partial y^2} + \frac{\omega^2}{c_L^2}\Phi = 0 \\ \frac{\partial^2\Psi}{\partial x^2} + \frac{\partial^2\Psi}{\partial y^2} + \frac{\omega^2}{c_T^2}\Psi = 0 \end{aligned} \tag{3}$$

The time dependence for these waves is assumed harmonic, in the form $e^{i\omega t}$, bringing the general solution to Equation (3) as:

$$\begin{aligned} \Phi &= (A_1 \sin py + A_2 \cos py)e^{i(\xi x - \omega t)} \\ \Psi &= (B_1 \sin qy + B_2 \cos qy)e^{i(\xi x - \omega t)} \end{aligned} \tag{4}$$

where $\xi = \omega/c$ is the wavenumber and:

$$p^2 = \frac{\omega^2}{c_L^2} - \xi^2 \quad , \quad q^2 = \frac{\omega^2}{c_T^2} - \xi^2 \tag{5}$$

The four integration constants, A_1, A_2, B_1, B_2 are to be obtained from the boundary conditions. Getting the relations between the potential functions and the displacements, stresses and strains:

$$\begin{aligned} u_x &= \frac{\partial\Phi}{\partial x} + \frac{\partial\Psi}{\partial y} \quad , \quad \tau_{yx} = \mu(2\frac{\partial^2\Phi}{\partial x\partial y} - \frac{\partial^2\Psi}{\partial x^2} + \frac{\partial^2\Psi}{\partial y^2}) \\ u_y &= \frac{\partial\Phi}{\partial y} + \frac{\partial\Psi}{\partial x} \quad , \quad \tau_{yy} = \lambda(\frac{\partial^2\Phi}{\partial x^2} + \frac{\partial^2\Phi}{\partial y^2}) + 2\mu(\frac{\partial^2\Phi}{\partial x^2} - \frac{\partial^2\Psi}{\partial x\partial y}) \\ \epsilon_x &= \frac{\partial u_x}{\partial x} \end{aligned} \tag{6}$$

and plugging them into the general solution equations gets:

$$\begin{aligned} u_x &= [(A_2 i\xi \cos py + B_1 q \cos qy) + (A_1 i\xi \sin py - B_2 q \sin qy)]e^{i(\xi x - \omega t)} \\ u_y &= [-(A_2 p \sin py + B_1 i\xi \sin qy) + (A_1 p \cos py + B_2 i\xi \cos qy)]e^{i(\xi x - \omega t)} \end{aligned} \tag{7}$$

For free wave motion, the homogeneous solution is derived by applying the stress-free boundary conditions at the upper and lower surfaces ($y = \pm d$), where d is half of the plate thickness, obtaining the characteristic equations:

$$D_S = (\xi^2 - q^2)^2 \cos pd \sin qd + 4\xi^2 pq \sin pd \cos qd = 0 \quad \text{(symmetric motion)} \tag{8}$$

$$D_A = (\xi^2 - q^2)^2 \sin pd \cos qd + 4\xi^2 pq \cos pd \sin qd = 0 \quad \text{(anti-symmetric motion)} \tag{9}$$

And finally, Equations (8) and (9) can be rewritten in the more compact form as the Rayleigh-Lamb equation:

$$\frac{\tan pd}{\tan qd} = -\left[\frac{4\xi^2 pq}{(\xi^2 - q^2)^2}\right]^{\pm 1} \tag{10}$$

where the exponent +1 corresponds to symmetric (S) motion and -1 to anti-symmetric (A) motion. Equations (8) and (9) accept a number of eigenvalues, $\xi_0^S, \xi_1^S, \xi_2^S, ...$, and $\xi_0^A, \xi_1^A, \xi_2^A, ...$, respectively. To each of those corresponds a set of eigencoefficients (A_2, B_1) for the symmetric case and (A_1, B_2) for the anti-symmetric one, that can be plugged into Equation (7) and yield the corresponding modes: $S_0, S_1, S_2, ..., S_n$ and $A_0, A_1, A_2, ..., A_n$.

The coefficients p and q in Equations (8) and (9) are dependant on the angular frequency ω, consequently the eigenvalues ξ_i^S and ξ_i^A will change accordingly, and since the wavespeeds correspond to $c_i = \omega/\xi_i$, they will also change with frequency, and this change produces the so-called wave dispersion.

LW are highly dispersive, meaning that the fundamental way to concretely describe their propagation in a material is through their dispersion curves, that plot the phase and group velocities against the excitation frequency (often shown as a product with thickness), since for each frequency-thickness product, and each solution of the Rayleigh-Lamb equation, one finds a corresponding LW mode [32, 36]. Figure 9 presents the dispersion curves for each of the first modes.

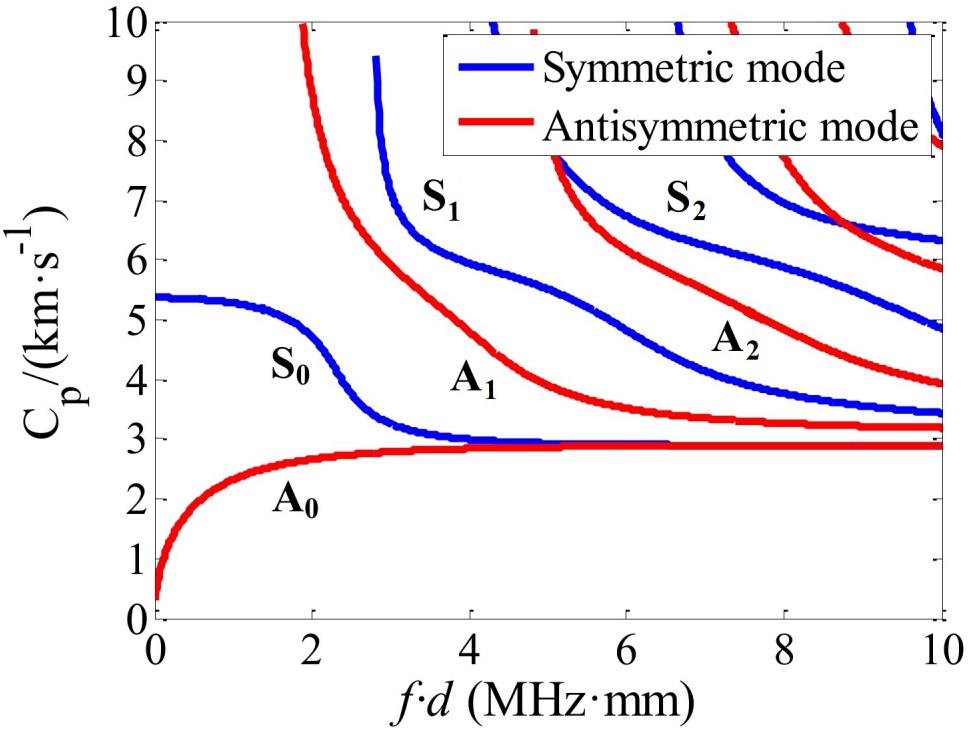


Figure 9: Wave speed dispersion curves in plates: Symmetric LW (S_n); Anti-Symmetric LW (A_n). From [8].

Where the C_p is the shear wave speed. A proper LW mode for damage detection

should feature non-dispersion, low attenuation, high sensitivity, easy excitability and good detectability. The best way to prevent wave dispersal is to have an input signal with a narrow bandwidth, such as a windowed toneburst for instance, making it a more frequently adopted input signal, rather than a pulse.

The generation of LW can be done through a variety of instruments, roughly grouped under five categories [35]:

- Ultrasonic probes;
- Laser;
- Interdigital transducers;
- Optical fibre;
- Piezoeletric elements.

All of which have, of course, strengths and weaknesses. Piezoeletric, or lead zirconate titanate (PZT) elements have advantages, since they can be used for both LW generation and acquisition, delivering excellent performance, allied to their neglectable mass/volume, effortless integration, outstanding mechanical strength, wide frequency response range, low power consumption and acoustic impedance, and low cost, making them particularly suitable for SHM applications as an in-situ generator/sensor. On the other hand, some nonlinear behaviour and hysteresis under large strains/voltages, or at high temperatures should be accounted for, and their brittleness and low fatigue life may cause concerns or limit some applications. Importantly, PZT-generated LW unavoidably excite multiple modes that generate complex response signals, requiring sophisticated signal processing to successfully utilize them to detect and classify damage [35].

3 Lamb Waves Signal Processing

3.1 Acquisition of simulation data

LW based NDT methods can detect incipient damage, often unnoticed by other techniques, with the help of suitable electric signals applied to a PZT actuator that induces that type of waves, translating into small mechanical vibrations to the structure subject to testing, or specimen, in laboratorial context. Other PZT sensors, placed strategically in the specimen, measure the vibrations and output electric signals, with distinct amplitude and phase from the input, whose characteristics will depend on the existence and type of damage in the structure.

As it was explained, the nature of LW dictates that these signals are complex, and extracting meaningful information about the structures health necessitates advanced algorithms to process the data, so naturally ML comes to mind: a rapidly emergent and developing field of Data Science (DS) that has recently been applied to many different fields involving data processing, that is intimately related to statistics, while making use of nowadays computational power to analyse data and make predictions, which is exactly what is needed. The tool used to develop these algorithms was `Python`, a general-purpose and open-source programming language, with thousands of custom libraries and modules made by the community, widespread documentation on the internet and very suitable for DS endeavors. The IDE - integrated development environment - used was `Spyder`, present on the Anaconda platform.

Be that as it may, data processing algorithms based on ML require large volumes of data to be trained, therefore it would be impracticable to use experimental data in their development. The alternative is to use simulation data from numerical models, generated with finite element method (FEM) software - `ABAQUS`. For the purpose of this project, two simulated specimens were modeled:

– Aluminium plate with 1 PZT actuator and 3 PZT sensors placed forming right angles among each other, that can either be perfectly regular (no damage) or contain one hole with size 2 mm, 6 mm or 10 mm, placed in a random spot in the plate and marked with designated coordinates (x,y). Both the hole size and its position vary with each test, if present. In Figure 10 the model of the aluminium plate used in the simulations is presented, where the red squares represent the PZT sensors, and the orange square is the PZT actuator.

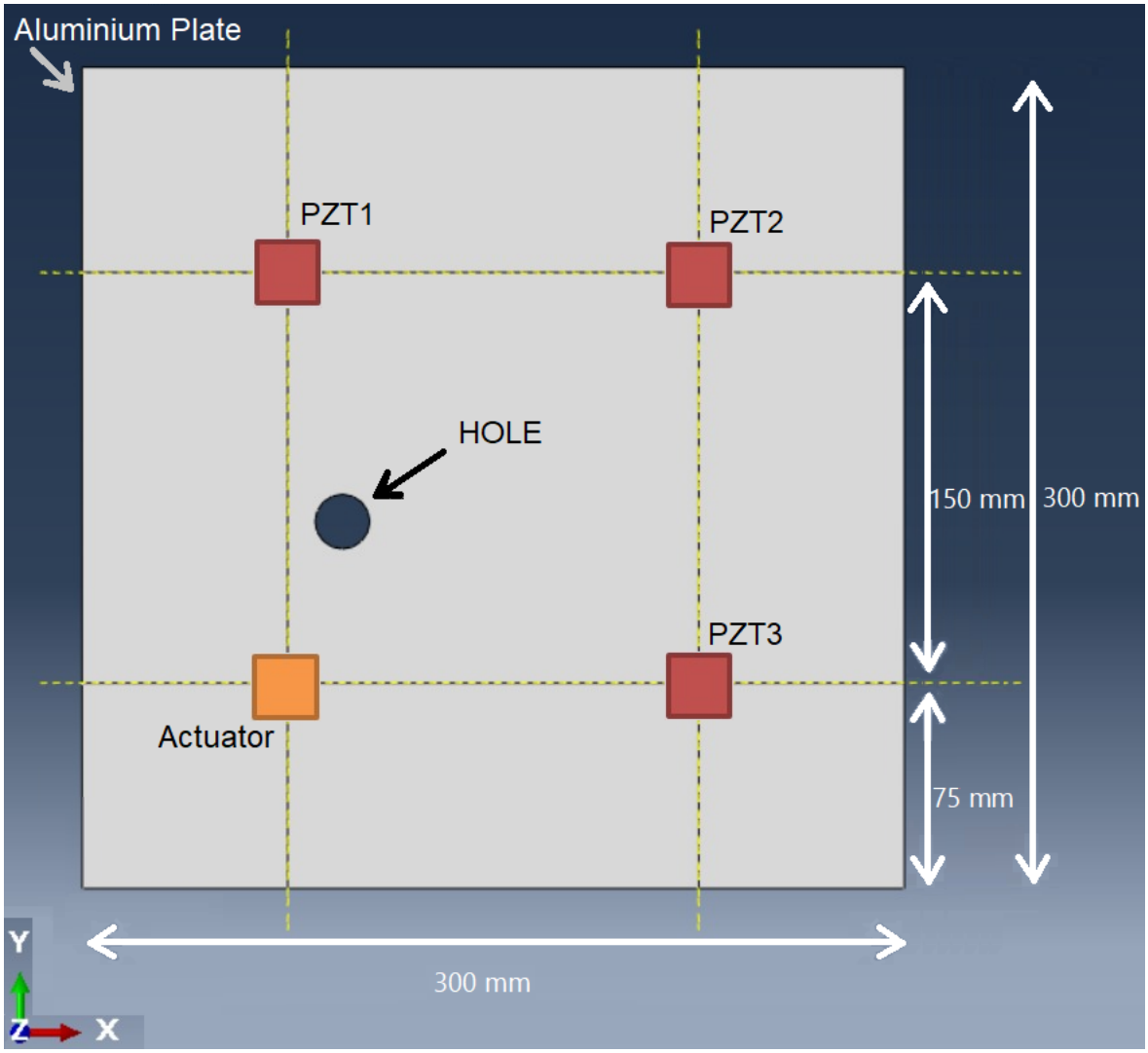


Figure 10: Model of the aluminium plate simulation setup.

From this simulation, data from 600 tests were used, totaling 600×3 sensors $= 1800$ time series;

– An adhesive joint among two 150 x 150 x 2 mm aluminium plates, with a single-lap joint design with 25 mm overlap with varying degrees of adhesion strength on each test (ranging from 600 to 270000 kPa), with 1 PZT actuator on the top plate and 1 PZT sensor on the lower plate, both centered on the plates and placed 30 mm from the edge. The aluminium plate has a density $\rho = 2500$ kg/m^3, Poisson ratio $\nu = 0.33$ and Young modulus $E = 72.4$ GPa. The chosen adhesive to simulate was a 0.2 mm layer of Nagase T-836/R-810, as it has great potential for industrial applications, specifically aeronautical and automotive. The simulated adhesive joint is presented on Figure 11.

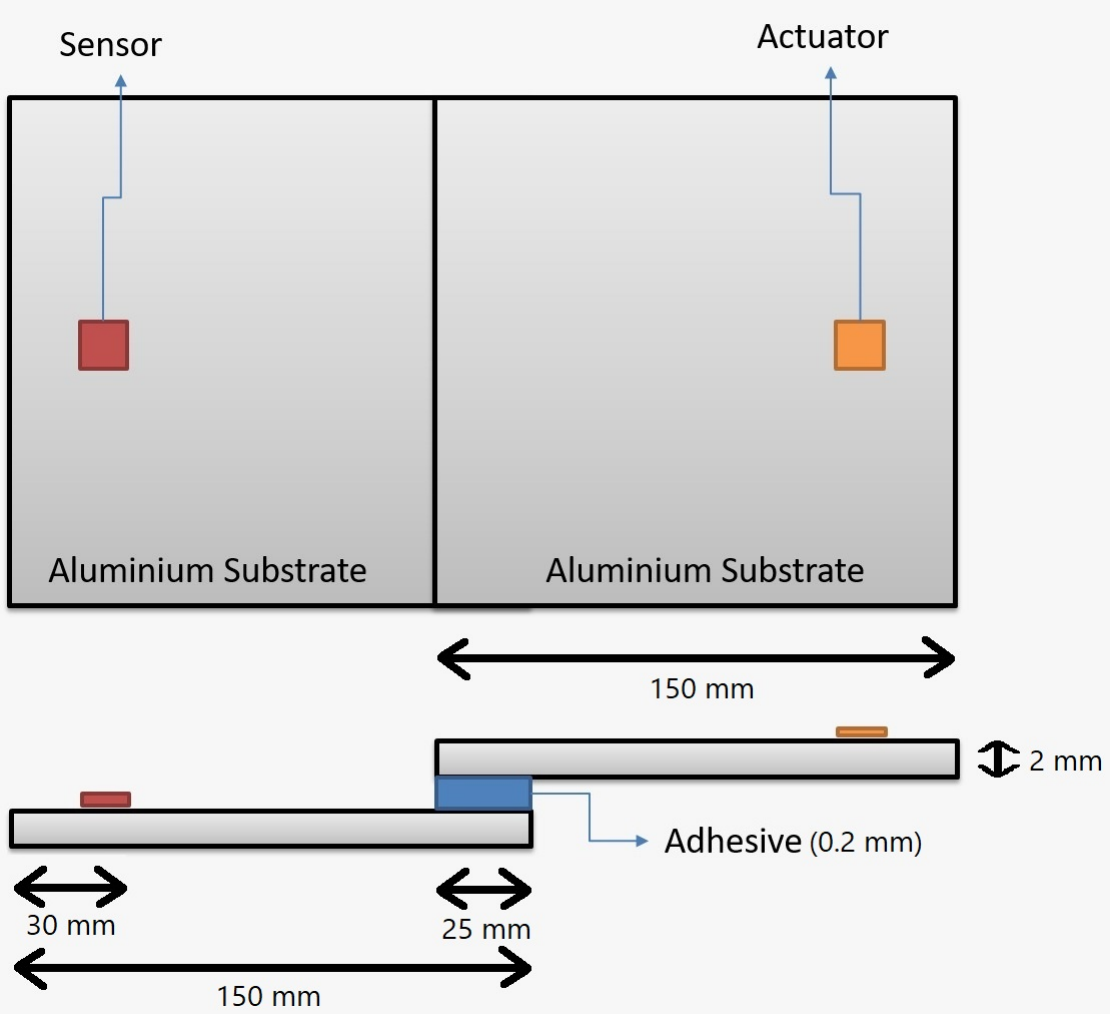


Figure 11: Model of the adhesive joint simulation setup.

This simulation was run 900 times, and since it measures only one sensor's displacement, accounts for 900 time series.

The reason for having these 2 simulations was the fact that the first simulation was already implemented and had results in the beginning of this project (although the results were improved thanks to corrections to some parameters), and the base algorithms were developed with those results. Then, they were adapted and applied to the adhesive simulation data. Both simulations ran with a fixed time of 0.5 ms, as this is enough time to have the waves interact with the defects / adhesive joint and arrive at the sensors without receiving too many reflected waves from the opposite wall. The excitation signal chosen was a 5-cycle Hanning-windowed sinusoidal tone burst with the central frequency of 100 kHz, applied to the PZT elements, as it is narrow in terms of bandwidth, below the frequencies in which multiple modes appear, but still large enough to detect damage.

The simulations themselves were not part of this project, only their results were accounted for, as the starting point of the damage detection process. They were packaged in a Comma Separated Value (.csv) file, containing the information for each test in rows,

that begin with a few columns of the test specifications (hole size/location or adhesion strength), followed by the values for the PZT sensor signal, each column with one value for each time step.

As it would be expected, the first procedure was to import those signals into the programming environment and represent them in a graph, in order to easily visualize them. Evidently, the number of examples shown will be kept to a minimum, not to overcrowd the report. From this point forward, all the results presented are directly extracted from the developed `Python` code, which will be presented in its entirety as an Appendix.

Figure 12: Representation of a random aluminium plate test, with a signal from each of the 3 PZT sensors. Hole size = 2 mm; Hole position (x,y) = (194,209) mm.

From just looking at Figure 12, the chaotic nature of these signals is rapidly assessed. They are certainly different from each other, due to the sensor's position on the plate, the interaction of the LW with the defect and so on, but definitely nothing can be concluded from observing their format. Nevertheless, this is just a representation of the signal, the real problem now is how to handle these large amounts of information.

3.2 Concept of *features*

Even with the powerful tools that ML has to offer, it is difficult for ML algorithms to produce proper results if the input is rough data that is not preprocessed. In this case, there could exist outliers (tests that did not go as planned and/or yielded strange results) that need to be discarded, missing values on tests that should be eliminated or appended with accordingly interpolated values [37]. It is also important to make the tests uniform in terms of array size, so that all signals are in a vector of equal dimension. That is achieved

by interpolating the whole original array of values, imposing a fixed time step to fit an array of predetermined size - 5000 steps in this case to be exact, meaning 5000 values at constant time intervals. This preprocessing is fundamental in every DS project, even more so if the data is obtained from real life sensors, that might contain noise and interference effects that should be minimized, or malfunctions that should be accounted for. Such is not the case in the present project, as all the data is obtained from simulations, so the preprocessing stage has a lighter overall significance.

Still, after the first cleaning of the data, there is still a tremendous amount of it to deal with. Despite only 2 dimensions existing at the core of the problem (displacement as a function of time), a large number of observations for that dependable variable are registered, turning the whole data set into a high-dimensional one. It is important to reduce this dimensionality in order to handle the data adequately, and use it successfully in ML algorithms. Ideally, the reduced representation should have a dimensionality that corresponds to the intrinsic dimensionality of the data, the minimum number of parameters needed to account for the observed properties [38].

But notice, one needs to differentiate between the number of cases (observations) in a large data set, and the number of variables available for each case - these variables comprise specific information about the data and are referred to as *"features"*. There are none of these in the original data set, only raw test data in the form of time series, so putting in place a mechanism that can analyse this raw data and extract meaningful features, to be used later in ML algorithms, was a priority in this project. This procedure can be viewed as "nontrivial extraction of implicit, previously unknown and potentially useful information from data, or the search for relationships and global patterns that exist in databases" [39].

Feature Engineering plays a vital role in ML algorithms and big data analytics. Indeed, little can be achieved if there is a short amount of features to represent the underlying data objects, and the quality of the results obtained in those algorithms will reflect the quality of the available features themselves. It encompasses the generation, extraction, transformation, selection, analysis and evaluation of features - attributes of data that are relevant to a ML process [40, 41].

Feature Engineering is often data specific and application dependent, which means that different data types - text, images, streaming data, social media data - require specialized techniques [40]. This project is based on time series that are not in any way correlated with date or time, like many literature's time series examples are - price evolution of some product/stock, temperature variation in a sensor over a time period, etc. - there is no need to recognize features in typical categories such as Date-Related, Lag Features, or Rolling Window Features, since these are mainly used to predict the next values of a given time series, accounting for trend and seasonality for example, which is a common motivation for time series analysis (weather forecasting, econometrics, etc.) [42, 43].

In this case, the sensor signals are time series that stand alone in time, and the

information extracted from them is destined to predict not their next values, but an intrinsic property - damage - that can influence the signals (expanded upon later on).

3.3 Manual feature extraction

In the early phases of this project there was an interest in discovering these features, their types and how to categorize them appropriately in the context of the project, how to visualize them, represent them, and to have a greater grasp of what they mean. The way to obtain them manually is to apply, for the most part, mathematical and statistical operations to the signal data.

It is useful to distinguish three main types of features, besides simple **Statistical Features** (which can also be significant), based on the domain their information is in, namely **Time Domain**, **Frequency Domain** and **Time-Frequency Domain**:

Statistical features

These are the simplest, obtained directly from the signal by statistical analysis, they represent basic information about the data. Some examples are the Mean, Variance, Standard Deviation, Skewness, Kurtosis and higher order moments, and even Maximum and Minimum values (Peaks), among many others. Even though they do not result from complex analysis and advanced methods, their information can be really valuable for ML algorithms [44].

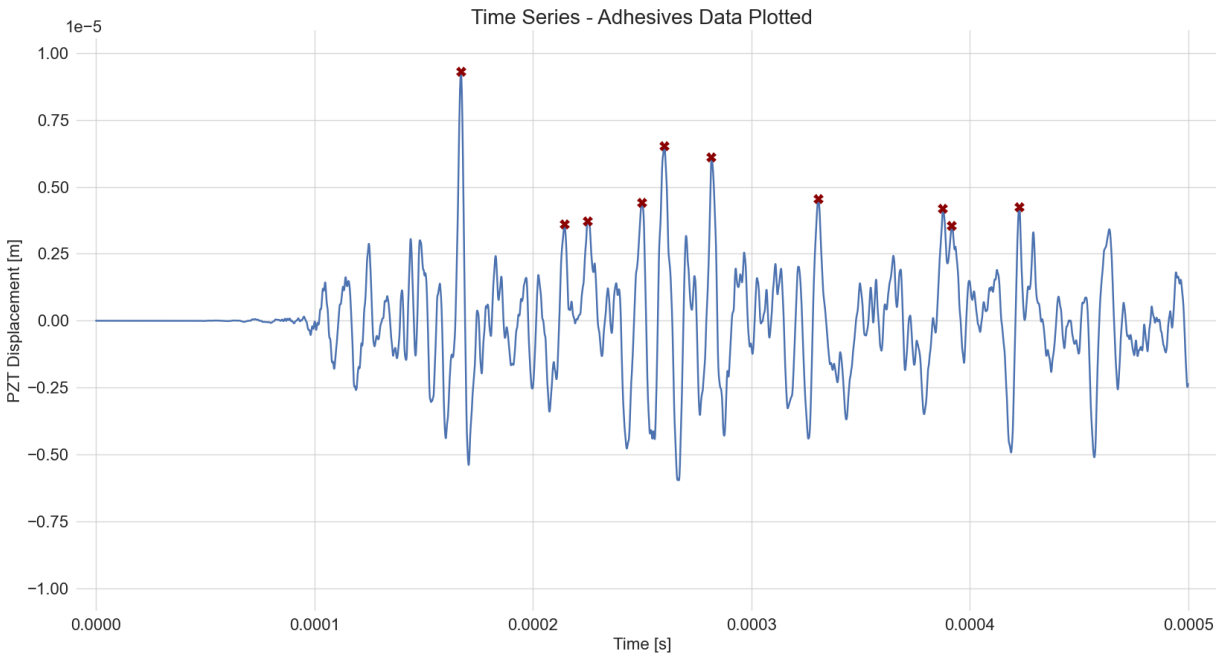


Figure 13: Representation of an adhesive joint test, with a signal from the PZT Sensor and the top 10 peaks marked. Adhesion strength = 900 kPa.

For example in Figure 13 only the top 10 peaks were marked directly on the time series

plot, but the values of all the local minima and maxima of the entire domain is saved in a variable, for each of the tests - a statistical feature.

To have a comparison, Figure 14 is the signal from a test with a higher adhesion strength in the joint.

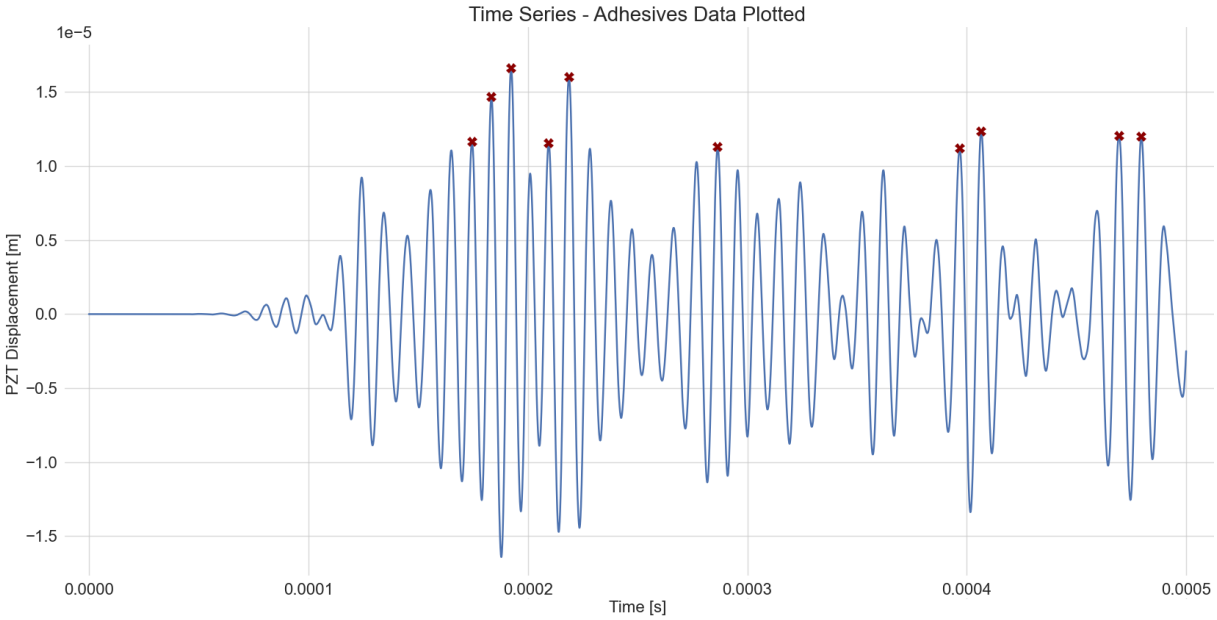


Figure 14: Representation of an adhesive joint test, with a signal from the PZT Sensor and the top 10 peaks marked. Adhesion strength = 50100 kPa.

Significantly different response between the two signals, it is immediately perceptible that the second signal comes from a stronger, firmer and more consistent medium, as the LW propagates and arrives at the sensor more smoothly. The average displacement measured on the sensor is higher, and all of the 10 peaks appear above the 1×10^{-5} m mark, while on the first test, the single highest peak falls short of that mark, being the only one that surpasses 0.75×10^{-5} m.

Time domain features

Even though the objective is not to predict future values for the signals using typical time based models, like the Autoregressive (AR), Integrated (I) and Moving Average (MA) models, that can be fused together in the ARIMA model first introduced by Box and Jenkins [45], some orders/parameters of these models can be used as a feature in itself, as well as the Auto-Correlation dimensions of the signal (where it is compared to itself with a small delay). The Auto-Correlation coefficients produced a by the vibration of a healthy structure could be different than those from a faulty one. From the ML algorithms viewpoint, these features are just arrays of organized information to be used in a specific task, with no connection to the time-based models that originated them, or their purpose.

Frequency domain features

Frequency domain analysis is arguably the most far-reaching set of mathematical tools utilized in engineering, specially signal analysis, and its cornerstone is the Fourier Transform (FT). It is based on the premise that every function - no matter how complex it looks - can be decomposed into a sum of simpler functions, a concept proposed by Joseph Fourier in 1822 in his publication *The Analytical Theory of Heat*, that far transcended the particular subject of heat conduction [46, 47].

Taking $f(t)$ as a time-dependant input signal, that may be composed by harmonic and/or periodic elements, its Fourier Transform $F(\omega)$ is called the signal's *spectrum*, and can be viewed as the frequency response - a transformation of the time signal into a sum of basis functions (sinusoidal) of various frequencies, which the original signal contains as periodic components, where [46, 48, 49]:

$$F(\omega) = \int_{t=-\infty}^{\infty} f(t) e^{-j\omega t}\, dt \tag{11}$$

with $j = \sqrt{-1}$ and the complex exponentials as the sinusoids: $e^{j\theta} = \cos\theta + j\sin\theta$. A Fourier Transform pair is often written $f(t) \leftrightarrow F(\omega)$. The Inverse Fourier Transform is also applicable, changing the content from the frequency to the time domain:

$$f(t) = \frac{1}{2\pi} \int_{\omega=-\infty}^{\infty} F(\omega) e^{j\omega t}\, d\omega \tag{12}$$

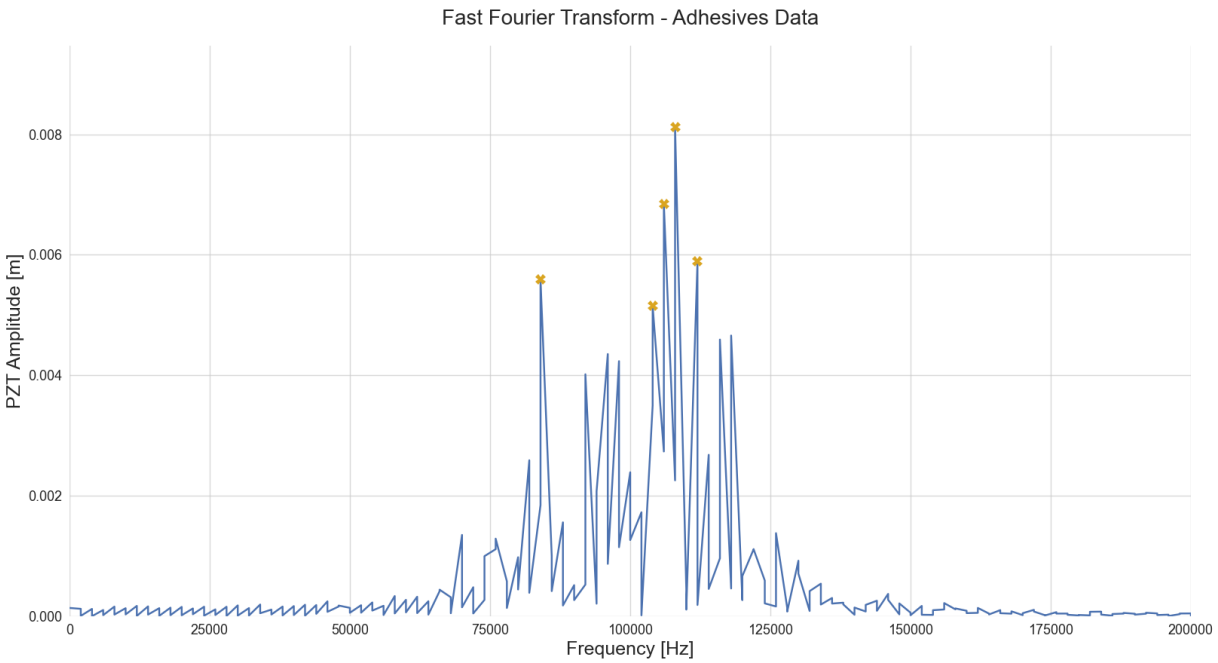
$F(\omega)$ written as a complex number in terms of its magnitude and phase, ultimately tells how much content the original signal has at any frequency ω. The integral operator on the equations automatically conveys the notion of a continuous input function. However, as is the case with most of the real-life signals, the input functions tend to be discrete, by nature of the sampling process that occurs on sensors and transducers, in this instance simulated. This results in a discrete signal with finite duration, presented as an array of values sampled at a designated frequency f_s, and so the Discrete Fourier Transform (DFT) is the indicated tool to obtain its spectrum X_k.

Supposing the signal is x_n of length N, for $n = 0...N-1$, the spectrum X_k, which ends up being a sequence of N complex numbers, comes [49]:

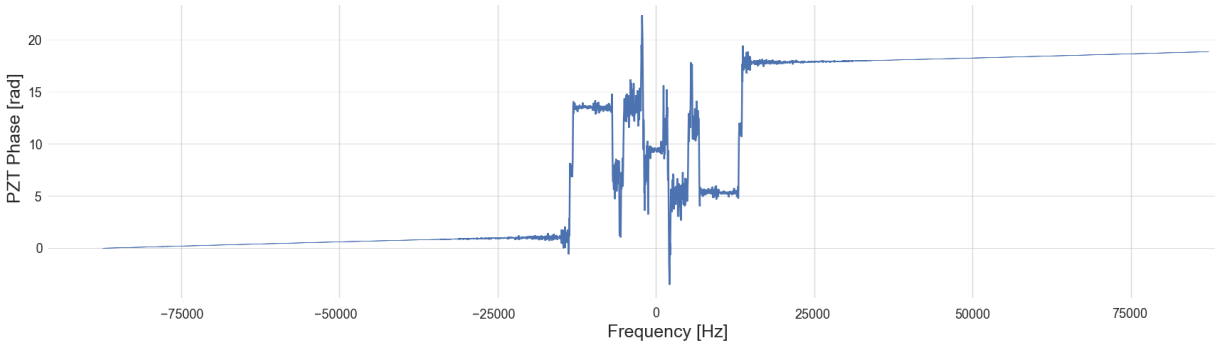
$$X_k = \sum_{n=0}^{N-1} x_n e^{-jK\omega_0 n} \quad \text{where} \quad \omega_0 = \frac{2\pi}{N} \tag{13}$$

Similarly to the FT, the DFT also indicates the "amount" of frequency $k\omega_0$ contained in the original signal. The DFT is considered a real workhorse in the computational signal analysis world, due to a fast and efficient algorithm for calculating it called the Fast Fourier Transform (FFT), and is the de facto standard to calculate a Fourier Transform, present in almost any scientific computing libraries and packages, in every programming language, including `Python` [46, 49].

Figure 15 shows the FFT, from the SciPy library [9], applied to an adhesive joint test, programmed to retrieve the frequency-based information of its component, with the top 5 highest frequency amplitudes marked, as well as the phase content, shifting the zero-frequency component to the center of the spectrum:



(a) Amplitude



(b) Phase

Figure 15: Fast Fourier Transform representation of an adhesive test signal with the top 5 peaks marked. Adhesion strength = 50100 kPa.

The frequency content of the signal is expectedly diverse, with the main frequency response band around 100 kHz (the excitation frequency), mainly between 75 and 125 kHz, but with some residual spectral content across the whole range. Again, the top frequencies corresponding to the highest amplitudes are marked and saved as a feature, but a lot of other frequency domain features can be constructed from the spectral information, such as the FFT phase angle content, the Power Spectral Density (PSD), Spectral Moment, etc.;

Time-Frequency domain features

It is clear that the Fourier Transform is of great use to transform a signal into the frequency domain, where it has great resolution, at the expense of the time domain information, as it contains none. In other words, it is known at which frequencies the signal oscillates, but not **at which time** these oscillations occur. So if a signal has a dynamic frequency spectrum, i.e. the frequency content changing over time or frequencies appearing abruptly for a short period of time, the Fourier Transform will not expose them. For that, the Time-Frequency domain is approached, specially through the Short-Time Fourier Transform (STFT) or the Wavelet Transform, among others, like the Hilbert-Huang Transform [46].

The STFT can be used as a way of quantifying the change of a non-stationary signal's frequency and phase content over time, by dividing a time based signal into shorter segments of equal length and then computing the Fourier Transform in each of those segments separately, therefore keeping in the results of the magnitude and phase content for each point in time. The use the Fast Fourier Transform on these segments yields the discrete-time STFT, expressed as:

$$STFT_{x(k)}(m,n) = \sum_{k=0}^{L-1} x(k)g(k-m)e^{-j2\pi nk/L} \tag{14}$$

where $x(k)$ denotes the discrete time signal, that is multiplied by $g(k)$, an L-point window function with a fixed size n, that will divide the signal into chunks as it slides through the signal with m amount of shift. The FFT is then applied to each of these chunks to compute the STFT [50, 51]. The Hamming window, a smooth, "bell-shaped" curve is a popular choice for the window function employed, shown on Figure 16. The STFT can also be called a Gabor transform, if the window used is a Gaussian function [52].

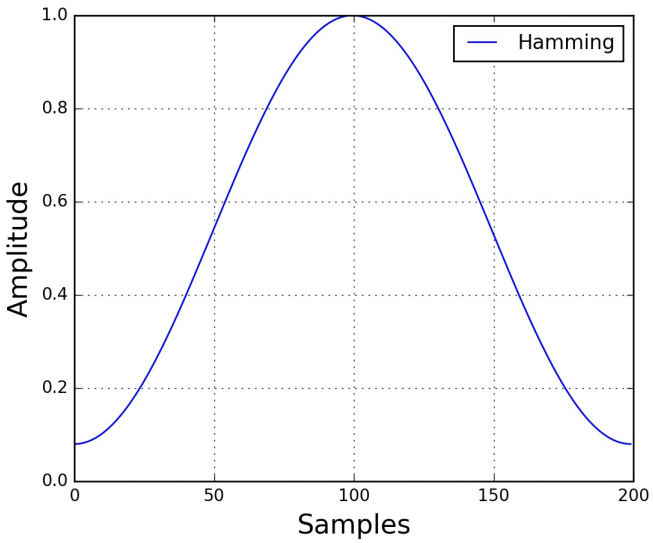


Figure 16: Hamming Window function. From [9].

The STFT is utilized to construct what is called the Spectrogram, a 2-D visualization that plots, in a logarithmic scale, the normalized, squared magnitude of the STFT coefficients [50,53]. There are some limitations, however, to this operation. The fact that the window function has a fixed size (frequency modulation bandwidth) causes a constant time-frequency resolution, and as Heisenberg stated in his classical Uncertainty Principle in 1927, *the position and the velocity* (in this case, frequency) *of an object cannot both be measured at the same time, even in theory* [54]. This means that there is a trade-off in resolution, having a narrow-width window promotes great resolution in the time domain, but poor resolution in the frequency domain, and vice-versa [50]. In addition, using a fixed window length, the STFT still cannot capture events with different duration or when the signal contains fast (sharp) events [51]. Thus, finding a suitable window function and time-frequency parameters *a priori* for an ideal STFT implementation on an arbitrary non-stationary signal can be rough and unpractical [55].

Another transformation from the original time signal to the time-frequency domain is the Wavelet Transform, a powerful variant of the Fourier Transform that tries to mitigate the STFT limitations. Starting by defining wavelets, a concept first introduced in 1982 by french geophysicist Jean Morlet [56], they are a family of functions obtained from a single prototype, a special basis function called "mother wavelet" $\psi(t)$ - a small, time-bound, wave-like oscillation that decays quickly. This basis function is drawn from a large dictionary of possibilities, has both temporal and frequential components, and the family of wavelets is obtained by dilations and contractions (scaling) in addition to translations. They are defined by [56]:

$$\psi_{a,b}(t) = \left(\frac{1}{\sqrt{|a|}}\right)\psi\left(\frac{t-b}{a}\right) \tag{15}$$

The parameters $a, b \in R$, with $a(\neq 0)$ being the scaling parameter, measuring the degree of compression, and b the translation parameter, that determines the time location of the wavelet. This results in time-widths that are adapted to the wavelet frequencies, if $|a| < 1$, then we have a compressed version of the mother wavelet, that will correspond mainly to higher frequencies, while $|a| > 1$ makes for a $\psi_{a,b}(t)$ with a larger time-width than the mother, corresponding to lower frequencies. This is the main reason for the success of the Wavelet analysis, as it creates a multiresolution analysis of the signal [52].

To better understand this reason with an analogy, the wavelet analysis can be compared with a microscope. First, one chooses the magnification, a large wavelet. Then one moves to the chosen location, the translation parameter, then small steps in reduction of magnification (shorter wavelets from the family) are used to catch small details, creating a coarse to fine gradient of analysis [52].

Given the family of affine wavelets, any signal $f(t) \in L^2(\mathbf{R})$ can be expressed as the

wavelet series of f:

$$f(t) = \sum_k \sum_l d_{k,l} \psi_{k,l}(t), \tag{16}$$

where the wavelet coefficient comes:

$$d_{k,l} = \langle f(t), \psi_{k,l}(t) \rangle = \int_{-\infty}^{\infty} f(t)\psi(t)dt \tag{17}$$

These coefficients are really the principal takeaway from the whole operation, and are kept as a feature. In the context of this project, for each parameter combination on the Ricker Wavelet, an array of coefficients (one for each test) calculated with the Continuous Wavelet Transform, is saved.

A typical choice for a family of wavelets, specially for the Continuous Wavelet Transform (CWT), is the Ricker Wavelet, shown in Figure 17. It is the negative normalized second derivative of a Gaussian function, also known as the Marr Wavelet or the "Mexican Hat" Wavelet in the Americas, for its resemblance with a *"sombrero"* hat [57]:

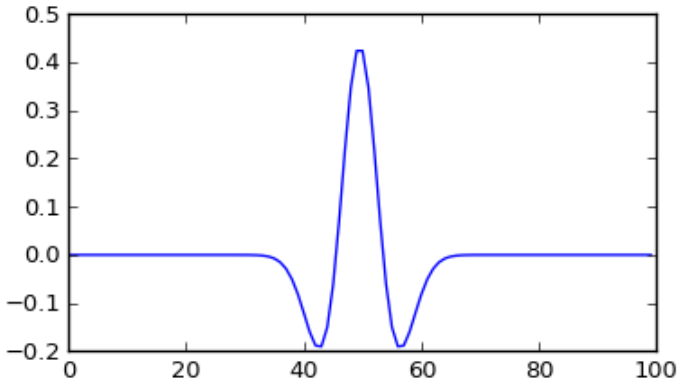


Figure 17: Ricker Wavelet. From [9].

The Wavelet Transform is without question a remarkable tool for analysis of non-stationary data, assuredly more versatile than the Short-Time Fourier Transform. Nevertheless, both are linear decompositions, thus suffering from problems that arise form the Uncertainty Principle. Moreover, their basis functions are established *a priori*, making them prone to spurious harmonics or ultimately incorrect interpretations of the data. To overcome these drawbacks, methods like the Hilbert-Huang Transform (HHT), an "empirical mode decomposition" method with which any complicated data set can be decomposed into a finite and often small number of "intrinsic mode functions", but in an adaptive way, based on the local characteristic time scale of the data [55, 58].

It is easily perceived how the manual feature extraction process is difficult, time consuming and sometimes inaccurate, or just not worth the effort for a single feature that might not be useful at all for a given ML algorithm. So ways of generating features from

original time series data must be developed, in a better, more efficient manner, extracting the highest number of features possible in a short amount of time, that can then be processed and chosen from with purpose.

3.4 Visualization techniques

Up until now, all the feature extraction procedures or plots that were shown, either original time-based signals from the sensors or their frequency domain transforms, pertained to one single testing instance. But if a classification based on those signals is to be made, at some point the various instances of testing have to be compared in some way, and the first step on that comparison is through the visualization of the signals as an aggregated group. This is not trivial for the sensor data, as it is well understood by now, this data set is a time-dependant one, composed of a large number of observations for each test (resulting in a high-dimensional collection of real vectors). So, just as in the concept of features, dimensionality reduction must be applied; but instead as a way of extracting information from the data as unidimensional features, it needs to be applied as a procedure to create projections of that data, mapping it into a low dimensional space, in order to enable an insightful view into how each case relates to the others, namely the appearance of structures such as high local density, interesting relations between observations and the presence of clusters [59, 60].

Time-oriented data visualization is a widely researched topic, and according to Aigner, et al., visualization techniques can be categorized in many ways, for instance the structure of time itself (linear, cyclic or branched), the frame of reference (spatial or abstract), the number of variables that are time-dependant, and the dimensionality that is to be obtained (2D or 3D, since those are the intuitively understood by the human brain). Most of the approaches published in recent years are specific to a particular problem, but there are also generic ones suitable for simple tasks [61].

At this stage, two of the most widespread visualization techniques are implemented for the projection and visualization of the time series themselves, and will also be applied to some select features later on. They are Multidimensional Scaling (MDS), and t-distributed Stochastic Neighbor Embedding (t-SNE).

It is a common procedure in data mining and DS methods to transform the raw data formats into more suitable and consistent ones, through processes like smoothing, generalization and normalization. Normalization is likely to improve accuracy and efficiency of classification algorithms, and for distance-based methods in particular it prevents attributes with initially large value ranges from outweighing those with smaller ranges, uniforming the importance of any given testing instance the while keeping the intrinsic information intact. Among others, data normalization methods include Min-Max normalization, used in this project, Z-Score normalization, l^2 normalization and normalization by decimal scaling [62, 63].

Min-Max normalization performs a linear transformation on the original data. If an

attribute A has a range of values of $[a_{min}, a_{max}]$, Min-Max normalization maps a value x of A, to its counterpart x' in the new predetermined range $[a'_{min}, a'_{max}]$ by computing [62]:

$$x' = \frac{x - a_{min}}{a_{max} - a_{min}} \times (a'_{max} - a'_{min}) + a'_{min} \tag{18}$$

where, in this case, the new selected range was $[a'_{min}, a'_{max}] = [0, 1]$, a common, practical choice.

From the `scikit-learn` library in `Python` [64], code for the computation of not only the Min-Max Scaler, but also the MDS and t-SNE tools to be implemented, is available and well documented.

Multidimensional Scaling (MDS)

Multidimensional Scaling is a multivariate statistical method that represents measurements of proximity/similarity (or dissimilarity) among pairs of objects geometrically, as distances between points of a low-dimensional space. It has its origins in psychometric analysis, where it was introduced to help understand people's judgements of the similarity among members of a set of objects. Torgerson proposed the first MDS method and coined its name, Multidimensional Scaling, that can also be known as Principal Coordinates Analysis (PCoA) or even Torgerson Scaling, in his honour [65].

In its application at this stage of the project, it shows the correlations among instances of the simulation tests, displaying each of these tests as a point on a plane, so that the closer together the points are, the more positively the respective tests are correlated, turning the data from immense arrays of numbers to an accessible visual representation, for easy inspection and exploration [66].

This application of MDS, like most in the research community, is exploratory, designed to uncover the data elements accounting for the proximity of the data, rather than test *a priori* hypothesis about the existence and properties of those elements [67]. The vastly used family of procedures known as *Principal Components Analysis* (PCA), which should not be mistaken with PCoA, is closely related to MDS in function, but differs in some key aspects, the principal being the fact that MDS starts with a matrix of similarities between a set of individuals, while PCA starts directly with the initial data matrix, but in many cases Euclidean distances are used, their output will be similar [68, 69].

MDS models are defined by the similarity or dissimilarity of data - the proximity indexes p_{ij} between pairs (i, j) of objects, that construct an $n \times n$ matrix $\mathbf{C}$, being n the total number of objects - and how those proximity indexes are mapped into distances of an m-dimensional MDS space configuration: $\mathbf{X}$. In classic MDS, $\mathbf{C}$ is symmetric, with $p_{ij} > 0$ for $i \neq j$ and $p_{ii} = 0$, and its main diagonal is composed of "1" [60].

The mapping is given by a representation function $f(p_{ij})$, that specifies how the proximities should be related to the distances $d_{ij}(\mathbf{X})$, seeking the configuration (in a given dimensionality m) whose distances satisfy f as closely as possible. The condition "as

closely as possible" is quantified by a badness-of-fit measure, what's called a *loss function* - an expression that aggregates the representation errors : $e_{ij} = f(p_{ij}) - d_{ij}(\mathbf{X})$. The most common loss function in MDS is named raw-Stress (ρ), also known as "Kruskal stress" due to its creator [68], a normed sum-of-squares of these pairwise errors, and minimizing this function leads to the most accurate representation of the data. Stress varies between 0 and 1, with values near 0 evidently indicating a better fit [66, 70].

$$\rho = [f(p_{ij}) - d_{ij}]^2, i, j = 1, ..., n \tag{19}$$

Shepard plots are also used to evaluate the fit of the mapping, by comparing d_{ij} versus p_{ij} for a given value of m. A narrow scatter of points, resembling a smooth straight line without sudden steps indicates a successful representation [66, 68, 70].

The m-dimensional MDS space configuration always refers to a coordinate system, customarily a set of m directed axes, perpendicular to each other and intersecting in one point, the origin. If the value for m is chosen to be 2, for example, this would define a Cartesian plane. Since the MDS interpretation is based on the emerging clusters and distances between points in the mapping, rather than on their absolute coordinates, the units of the axes are meaningless, and so the MDS map can be rotated and translated, as the distances between points remain the same [60,66]. As for the distances' definition, the most natural and frequently used is the Euclidean distance, corresponding to the length of the straight line segment connecting i and j, computed by the formula [66]:

$$d_{ij}(\mathbf{X}) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2} \tag{20}$$

Thus, $d_{ij}(\mathbf{X})$ equals the square root of the sum of the intradimensional differences $x_{ia} - x_{ja}$, which in plain terms is the Pythagorean theorem for the length of the hypotenuse of a right triangle. Generalizing to the m-dimensional case, yields:

$$d_{ij}(\mathbf{X}) = \left[\sum_{a=1}^{m} (x_{ia} - x_{ja})^2 \right]^{1/2} \tag{21}$$

The dissimilarity matrix $\mathbf{C}$ can adopt different measures, such as the so-called Canberra and Manhattan distances among others. It should be noted that the use of alternative measure methods within the MDS is a common procedure, often having distinct representations of the same data set, that view phenomena with different perspectives, opening up the possibility of choosing the MDS charts that yield better visualizations [70]. For this project however, only the Euclidean distance was used, since this method was only used as a preliminary visualization tool, and is not really included in the classification algorithm beyond that.

The results for the MDS application for the LW response signal data sets, for both the aluminium plate and adhesive joint simulations, will be displayed now in a 2-dimensional configuration. It is important to point out that the axes in the plot have no units nor a physical meaning, they represent just the values of a Cartesian plane.

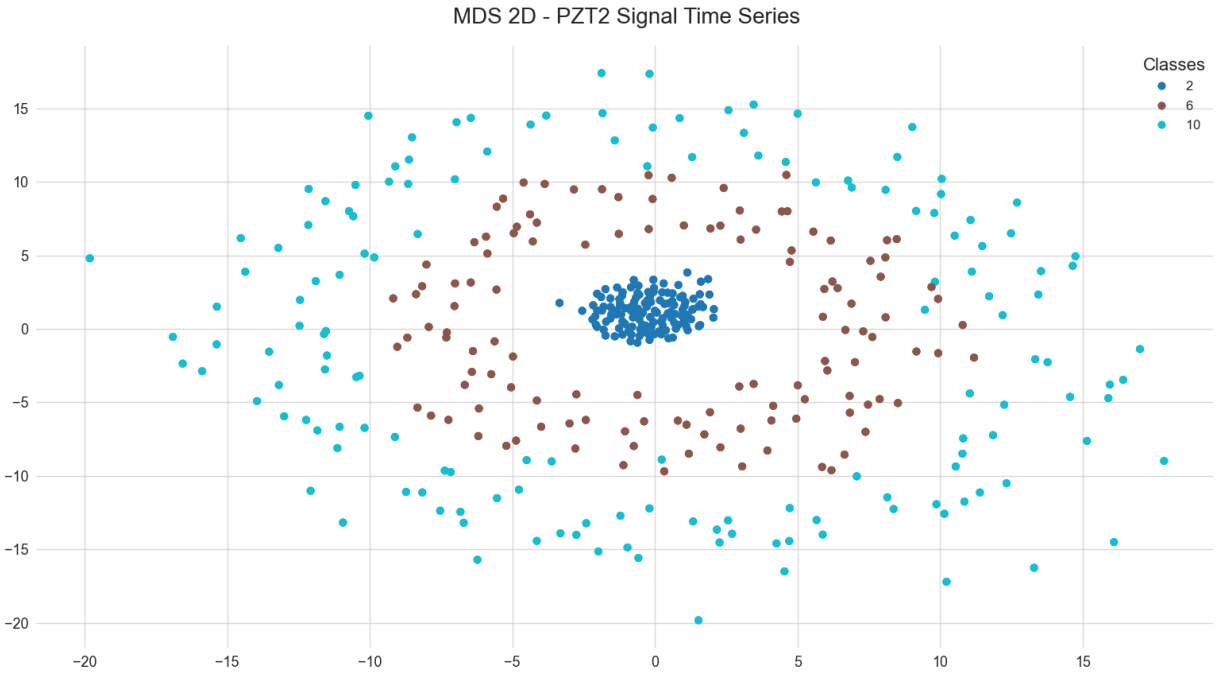


Figure 18: 2-D MDS Visualization of the aluminium plate with holes simulation tests (only damaged tests).

The MDS plot in Figure 18, retrieved from the PZT2 sensor (diagonally opposed from the PZT actuator), each of the points displayed represents one single test, coloured according to the respective size of the hole present on the test, either 2mm, 6mm or 10mm, not containing the tests with no hole. Please note that the MDS plots from the other two sensors, PZT1 and PZT3 are extremely similar to this one in every aspect, and if all three sensors' signals were represented, the plot would be redundant and three times denser. The representation shows clearly that the relation between signals is not random, there are definitely grounds to assume that they differentiate themselves distinctively according to a pattern: the 2mm hole class can easily be distinguished from the other two, since all the 2mm hole tests stand together, with minimal distance from each other; it can also be told that the pairwise distance between tests from the 6mm hole class is shorter than that of a pair from the 10mm hole class.

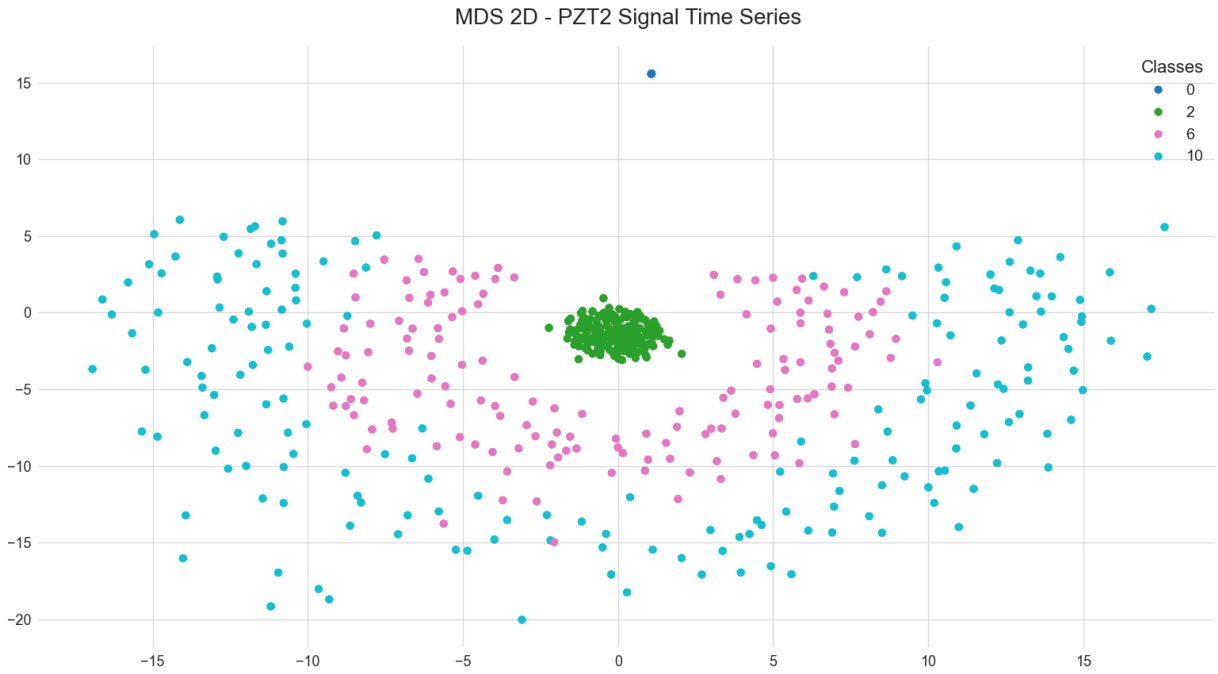


Figure 19: 2-D MDS Visualization of the aluminium plate with holes simulation tests (all tests).

In Figure 19, now including the tests with no damage (take into account that the simulations are deterministic, so the simulations without any damage will always return the same resulting signal every time), it is visible that the respective time series also distance themselves significantly from all the remaining, which is very promising for a simple damaged vs not-damaged classification.

However, these results should be taken carefully, as the mapping does not show a clear-cut division in three well separated clusters, which would be an ideal scenario. On the contrary, the most distant pairs of tests are part of the same class, 10mm hole. That may very well be caused by the fact that the hole position on the plate, while not directly represented on the plot, is a greater influence on the signal if the hole size is larger, which actually makes sense intuitively, as the propagation of the LW is less affected if the material void is smaller, wherever it may be encountered.

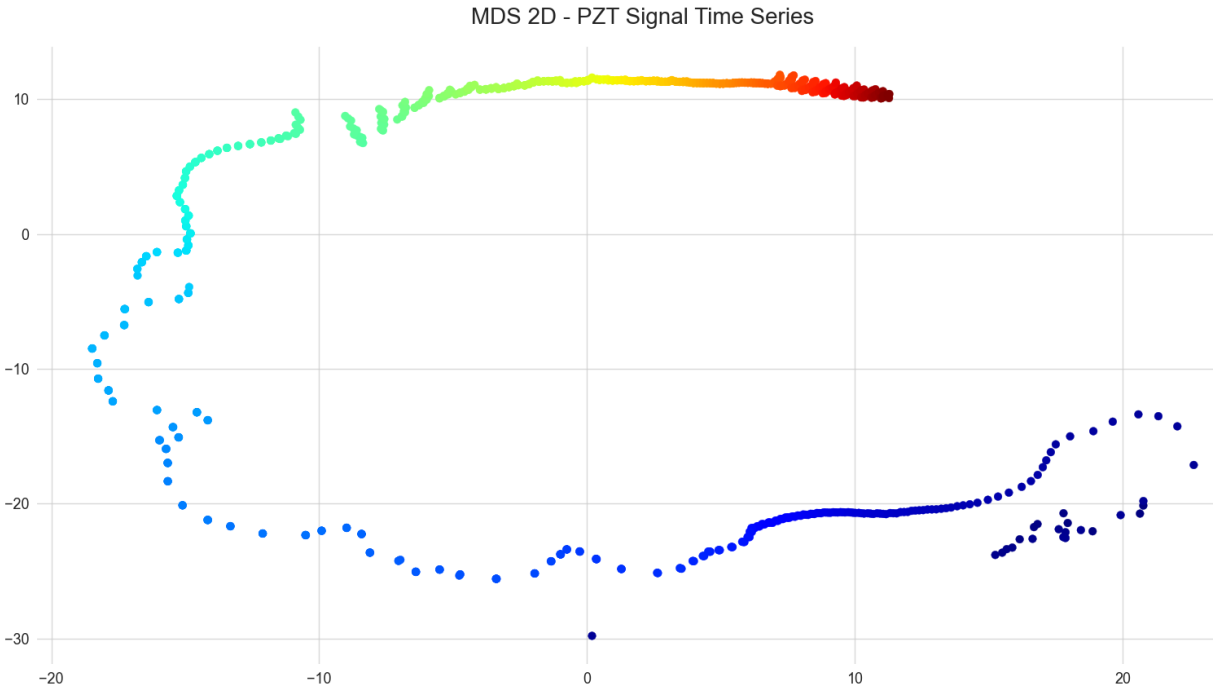


Figure 20: 2-D MDS Visualization of the adhesive joint simulation tests.

Considering now the adhesive joint testing MDS chart in Figure 20, the absence of legend is due to the fact that the results are not shown per class, but directly using the adhesion strength value, from the weakest adhesion being the deepest blue to the strongest adhesion in the dark red, yielding the direct realization that the LW propagation signals are as closely related as the adhesion strength itself, more so on the strongest end of the range, where an increase in strength doesn't raise the tests pairwise distance as much as the same increment of strength in the medium-lower range. For the construction of this data visualization plot, there was no need to employ three dimensional MDS, since the added dimension's values would not vary, and the 2-D visualization is perfectly suited to illustrate the relations among the data.

t-distributed Stochastic Neighbor Embedding (t-SNE)

Stochastic Neighbor Embedding (SNE), as originally presented by Hinton and Roweis [71], is a probabilistic approach to the task of placing objects, from high-dimensional vectors or by pairwise dissimilarities, in a low-dimensional space in such a way that preserves neighbor identities. It serves the same purpose, but unlike MDS, which is a linear technique that operates without requiring statistical distribution assumptions, SNE makes use of a Gaussian distribution, centered on each object of the high dimensional space, and the densities under this Gaussian are used to define a probability distribution over all the potential neighbors of the object [68, 71].

Briefly, it starts by converting the high-dimensional Euclidean distances between datapoints, x_j and x_i into conditional probabilities $p_{j|i}$, representing the probability that x_i would pick x_j as its neighbor, if neighbors were picked in proportion to their probability

density under a Gaussian centered at x_i. This probability will be high for nearby points, and almost infinitesimal for widely separated ones.

Now for their low-dimensional counterparts, y_i and y_j, a similar conditional probability is computed (but with a fixed variance), denoted by $q_{j|i}$. If these conditional probabilities are equal, in other words, if the distributions are matched, the model is correctly mapping the data [71, 72].

So, the aim of the embedding is to match the distributions as well as possible, which is achieved, in a similar fashion to MDS, by minimizing a cost function. A natural measure of the faithfulness with which the probabilities match is the Kullback-Leibler divergence, therefore, a cost function C is defined by a sum of the Kullback-Leibler divergences between the original ($p_{j|i}$) and the induced ($q_{j|i}$) distributions over neighbors for each object. SNE minimizes C using a gradient descent method [71, 72].

$$C = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \tag{22}$$

The t-SNE is a variation of SNE, with two important differences, aiming to alleviate two problems with the SNE method [71, 72]:

- Because the Kullback-Leibler divergence is not symmetric, different types of error in the pairwise distances in the low-dimensional map are not weighted equally, for instance, there is a large cost for using widely spaced map points to represent nearby datapoints, but a small cost for using neighboring map points to represent datapoints that are far apart. In t-SNE, conditions to this function are applied to ensure that it is symmetrized, eliminating that problem, and also, since its gradient has a simpler form, is faster to compute. It is said to be symmetric because it has the property that $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$, $\forall i, j$, hereby named joint probabilities;
- The issue known as the "Crowding Problem", lies in the fact that if small distances are to be accurately mapped, the moderately distanced datapoints will be placed much too far away in the representation. As a solution, in t-SNE, instead of using a Gaussian distribution on both the high and low-dimensional probabilities, it is swapped by the heavy-tailed Student t-distribution with one degree of freedom (also known as Cauchy distribution) on the low-dimensional map, q_{ij}, using the mismatched tails on the distribution to compensate the mismatched dimensionalities on the data. It also speeds up the computation of the process since Student t-distribution does not involve an exponential.

There are various parameters that can be tinkered with in a t-SNE application in order to optimize the cost function gradient descent, some of which are the number of iterations, early exaggeration, learning rate and perplexity, interpreted as a smooth measure of the effective number of neighbors; so adjusting all these parameters yields different results even for the same data set [72].

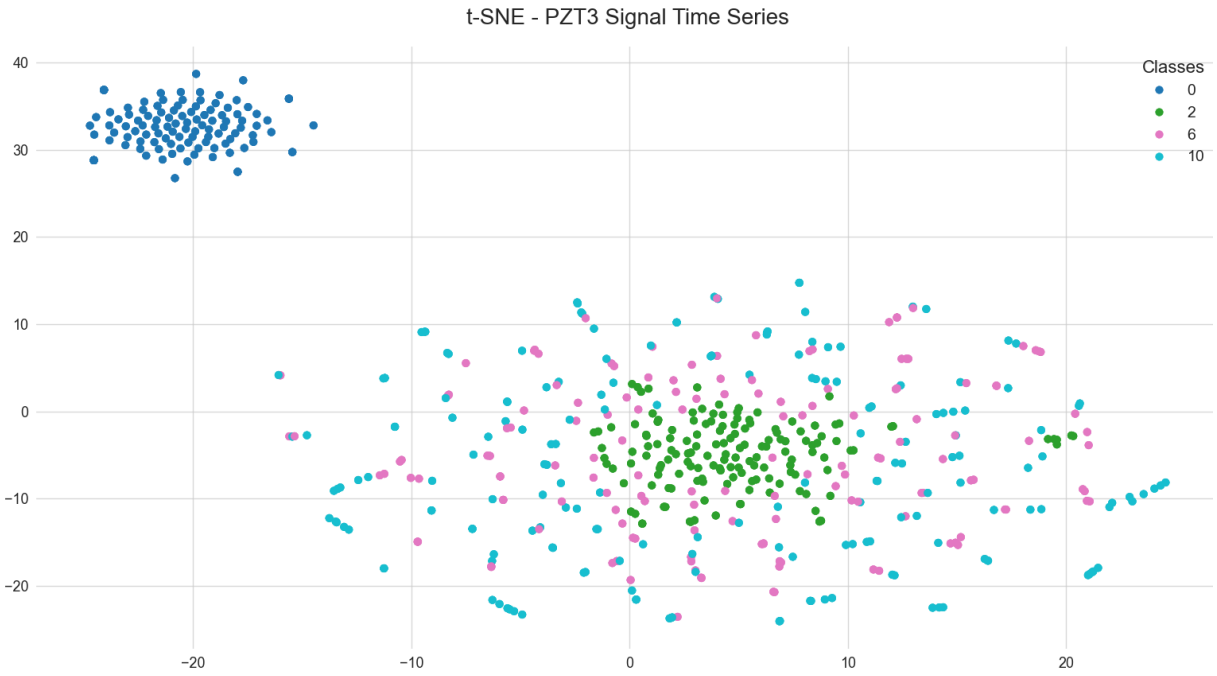


Figure 21: 2-D t-SNE Visualization of the aluminium plate with holes simulation tests.

The mapping on Figure 21 is the standard t-SNE plotting of the tests. Again, please note that the axes in the plot have no units. The parameters were tinkered with during the implementation, but they did not yield any particular pattern, not any glaring evidence of clustering or orientation, apart from the unequivocal separation of the tests with no damage. Notice that they are not condensed into a single point as in the MDS, since this method is not strictly dependent on Euclidean distanced. The tests from the 2mm hole size class are generally closer together, and considerably distanced from the rest. There are some local structures, particularly in the 10mm hole class, where sets of 3-5 tests forming a straight line, that also happens in the other two sensors' representations, and that remain even with heavy variations of the t-SNE parameters, but not much can be inferred.

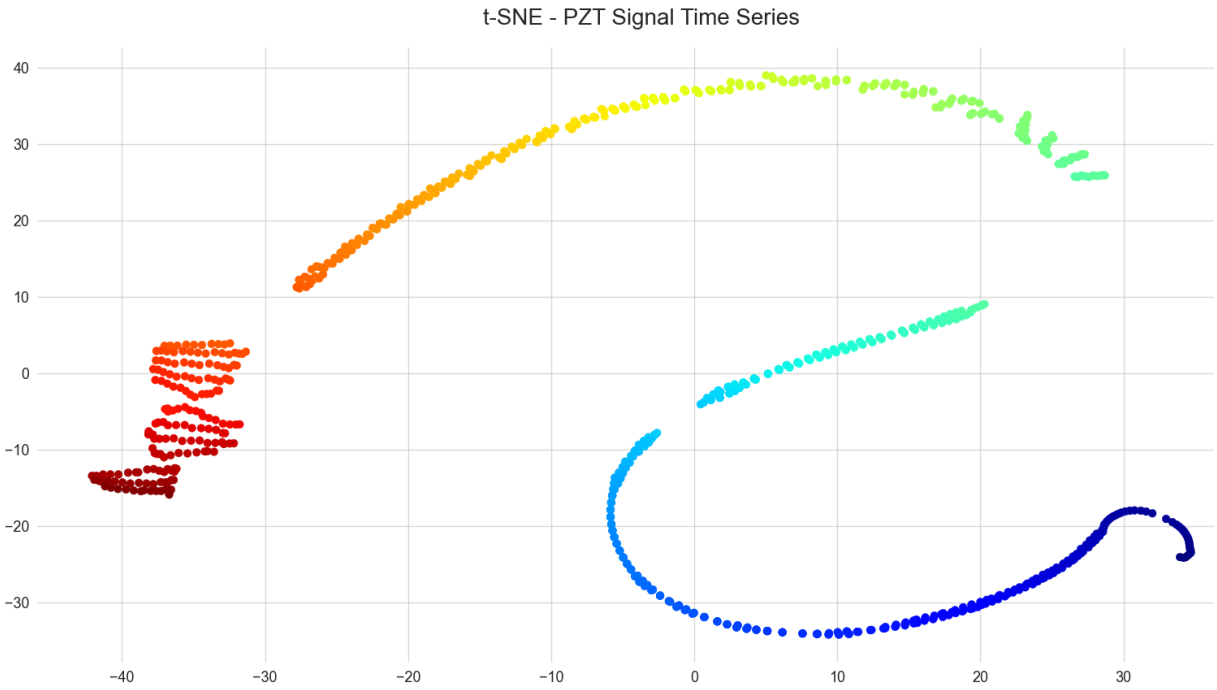


Figure 22: 2-D t-SNE Visualization of the adhesive joint simulation tests.

Figure 22 presents the application of bi-dimensional t-SNE to the adhesive joint testing data, following the color map convention used on the MDS visualization, and the result has the same overall character, with a steady evolution of distances following the adhesion strength increase, maybe even more smoothly than on the MDS, and with a .

The amount of methods and techniques to visualize high-dimensional data in low-dimensional maps preserving the data's intrinsic structure is huge, both linearly and non-linearly, with various distances, loss functions, optimization parameters, etc. Obviously not all of them are suitable for every application, and exhaustively going through them one by one and scrutinizing their results on the same database does not seem practical nor rewarding in any way, since a consistent, clean, and properly preprocessed data set will be fairly well visualized in most of them.

Be that as it may, both the MDS and t-SNE applications on the original time based signals were great for a preliminary observation of the whole data set, and actually quite insightful in some aspects, but further analysis must be developed to successfully establish a damage classification framework.

4 Automatic Feature Extraction

4.1 Projecting a machine learning pipeline

After defining the concept of feature and presenting its value to a ML algorithm, the main body of the project and the approach that was taken to process the data can be presented - the construction of an automatic feature extraction tool.

Viewed as one of the general topics of feature engineering, Automatic Feature Extraction is a methodology capable of automatically generating a large number of features from a data set and subsequently selecting an effective subset of these features to be applied in the ML algorithms. To have the most number of features possible being initially extracted from the time series is the best way to differentiate the information within the signals, compiling it in all the domains discussed in the previous chapter, and then choosing the best, most meaningful and adequate to the problem at hand, so that ML algorithms can use them in the most effective way possible to detect damage [40].

The feature extraction process is only the first step (after first cleaning and preprocessing of the raw data) in what will be the automated data analysis and damage detection pipeline. Figure 23 shows the general workflow of an SHM method, which perfectly fits this project's essence:

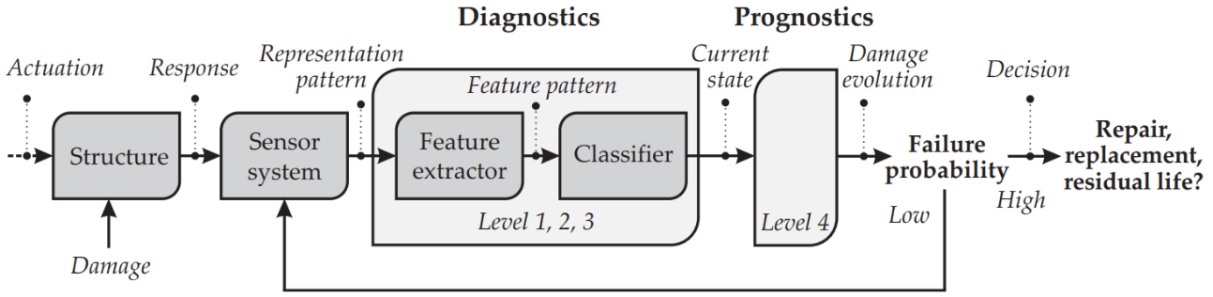


Figure 23: Damage Detection ML Pipeline. From [10].

Level 1 is concerned with the first analysis of the system's response to the mechanical actuation, through LW generation and propagation, much like the analysis conducted and presented in Chapter 3, as well as the feature extraction process, that will be discussed in the current section of the report.

Levels 2 and 3 are both involved with taking those features and applying them as the input to classification ML algorithms. Level 2 really comes down to answering the question: "Does any damage exist within the structure". If the answer is no, then it would be safe to assume the failure probability is low, and that would be the end of the inspection. If the answer is yes, however, then Level 3 is called upon, to retrieve information regarding the damage, its location and characterization, answering the questions: "Where, what kind, and how severe is the concerned damage?".

Level 4 would go beyond these questions, being the stage where an evaluation is made,

following perhaps a protocol or standards based on previous experience and testing, to try to approximate the residual lifetime and failure probability, given the information coming from Levels 2 and 3. It falls outside of the scope of this project, which focuses on Levels 1 and 2.

The automation of the feature extraction process was really the key, the main point of emphasis of the whole project, and so, a research on the state of the art of feature engineering `Python` programming libraries and modules was in order. A few candidates were tried out before fully implementing the most suitable one - `tsfresh`. But before detailing its characteristics and implementation, an acknowledgement is made to all the potential libraries that could have been used for this purpose, as well as the basic, commonly used libraries that allowed for the handling, preparation and representation of the data within `Python`.

4.2 Programming libraries

`Python` programming is very much reliant on community developed libraries and packages that specialize on certain tasks, than can easily be called upon and incorporated into personal functions and scripts. They play a vital role in the development of code in `Python`, whether for data science, ML, or really any other area.

The following packages were imported and used extensively along the development of the project [73]:

- `Matplotlib` - Uses `Python Script` to write and represent 2-D and 3-D graphs and plots, with a `MATLAB`-like interface [74]. All the graphs and plots presented in this report were generated using this tool;
- `NumPy` - Popular, fast and efficient array processing package, having the tools to manage and operate arrays and matrices, facilitating the data managing [75];
- `pandas` - Purposefully written for `Python` language, it is a must learn package for data science endeavors, setting up an intuitive and adjustable platform for the manipulation and organization of structured data, e.g. time series [76]. The `DataFrame` format with smart indexing and data labeling were crucial to the data organization in the feature extraction process;
- `SciPy library` - A collection of numerical algorithms and domain-specific toolboxes, essential for signal processing, statistics, interpolation, linear algebra, among others [9];
- `scikit-learn` - Simple and useful ML library, `sklearn` (as it is abbreviated) works in complete harmony with `NumPy` and `SciPy`. Very clean and neat Application Programming Interface (API), contains a variety of ML algorithms and procedures, from

dimensionality reduction to clustering, classification and regression, with excellent documentation making it very beginner-friendly [64].

All of these (except for `sklearn`) are actually a collection of core packages belonging to the `SciPy` ecosystem, pretty much the gold standard end effectively the cornerstone of advanced `Python` programming, since the vast majority of the community libraries and packages are built upon the structures and platforms provided by this core.

As for the automatic feature extraction, some prospects were initially identified as potential candidates to integrate the classification pipeline as feature extractor, namely `hctsa` (on `MATLAB`), `featuretools`, `FATS`, `Cesium`, `TSFEL` and `tsfresh`. Most of these specialize in time series feature extraction, and the first one to be tried out was `TSFEL` (Time Series Feature Extraction Library). Developed by Fraunhofer AICOS Portugal, this package provides exploratory feature extraction tasks on time series without requiring significant programming effort, automatically extracting over 60 different features on the statistical, temporal and spectral domains [77]. It was successfully implemented on the aluminium plate with holes testing data set, extracting the complete set of features. Having that working solution to fall back on, the next candidate was tested, which ultimately ended up being the chosen one for its outstanding performance and far-reaching package.

4.3 `tsfresh`

The **T**ime **S**eries **F**eatu**R**e **E**xtraction on basis of **S**calable **H**ypothesis tests (`tsfresh`) is a `Python` package developed by Christ et. al [11], designed to spare data scientists and engineers of the multifarious task of considering dozens of signal processing algorithms and time series analysis, by combining different time series characterization methods and using them to compute features, with and added tool of feature selection based on statistical significance for predicting a target.

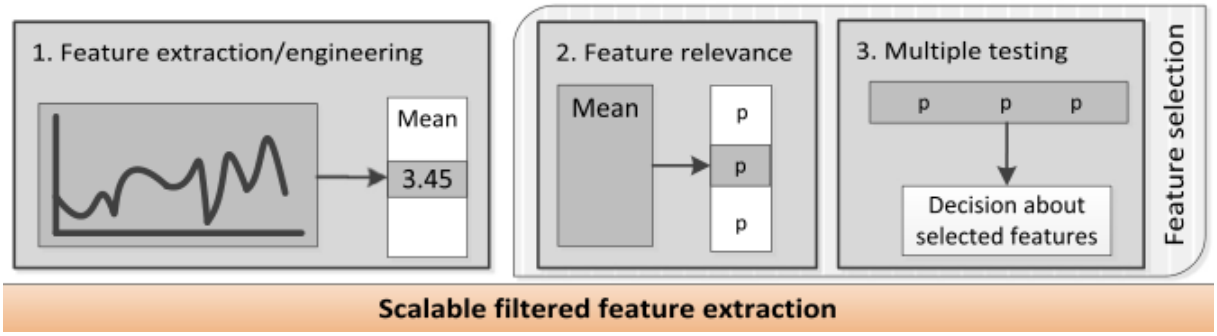


Figure 24: `tsfresh` three-step process for feature extraction and selection. From [11].

Figure 24 illustrates the three-step progression of the `tsfresh` algorithm. The first step is the feature extraction from the time series, resulting in a $M \times N$ matrix, where the M rows correspond to the time series, identified by their `id`, while each of the N columns

correspond to an extracted feature, with a unique identification composed by: the kind of time series (which sensor) that originated the data, followed by the feature calculator name, and lastly the key-value pairs of parameters configuring the respective feature calculator. So, for instance, a column named *"zt3_spkt_welch_density_coeff_5"* would contain the values of the estimates for the cross power spectral density at a designated frequency ("coeff = 5"), extracted from the time series signals of sensor PZT3. A sample of the feature matrix is shown on Figure 25:

ID	zt1__fft_coefficient__attr_"angle"__coeff_64	zt2__change_quantiles__f_agg_"mean"__isabs_True__qh_0.6__ql_0.4	zt2__fft_coefficient__attr_"angle"__coeff_25	zt3__benford_correlation
0	-165.021	5.35231e-08	162.193	0.980251
1	-166.597	6.29235e-08	-159.883	0.97646
2	164.847	4.48842e-08	67.9614	0.965603
3	-171.061	6.10006e-08	158.88	0.969739
4	159.306	7.38881e-08	171.275	0.962898
5	158.972	9.23835e-08	133.348	0.912316
6	-173.121	5.3202e-08	155.487	0.975048
7	159.821	4.74425e-08	87.5884	0.91597
8	154.682	6.82938e-08	143.163	0.955324
9	161.466	4.52758e-08	102.138	0.964703
10	143.166	5.80039e-08	-152.86	0.959445
11	-69.0983	6.33425e-08	167.786	0.951394
12	-178.289	4.77109e-08	152.026	0.979217

Figure 25: `tsfresh` feature matrix sample.

The second step, which can be run separately, in parallel with any other ongoing feature extraction thread, is the estimation of each feature's relevance to a given ML task, through hypothesis tests and calculation of the respective p-values. The hypothesis tests are automatically configured depending on the type of supervised ML problem (classification/regression) and feature type (categorical/continuous). The third step involves a multiple testing procedure, utilizing an instrument called *Benjamini-Hochberg Method* to control the false discovery rate (FDR). These two steps comprise the feature selection component, and will be expanded upon just ahead.

The second and third steps' runtime is negligible compared to the first one. The feature extraction process, both on the adhesive joint and aluminium plate with holes data sets, took several hours to conclude.

The complete list of features from the feature extraction module and their meaning will be added as an Appendix at the end of the report.

Data preparation

The `tsfresh` package makes use of standard APIs and core packages of `Python` (e.g `pandas` and `scikit-learn`), allowing for a straight forward implementation. By deploying the `pandas.DataFrame` data structure as input and output objects, the only concern is that the data has to be prepared before undergoing the extraction process, since the output will always be returned in the same format, just like the sample shown above.

The input `DataFrame` has four important column types that must be correctly organized for a successful extraction [78]:

- `column_id` - indicates which entities the time series belong to, in this project's context, each test/simulation. Features will be extracted individually for each entity `id`, and the resulting feature matrix will contain one row per `id`;
- `column_sort` - contains the values that sort the time in the time series, that is, the array of time steps. It is not necessary, but recommended that every time series has the same amount of equidistant time steps. Such measure was already taken in the preprocessing of the data through an interpolation function with 5000 time steps;
- `column_kind` - indicates the names of the different series types (sensors). For each kind, the features will be calculated individually and independently;
- `column_value` - presents the actual values of the time series, the measurements taken by each sensor.

It should be noted that none of the columns is allowed to contain any `NaN` (Not a Number/null), `Inf` or `-Inf` (Infinity) values. This was also taken care of in the preprocessing phase. The input `DataFrame` will the be organized as follows on Table 1:

Table 1: The `tsfresh` Feature Extraction input `DataFrame` matrix

`id`	`time`	`kind`	`value`
1	t1	pzt1	pzt1(1,t1)
1	t2	pzt1	pzt1(1,t2)
1	...	pzt1	pzt1(1,...)
1	t5000	pzt1	pzt1(1,t5000)
1	t1	pzt2	pzt2(1,t1)
1	...	pzt2	pzt2(1,...)
1	t5000	pzt2	pzt2(1,t5000)
1	t1	pzt3	pzt3(1,t1)
1	...	pzt3	pzt3(1,...)
2	t1	pzt1	pzt1(2,t1)
2	...	pzt1	pzt1(2,...)
...	...	...	...
M	t1	pzt3	pzt3(M,t1)
M	...	pzt3	pzt3(M,...)
M	t5000	pzt3	pzt3(M,t5000)

The time series of a data set are stacked vertically, with repetitive patterns on the first three columns, and the values of each time series in the fourth.

Feature selection

The effectiveness of a feature is ultimately measured in terms of its performance to the ML task at hand, and whether or not it improves its metrics. In other words, whether or not the feature has relevance to the problem at hand [40].

An important aspect of feature selection concerns the computational power involved, and ultimately the time that it takes to complete a feature extraction process for a large data set. From a business perspective, it makes sense to restrict the feature extraction to the reduced, most meaningful pool of features, leaving apart the redundant, unnecessary and often detrimental ones, while speeding up the extraction process, saving time and resources.

For classification and regression tasks, the significance of extracted features is of particular importance, owning to the fact that too many irrelevant and redundant features will likely result in *overfitting* - a fundamental issue in supervised ML, where a possible number of reasons, like presence of noise, limited size of the training data set, or complexity of classifiers, impair the ability of the algorithm to generalize beyond the training data set. In other words, the model performs perfectly on the training set, while fitting poorly on the testing one, because it has "memorized" all the data on the training set, instead of learning the discipline hidden behind the data. For a well-functioning algorithms, this phenomenon must be avoided [79]. *Underfitting* is also a possible danger, namely if the model is not able to capture the dynamics shown by the training set (often because it is too limited).

The feature selection module of the `tsfresh` package contains a two-step method to evaluate the importance of different extracted features:

1. Univariate Hypothesis Testing;
2. Benjamini-Hochberg Procedure.

Firstly, the influence of each extracted feature on the target is evaluated, through a univariate hypothesis test, with the calculation of the respective p-values. To do so, the target vector must be defined, according to the intended ML task - either classification or regression. This topic will be further expanded on Chapter 5, but for this purpose, and since the ML algorithms that will be used are of the classification kind, the target vectors for both the adhesive joint and aluminium plate with holes testing can be promptly established.

All time series were generated on simulations ran with predetermined specifications - the adhesion strength of the joint, and the size/position of the hole present in the plate - with different values for this specifications, in order to construct a diverse data set. These values are fundamentally the needed target, the classes in which the data set can be divided into. They are the characteristic that is subject to analysis, and that the ML algorithms are being trained to predict. For the aluminium plate, the intention is

to classify the damage by predicting the hole existence/size (the position is disregarded in this project), so the target vector is simply composed by four classes: **0**, **2**, **6**, and **10**. Note that these do not represent integers with intrinsic value, they are just unique labels that describe the data, that is, one class label for each hole size present (none, 2mm, 6mm, and 10mm). For the classification of adhesion data however, considering that the adhesion strength value varies from 600 to 270000 kPa, five classes were arbitrarily assumed, by dividing that range in five equal intervals, intended to illustrate qualitatively the data: **1** - Very Low Adhesion Force; **2** - Low Adhesion force; **3** - Medium Adhesion force; **4** - High Adhesion force; **5** - Very High Adhesion force; The target vectors are then composed by the corresponding time series `id`, that should be consistent with the features `id`, of its respective class. An example is represented in Table 2:

Table 2: The `tsfresh` target vector examples for classification-task oriented feature selection

Plate with Holes			**Adhesive Joint**		
`id`	`target`	**Hole Size**	`id`	`target`	**Adhesion Strength**
1	2	2mm	1	1	600 - 54480 kPa
2	6	6mm	2	3	108360 - 162240 kPa
3	6	6mm	3	4	162240 - 216120 kPa
4	2	2mm	4	2	54480 - 108360 kPa
5	10	10mm	5	3	108360 - 162240 kPa
6	0	none	6	5	216120 - 270000 kPa
7	10	10mm	7	1	600 - 54480 kPa
...	6	6mm	...	4	162240 - 216120 kPa

The features' relevance will be tested individually with respect to each and every class contained on the given target vector, according to the specified ML task. To accomplish this, for each feature from the `DataFrame`, a univariate significance test is conducted.

An hypothesis test is a statistical inference procedure with the fundamental objective of assessing the plausibility of an assumption regarding a population parameter. Population is a pool of sampled data, in this case generated data. In other words, whether or not the data supports a claim made about a populational parameter. The test invariably leads to a positive or negative assessment of that claim, based on statistical characteristics of the data, either rejecting or failing to reject the given hypothesis. However, this defines what is called a parametric test, because the hypothesis must be explicitly about a parameter of the population, and it only works with continuous data, assuming it has an underlying statistical distribution [80]. Nonparametric tests, on the other hand, do not need to meet those criteria, and so, they can be applied to other data types such as ordinal or nominal/categorical data, as is the case. There are other differences between them, one

of which is the statistical test employed. Nevertheless, the structure of the procedure is the same for both, and can be decomposed in four steps [80, 81]:

- **Hypothesis Definition** - The first step is to determine the conjecture that will be tested. For this, two complementary claims are designated:

  - H_0 = the feature is not relevant and should be discarded;
  - H_1 = the feature is relevant and should be kept;

  In other words, if the **Null Hypothesis (H_0)** is not rejected, the feature and the target are assumed to be independent, meaning that the feature has no influence on the target's class. If however, the **Null Hypothesis (H_0)** is rejected, the **Alternative Hypothesis (H_1)** is assumed to be true, meaning the target and feature are associated/dependent;

- **Statistical Test Selection and Calculation** - The test statistic is a summary of the information contained in the data. The data that is being worked is of the categorical type, which means non-parametric statistical tests must be employed. The `tsfresh` automatically decides between four test settings, depending if the features and target are binary or not. The four possible statistical tests are:

  - *Two-sided Univariate Fisher Exact Test* - Used when both the feature and the target are binary;
  - *Mann-Whitney U* or *Kolmogorov-Smirnov* - Used when testing the significance of a real-valued feature to a binary target;
  - *Kolmogorov-Smirnov* - Used when testing the significance of a binary feature to a real-valued target;
  - *Kendall's Tau* - Used when both the feature and the target are real-valued;

  The value of the statistical test must be calculated in order to make a decision about the null hypothesis;

- **Significance Level Specification** (α) - The significance level α is an arbitrarily defined value (typically 0.05 or 0.01) that will dictate the **decision rule** after identifying the value of the statistical test, and guarantee that the rule is applied consistently throughout the data set. What the significance level represents is **the probability (risk) of rejecting the null hypothesis when it is true** (designated a **type I** error). So if the α is set to 0.05, that means there is a 5% chance of wrongfully rejecting the null hypothesis. The value of each statistical test *(ST)* is compared to the value of $ST(\alpha)$, named critical value, and from that comparison a decision is made. Obviously, the smaller α is, the more confidence of not committing a type I error exists;

– **Statistical Test Result Evaluation** - The last phase of the process corresponds to the application of the decision rule, considering the result of the statistical test, of whether or not to reject the null hypothesis. This dichotomy is actually artificial, since the significance level is frankly arbitrary, and the data can contradict the null hypothesis to a greater or lesser extent. The exact level of that extent is called the ***p*-value** - the probability of obtaining test results at least as extreme as the results actually observed, under the assumption that the null hypothesis is correct. The smaller the p-value is, the higher is the belief that the null hypothesis is contradicted, because that means that such an extreme observed outcome would be exceptionally unlikely if the null hypothesis were to be true (the probability of committing a type I error is minuscule).

The output of the series of hypothesis tests performed on the data is an array of p-values, which form the relevance table, for each class of the target vector, all the p-values corresponding to each of the features are displayed. The distributions of p-values for each class are presented next in form of histogram, both for the adhesive joint and aluminium plate with holes data.

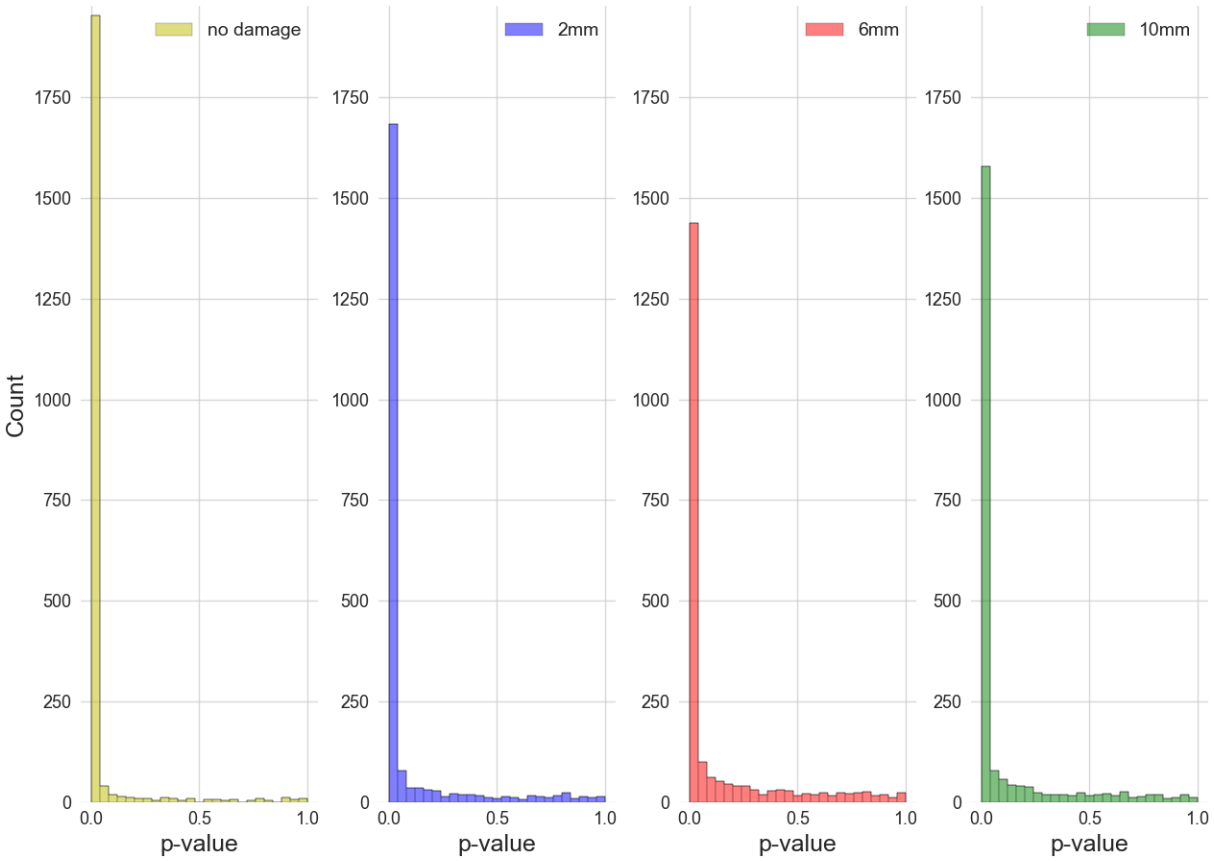


Figure 26: The p-value Histogram - Aluminium Plate Hole Size Classes

The most meaningful aspect of the histograms in Figure 26 is the first bar, corresponding to the smallest p-values (≤ 0.05). So, from an original pool of 2328 extracted

features, it can be noticed that the no damage class has the highest count of features with p-values under 0.05, with 1958, followed by the 2mm class, counting 1707 features, the 10mm class, with 1610, and finally the 6mm class, with 1467. Hence, there is an early indication that the 6mm class is the hardest to distinguish based on the available features by a slight margin.

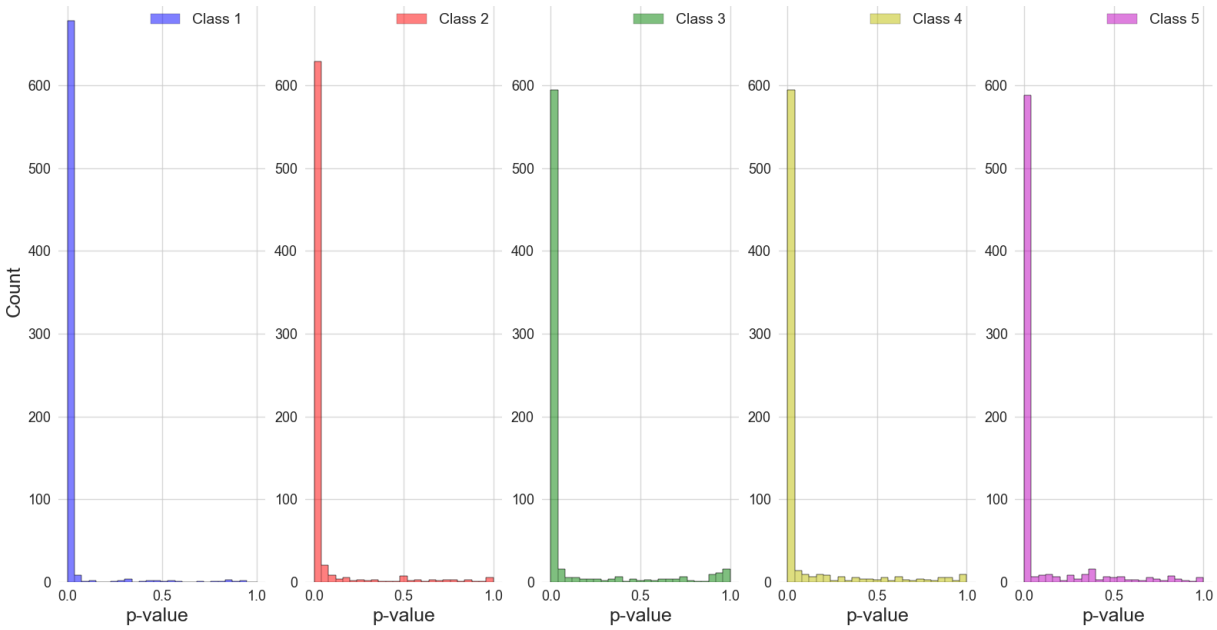


Figure 27: p-value Histogram - Adhesive Joint Strength Classes

For the adhesion strength classes, the histograms presented in Figure 27 are not only more uniform across classes, but also a larger percentage of the original 788 features were deemed statistically significant for the classification of each class, with the count of p-values ≤ 0.05 being, from class 1 to class 5, 681, 636, 599, 598, 590, respectively.

In any case, there is always the possibility of mistakes and wrong decisions being made, not only through type I errors, already discussed, but also type II errors - H_0 in not rejected, when in fact the alternative hypothesis H_1 is true [80].

But the probability of making mistakes is specially true in this testing environment, referred to as **"multiple testing"**, due to the fact that several different hypotheses are being tested simultaneously. If decisions about the individual hypotheses are based on the unadjusted marginal p-values obtained, then there is typically a large probability that some of the true null hypotheses will be rejected just from chance alone, meaning that some features will be deemed relevant, when in fact they are not [82].

For this reason, after the p-values relevance table is set, the `tsfresh` package executes the Benjamini-Hochberg Procedure, that is a multiple test procedure that decides which features to keep and which to cut off by controlling the **False Discovery Rate** (FDR), a concept formally described by Yoav Benjamini and Yosef Hochberg in 1995 [17]. To define the FDR, one starts by inspecting the possible outcomes of a generic multiple testing instance, demonstrated on Table 3:

Table 3: Possible outcomes when testing multiple null hypotheses. From [17].

	Test declared non-significant	**Test declared significant**	**Total**
True H_0	U	V	m_0
Non-true H_0	T	S	$m - m_0$
	m - R	R	m

Considering a situation where m (null) hypotheses, known in advance, are being tested, of which m_0 are true, but unknown, and $m - m_0$ is the number of true alternative hypotheses one has [17]:

- V - the number of false positives (type I error, also called "false discoveries");
- S - the number of true positives (also called "true discoveries");
- T - the number of false negatives (type II error);
- U - the number of true negatives;
- $R = V + S$ - the number of rejected null hypotheses (also called "discoveries", either true or false).

The symbol R is an observable random variable, and S, T, U, and V are unobservable random variables. The proportion of errors committed by falsely rejecting null hypotheses can be viewed through the random variable $\mathbf{Q} = V/(V + S)$, since V is the number of null hypotheses erroneously rejected. The variable is defined to be 0 if $R = 0$.

On account of the fact that both V and S are unobserved random variables, their values cannot be known, and so the False Discovery Rate is defined as being the *expectation* of $\mathbf{Q}$ [17]:

$$FDR = Q_e = E(\mathbf{Q}) = E\left(\frac{V}{V+S}\right) = E\left(\frac{V}{R}\right) \tag{23}$$

The goal of the Benjamini-Hochberg (BH) Procedure is to keep the FDR below a given threshold q. For that effect, let $p_{(1)} \leq p_{(2)} \leq ... \leq p_{(m)}$ be the ordered observed p-values. Define:

$$k = \texttt{max}\left\{i : p_{(i)} \leq \frac{i}{m}q\right\}, \tag{24}$$

and simply reject all hypotheses $H_{0i}, i = 0, 1, 2, ..., k$ for which $p_i \leq p_k$ [17]. If no such i exists, reject no hypothesis. The inequality:

$$E(\mathbf{Q}) \leq \frac{m_0}{m}q \leq q \tag{25}$$

is also verified [83].

Through this final method, the pool of extracted features considered relevant by the univariate hypothesis testing was filtered, leaving only the hereinafter called "selected features", those that have remained relevant for the classification task of at least one class. Table 4 shows the feature filtering process results.

Table 4: `tsfresh` classification-task oriented feature filtering steps results

	Plate with Holes				**Adhesive Joint**				
Classes	none	2mm	6mm	10mm	1	2	3	4	5
Extracted features	**2328**				**788**				
Features with p-value ≤ 0.05	1958	1707	1467	1610	681	636	599	598	590
Benjamini-Hochberg Selected features	1856	1539	1189	1383	657	559	555	571	571
Top features: B-H relevant for all classes	**447**				**244**				

Even though the Benjamini-Hochberg procedure reduced the feature number on all classes across the board, the most accentuated reduction is on the 6mm and 10mm hole class features for the aluminium plate classification, that had the least number of statistically significant features to start with. It should not be worrying at all, since there are plenty of features to identify all classes. The "Top" features are those considered relevant for all classes simultaneously.

To finalize this ever so important feature selection portion of the report, it is fair to mention that this automated feature selection module of `tsfresh` is just as appropriate and decisive as the feature extraction module itself. The fact that they work together so seamlessly and elegantly, allowing for the whole direct data processing, from preprocessed time series to finely selected, ML-task oriented features, made for an easy choice of implementation on this classification pipeline project. It is worthy to point out that the actual procedure implemented in the module is the Benjamini-Yekutieli procedure, in every way similar to the Benjamini-Hochberg procedure that was just described, but where the calculation of the parameter k is generalized to allow for correlation among features, as opposed to the Benjamini-Hochberg procedure where all features are assumed to be independent [83].

Having said that, if the feature selection module was not available, the `sklearn` library provides a repository of various sorts of selection techniques, some of the most popular developed to date, including univariate filter selection methods and recursive feature elimination algorithms, that could have been inserted into the pipeline at this stage. It should also be mentioned that the `tsfresh` package itself calls on `sklearn` functions, namely the statistical tests employed on the hypothesis tests, as well as borrowing the code for the

Benjamini-Hochberg Procedure from the `statsmodels` library.

4.4 Visualizing features

To end this chapter and move on to the classification of the time series based on their selected features, the techniques presented in Chapter 3 are used to visualize some of the features that resulted from the selection process. Please note that these are not being applied in a dimensionality reduction context - the features are already arrays of values, one for each time series, constituting a low dimensional data set. They are being applied in a data representation context, a way to portray the data's structure in a spatial fashion easily assimilated by the relatively untrained human eye. According to Young [84], multidimensional scaling may be applied to a very wide range of types of data. "Technically, any matrix of some raw or transformed data is a candidate for analysis by some type of multidimensional scaling method, if the elements of the data matrix indicate the strength or degree of relation between the objects or events represented by the rows and columns of the data matrix."

With that being said, the following figures represent the MDS and t-SNE plots of some of the top features, for both data sets, that will be processed by the algorithms in Chapter 5. Each point of the plots represents the calculated feature value of the corresponding test signal, ultimately representing the signal itself, hence the points color map associated with the hole size/adhesion strength of the signal.

For this purpose, the 3-dimensional mapping was chosen on both the MDS and t-SNE visualization of the features, unlike what was initially applied to the time series, as it brought no advantages in the visualization of those. These however, are snapshots of representations on a static position, where a good angle is shown, but the superlative thing about this type of representation is that it can be turned around, rotated about the three Cartesian axes without the data being changed, because the Euclidean distances are always kept, and so, there are different perspectives of the data to be appreciated with this procedure.

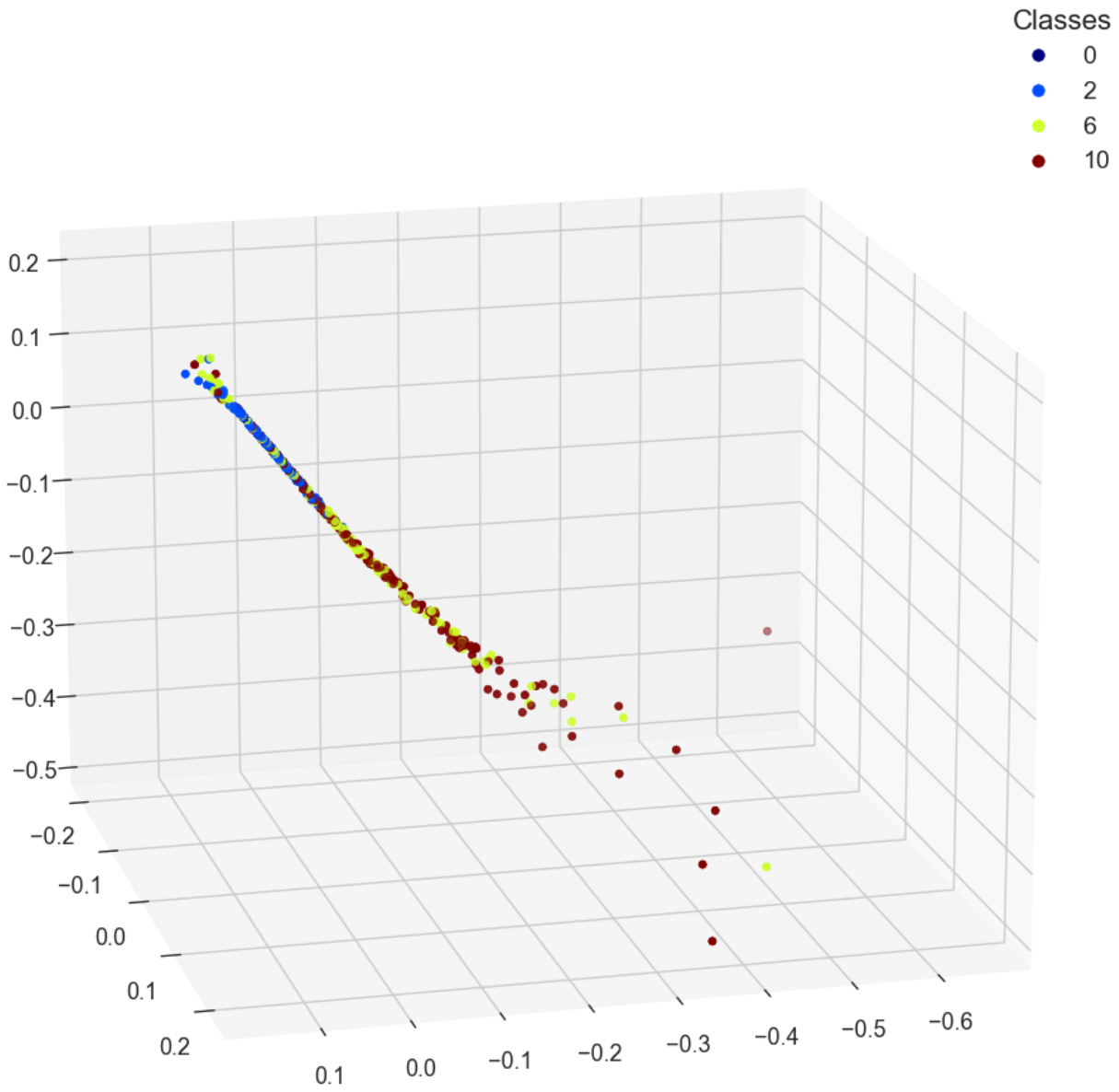


Figure 28: MDS 3-D Aluminium Plate Feature: "zt2_fft_coefficient_attr_abs_coeff_63"

Figure 28 represents a 3-D MDS mapped feature, related with the FFT values of the PZT2 signals from the aluminium plate data set. A large number of features will resemble this one, having the tests disposed in a fairly straight line, where the upper damage classes tend to diverge from the lower damage classes, but not in a perfectly separated manner. In this concrete example, the tests with 0 mm and 2 mm holes are neighboring on the top end of the line, and most of the 10 mm hole tests are placed towards the bottom, some of them dispersing quite a bit. The 6 mm hole class tests seems to appear all throughout the distribution.

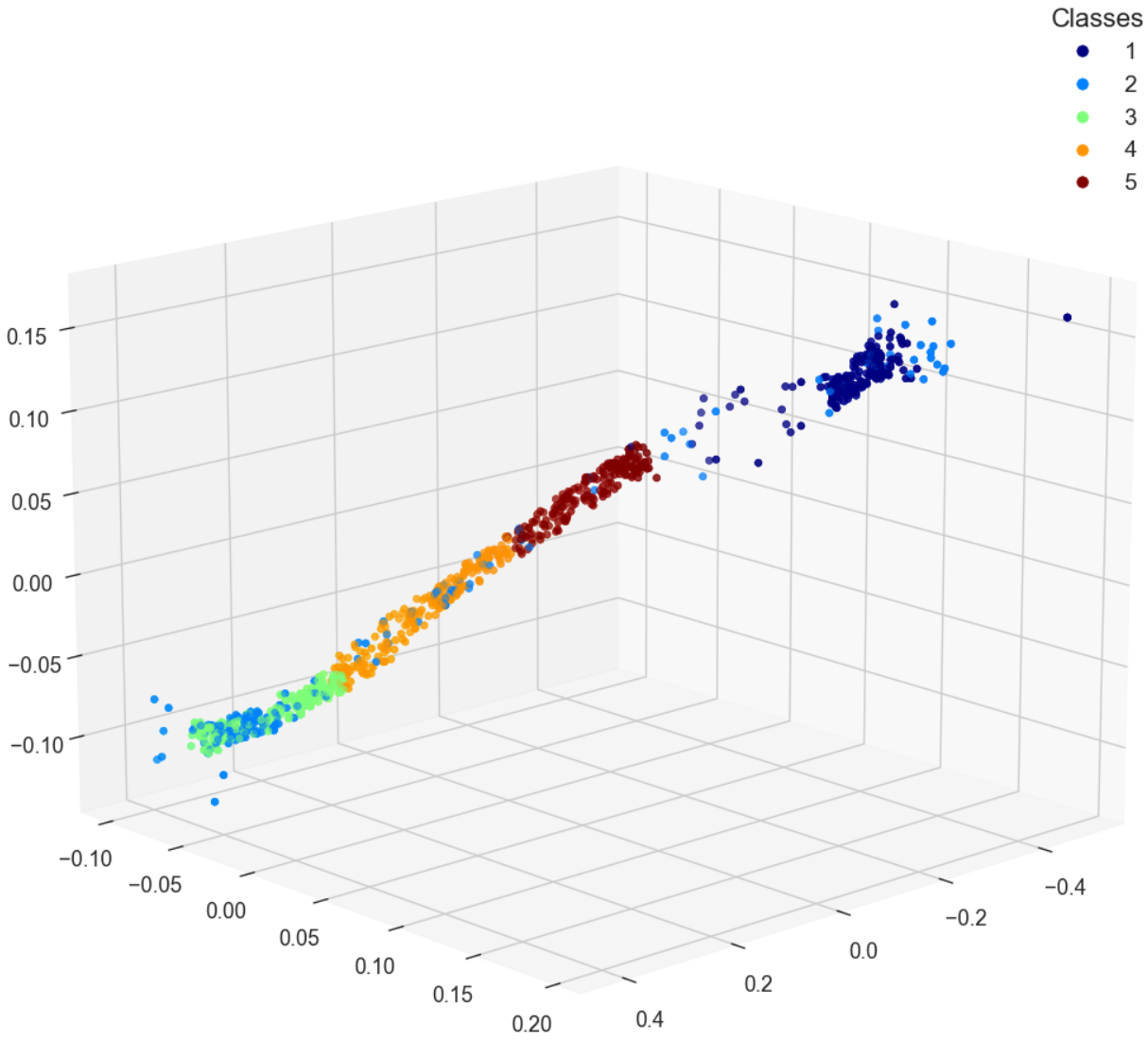


Figure 29: MDS 3-D Adhesive Joint Feature:
"value_agg_linear_trend_attr_intercept_chunk_len_10_f_agg_var"

On Figure 29, now referring to the adhesive joint data set, a more complex feature, calculated from a linear least-squares regression of the time series values, is represented in MDS mapping. The differentiation among classes is evident here, with the sole exception of class number 2 (low adhesion strength), that does not stand on its own, isolated from the others.

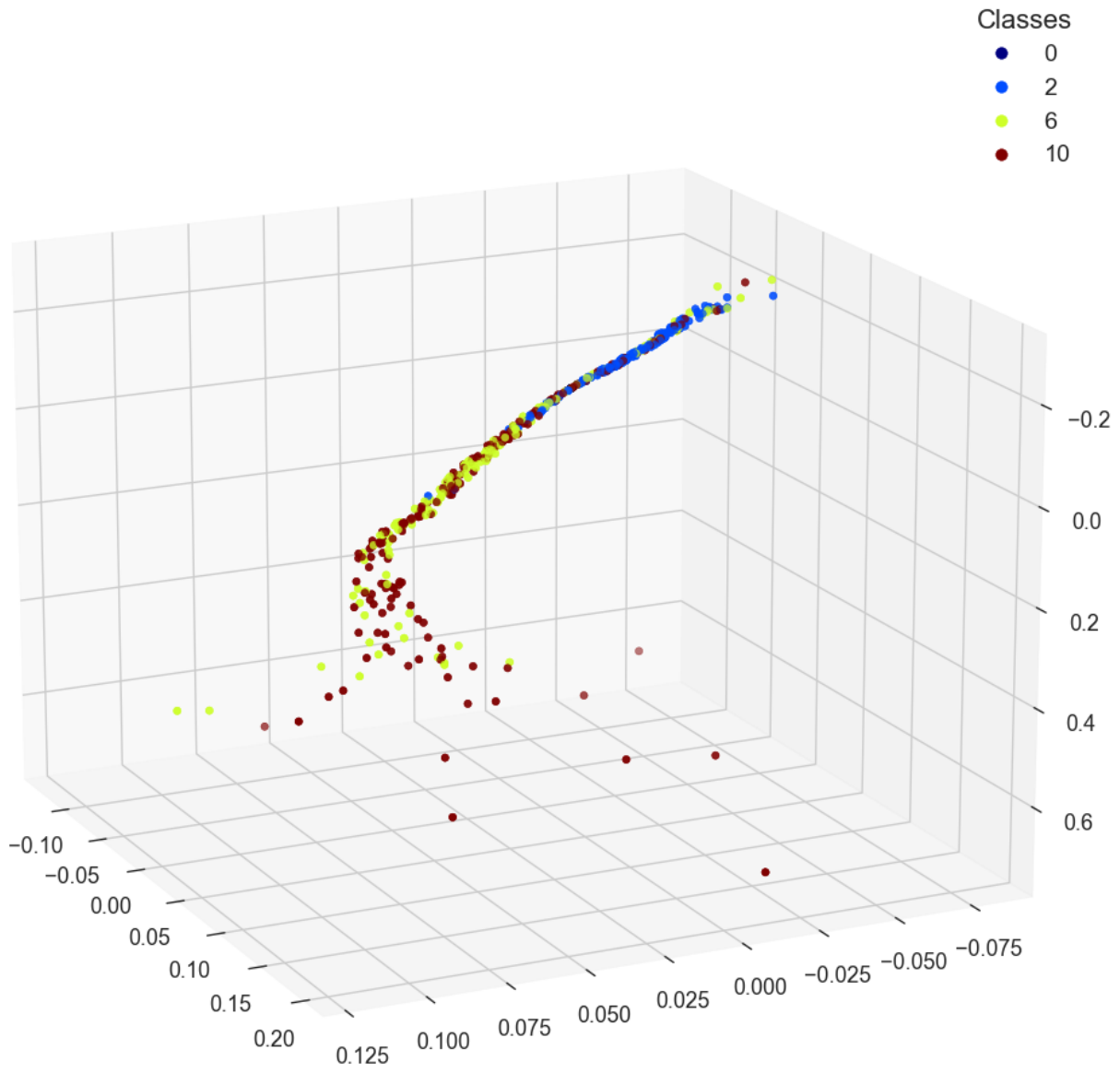


Figure 30: MDS 3-D Aluminium Plate Feature: "zt3_benford_correlation"

Lastly, Figures 30 and 31 represent the same feature from the PZT3 sensor data: the correlation values from first digit distribution when compared to the Newcomb-Benford's Law distribution, useful for anomaly detection applications. The difference is that one is mapped with the 3-D MDS technique, and the other employs the 3-D t-SNE. The MDS representation follows the same tendency that was pointed out on Figure 28, while the t-SNE displays a different arrangement, with the tests from class 0 forming a cluster, the 2 mm class tests disposed in a curved line, and the of 6 / 10 mm forming another curved line. Both mappings separate the low damage classes from the high damage classes, but do it in a different manner.

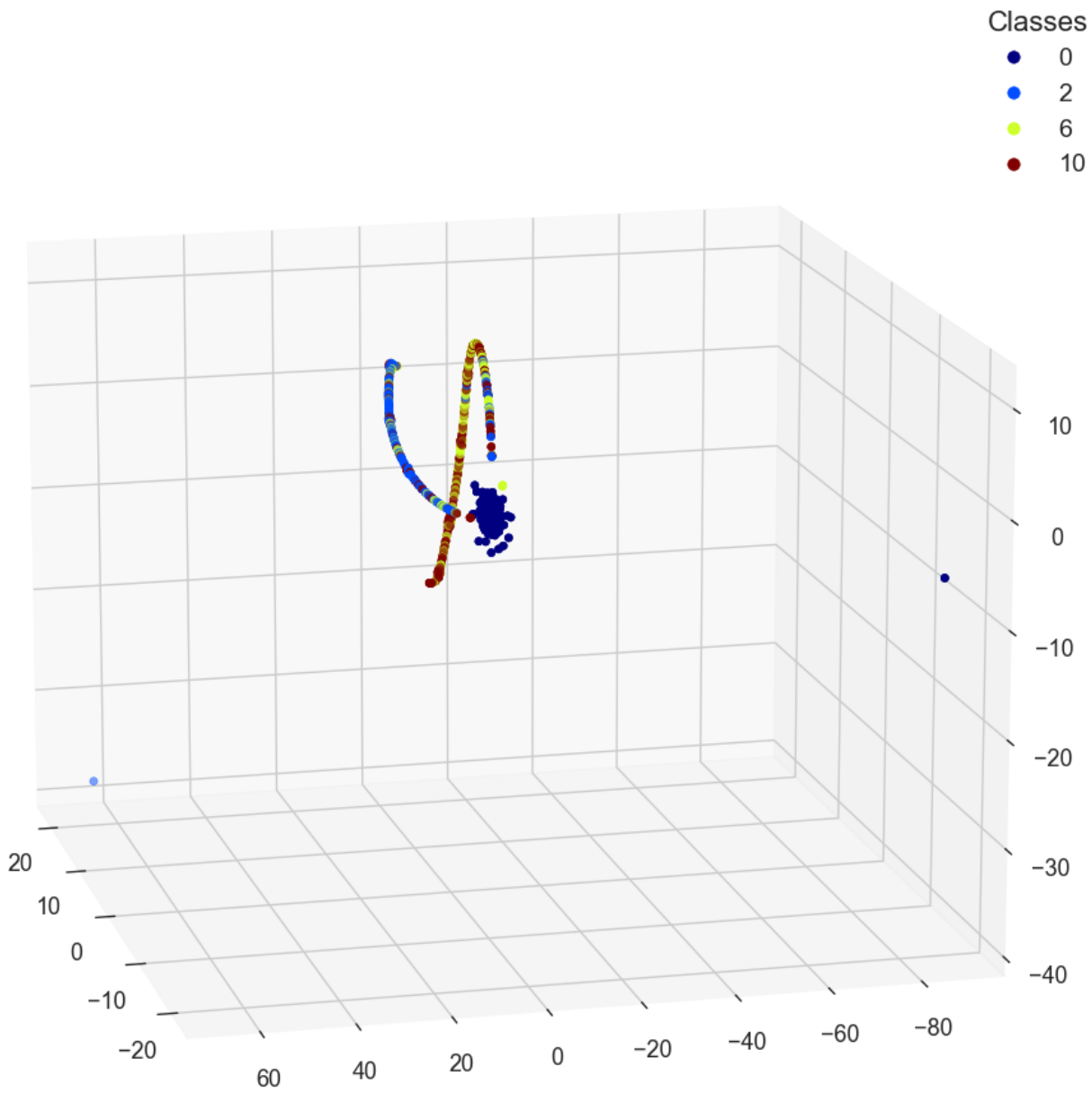


Figure 31: t-SNE 3-D Aluminium Plate Feature: "zt3_benford_correlation"

There are hundreds of top features that could be represented with these techniques, and in each one some specific differentiation among classes is perceptible, even to an untrained eye. In this light, the prospect of combining and inputting all of them into the ML algorithms becomes rather promising, as they stand to benefit from the information contained in each one to make the best possible prediction.

5 Classification Algorithms

5.1 Artificial Intelligence / Machine Learning

In the mid-twentieth century, Alan Turing originated the concept of intelligent machines, when he envisioned the possibility of machines capable of thinking. Since then, the Artificial Intelligence (AI) branch of computer science has been steadily expanding, from the creation of neural network models in 1943. From that point until the present day, the evolution of this field has been rapidly growing, following the improvements of computational power and its availability over the last two decades [85, 86].

Nowadays, AI is regarded as a general term, designating the field of computer sciences that studies "intelligent agents": any device that perceives its environment and takes actions that maximize its chance of success at some goal. It can be subdivided in various research topics, often overlapping in some aspects, that broadly encompass: ML, Neural Networks, Deep Learning, Natural Language Processing, Computer Vision, among others, and the application and evolution of spawned technologies like self-driving cars and intelligent routing in content delivery networks [86]. Numerous research articles have been published about AI in all of its facets, and each of the methods is the best suited for different applications. Nevertheless, the focus now will be put on ML, as it is a central instrument to the presented project.

ML is the subfield of AI that gives "computers the ability to learn without being explicitly programmed" [86]. Evolved from the study of pattern recognition, ML explores the construction and optimization of algorithms that can learn from, and make predictions on data. They operate by building and improving models from sample inputs, making data-driven decisions instead of following strict program instructions. It is closely related to (and often overlaps with) computational statistics, making regular use of mathematical and statistical methods.

ML algorithms are organized into four main types, based on the desired outcome of the algorithm [86]:

- **Supervised learning** - One of the most adopted ML methods, with about 70 percent employment rate, supervised learning algorithms are trained using labeled data, meaning that inputs are used where the desired output is known. The learning algorithm receives a set of inputs along with the correct outputs, and compares the generated outputs with the correct outputs, adjusting itself accordingly on the process, using the underlying patterns, to then predict the values of the label on additional unlabeled data subject to testing. If the output is a continuous value label, the procedure is called regression (not addressed in this project); if the output is a discrete number of categorical class labels, it is called classification;

- **Unsupervised learning** - Accounting for 10 to 20 percent usage rate, unsupervised learning is used on data that has no labels, so the system is not told "the

right answer", and the objective is not to explicitly find a correct answer. It is an exploratory procedure, in which the model must figure out what is being shown on its own, so the goal is to explore the data and find some structure within, ideally separating it into distinct categories (clustering);

– **Semi-supervised learning** - Semi-supervised learning is used for the same applications as supervised learning, but it uses both labeled and unlabeled data for training, meaning a small amount of labeled mixed with a large amount of unlabeled data, to be used in situations where the cost associated with labeling is too high to allow for a fully supervised program;

– **Reinforcement learning** - Often used for robotics and navigation, reinforcement learning is all about discovering which actions yield the greatest rewards through trial and error. This type of learning has three primary components: the agent (the learner or decision maker); the environment (everything the agent interacts with); and the actions (what the agent can do). The goal is for the agent to choose the actions which maximize the expected reward over a given time.

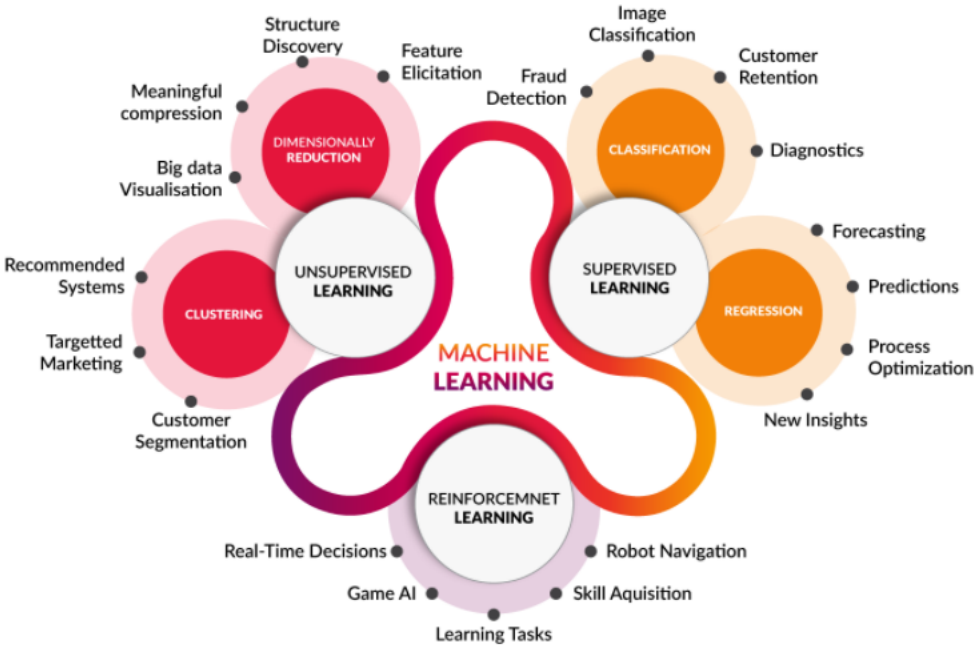


Figure 32: ML categories. From [12].

Figure 32 represents the relationship between this ML types and some possible applications of each one.

Every step of the process up until this point, as the pipeline flowchart in the beginning of Chapter 4 illustrates, is conducive to have the best damage detection and classification performance possible on supervised ML algorithms, since the data used was generated through well defined simulations, and so, obviously, all the necessary labels for each instance are known.

5.2 Supervised Machine Learning

At this stage of the project, it is time to "conclude" the classification pipeline by making predictions on damage / adhesion strength class on the respective data sets, as well confirming whether or not the feature extraction and selection that was carried out to such effect was productive.

It is now understandable why the `tsfresh` selection module, as it was mentioned in Chapter 4, required the definition of a target vector for that selection to be made, taking in consideration the classes present on that target. The target vector is nothing more than an array of classification labels corresponding to the damage class present in each time series, that the model will use to compare its results with the target and discover the patters responsible for it in order to improve the classification accuracy. That same exact target vector will be used in the training part of the algorithm.

When a classification problem has multiple output classes, it is said to be a multi-class problem, and there are two main possible strategies to approach it [16]:

- **One-vs-all** - The most common strategy, widely adopted by most algorithms, including this project. If there are n output classes, n classifiers are trained in parallel, considering a separation between an actual class and the pool of remaining ones. It is the default choice for its relatively light complexity $O(n)$, since at most $(n-1)$ checks are needed to find the right class;
- **One-vs-one** - The alternative trains a model for each pair of classes, making the complexity no longer linear $O(n^2)$, and the right class is determined by majority vote. If possible, it is preferable not to choose this strategy, unless a specific distinction between two classes is to be made.

Damage detection - classification algorithms

The time series extracted/selected features were put to the test through the use of 4 different classification algorithms:

- `Gaussian Naïve Bayes`;
- `K-Nearest Neighbors Classifier`;
- `Gradient Boosting Classifier`;
- `Random Forest Classifier`.

The code for all of these algorithms is provided by the `scikit-learn` library [64].

For each feature $\mathbf{x}$, the `Naïve Bayes` approach to classification is to formulate a probabilistic model to estimate the posterior probability $P(y|\mathbf{x})$ of the different classes y, and to predict the one with the largest posterior probability. By the Bayes' Theorem, it comes:

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})}, \tag{26}$$

where P(y) can be estimated by counting the proportion of class y in the training set, and P($\mathbf{x}$) can be ignored since we are comparing different y on the same $\mathbf{x}$; thus, the only component that needs to be considered is $P(\mathbf{x}|y)$. If an accurate estimate for $P(\mathbf{x}|y)$ is obtained, that will correspond to the **Bayes optimal classifier**, with the smallest theoretical error rate. Estimating $P(\mathbf{x}|y)$ is not straightforward, given that it involves the estimation of exponential numbers of joint-probabilities of the features. That can be avoided by assuming that in each class label, the n features are independent of each other, and so we can estimate the conditional probability by:

$$P(\mathbf{x}|y) = \prod_{i=1}^{n} P(x_i|y) \tag{27}$$

In the training stage, these probabilities are estimated, and then in the testing stage, a feature $\mathbf{x}$ will be predicted as label y, if y leads to the largest value of:

$$P(y|\mathbf{x}) \propto P(y) \prod_{i=1}^{n} P(x_i|y) \tag{28}$$

The different `Naïve Bayes` classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i|y)$, in this case, a Normal (Gaussian) distribution is appointed [13].

As for the `k-Nearest Neighbors` (KNN) algorithm, it relies on the simple principle that objects similar in the input space will also be similar in the output space. It is considered a *lazy learning* approach, known as *instance based* or *non-generalizing learning*, as it does not have an explicit training process nor does it attempt to construct a general internal model. It merely stores the instances of the training set. In training, the KNN model will pick out the k instances from the training set that are closer to the test instance, in terms of a certain distance (Euclidean, Manhattan, etc.). Then, for classification, the test instance will be classified to the majority class among the k instances, with k being a value specified by the user. The basic KNN uses uniform weights: the value assigned to a query point is computed from a simple majority vote of the nearest neighbors. Under some circumstances, it is better to weigh the neighbors such that the nearer neighbors contribute more to the fit [13, 64].

The remaining two algorithms, `Gradient Boosting` and `Random Forest` are from the same category, designated **Ensemble Methods**. Ensemble methods are state-of-the-

art ML approaches that train multiple learners to solve the same problem and combine their results, significantly improving the accuracy compared to a single learner model. Ensemble learning is also called *committee-based learning* or *multiple classifier systems*. They are constructed in two steps: generating the base models, that should be as accurate and diverse as possible, and then combining them, as shown in Figure 33 [13].

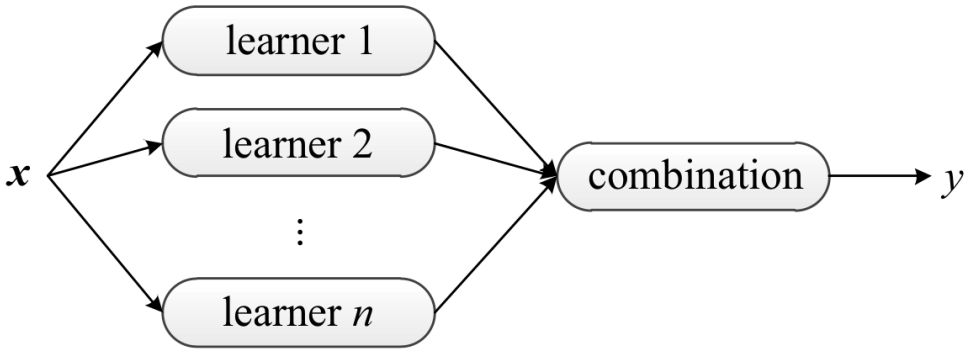


Figure 33: A common ensemble method architecture. From [13].

The ensemble contains a number of learners called **base learners**, generated from the training data by a base algorithm, which can be decision trees, neural networks or other kinds of models. All of the three ensemble methods mentioned are based on the decision tree model, and by using a single base algorithm, they produce *homogenous* base learners, leading to **homogenous ensembles**. That implies that *heterogeneous* ensembles also exist, using various learning algorithms that get the name individual/component learners, instead of base learners [13]. Before expanding on ensemble methods, defining the used base learner is in order.

A `Decision Tree` is a non-parametric model made up of tree-structured decision tests, working in a *divide-and-conquer* way, as exemplified in Figure 34, predicting the value of a target variable by learning simple decision rules inferred from the data features. It starts on the root node and from there each non-leaf node is associated with a feature test, also called a **split**. Data falling into the node will be split into subsets according to their different results on the feature test, eventually landing in a leaf node, associated with a classification label to which that instance will be assigned. `Decision Tree` learning algorithms are generally recursive processes, since in each step a data set is given and a split is selected, then acting as the given data set for the next step. The key of the decision tree algorithm is in the selection criteria of the splits. Typical ones include the "information gain" or the Gini criteria [13].

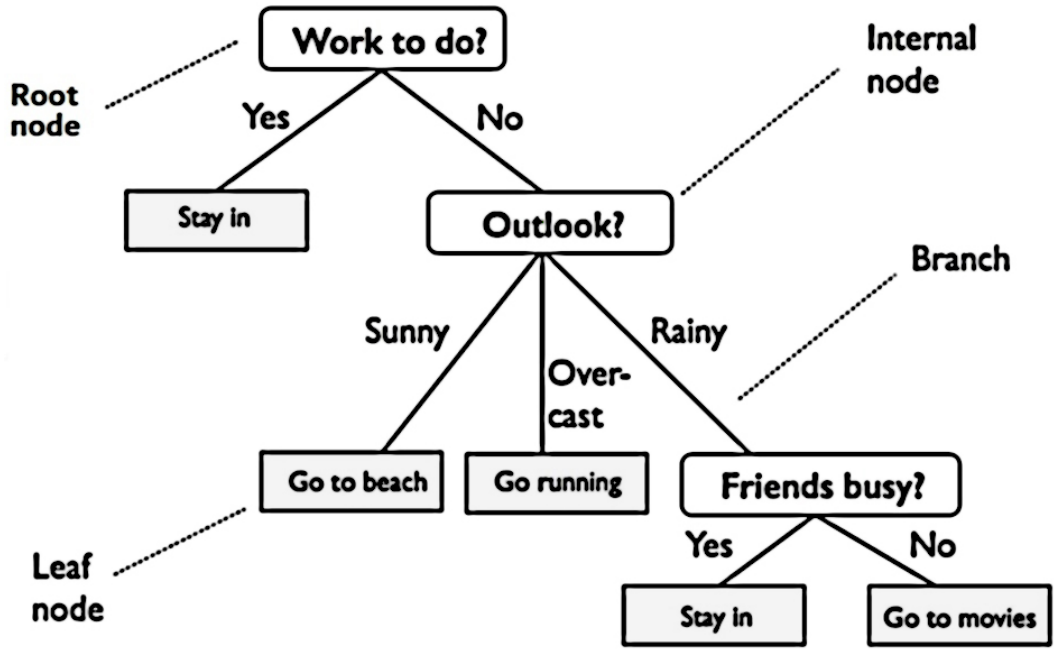


Figure 34: `Decision Tree` model example. Adapted from [14].

`Decision tree` models are prone to overfitting, by creating over-complex trees that do not generalise the data well. Mechanisms such as *pruning*, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem. `Decision tree` learners can also create biased trees if some classes dominate. It is therefore recommended to balance the data set in terms of class representation prior to fitting with the decision tree [13, 64].

The ensemble methods will combine different decision tree classifiers, improving the generalization ability of a single decision tree model. It is noteworthy that, as a general rule, the computational cost of constructing an ensemble is not much larger than the one involved when employing a single learner, so there are no serious disadvantages in selecting the former. A final distinction among ensemble methods has to do with how the base learners are generated, divided in two paradigms, roughly speaking [13]:

- **Boosting** - Refers to a family of algorithms that convert "weak learners" to "strong learners" by having the weak models working together. Characterized by being sequential ensemble methods, where the base learners are generated in succession, exploiting the dependence between them and boosting overall performance in a residual-decreasing way. Gradient Boosting is a representative of this kind;
- **Bagging** - Exploits the independence between base learners by setting them up parallel to each other, reducing the error by combining them and averaging out the results. To have independent base learners, one possibility is to apply bootstrap sampling (the bagging name derives from "bootstrap aggregating"), that means training those base models with non-overlapped subsets of the training data pool.

Not only does bagging improve performance, but also decreases variance and helps handling overfitting. Random Forest represents this kind.

According to Zhou [13] for many tasks, "the best off-the-shelf learning technique at present is an ensemble method such as `Random Forest`, facilitated with feature engineering which constructs/generates usually an overly large number of new features rather than simply working on the original features."

5.3 Classification metrics and evaluation

In ML, the algorithm's performance is presented and analysed through the results' **metrics**. Numerous metrics exist, specifically designed to evaluate certain aspects of each ML category (classification, regression, clustering, etc.), that can go from a broad assessment of the algorithms' behaviour, to very particular measurements that could be of special interest.

Having said that, prior to the models being put into practice, the data set must be split into two parts: the **training / testing sets**. In this split, a ratio must be chosen between the data to which the model will be fitted to (trained on), and the data left aside to be used in the class prediction (tested on). There is no ideal ratio, it varies with data type, data set size, and the model(s) to be used. Having said that, typical ratios used by researchers are 80/20, if the data set has a large size, or 70/30 for smaller ones. On this project, the 70% training / 30% testing ratio was adopted, as illustrated in Figure 35. It is also common to have a "validation set", a sample from the training set to perform prediction, useful to fine tune the model, check if the classifier can reproduce the known output, and also to compare against the actual testing set - if the results are fantastic on the training set, but poor on the testing set, it means the model is overfitting, for example [87].

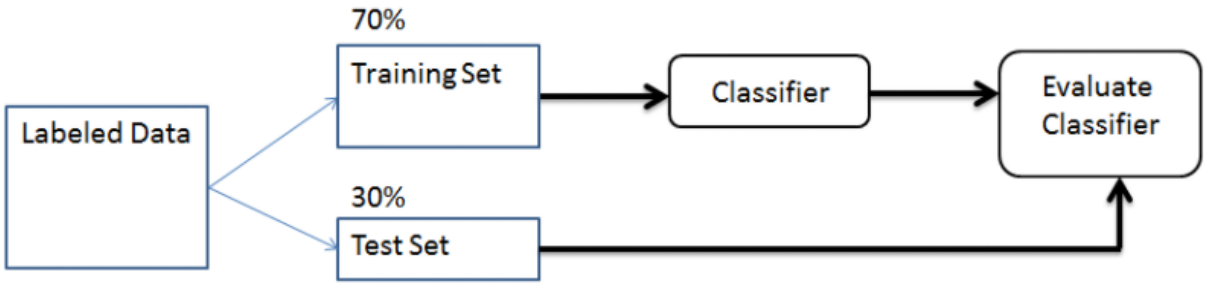


Figure 35: Supervised ML Train/Test Data Split. From [15].

While splitting, the properties of the original data set should be kept as much as possible, otherwise the validation results could be misleading (by chance of a really favorable or really poor split/sampling). To avoid that, a commonly used validation method is called k-**fold cross-validation**: the original data set is partitioned into k equal sized disjoint subsets, and then k runs of fitting/predicting are performed. For any iteration of

this process, $k-1$ subsets are randomly chosen as the training set, and the model predicts the remaining one, as illustrated on Figure 36. The average results of the k runs' metrics are taken as results of the cross-validation [13, 87]. A usual configuration is the 10-fold cross validation, adopted in this project to validate the best results, shown ahead.

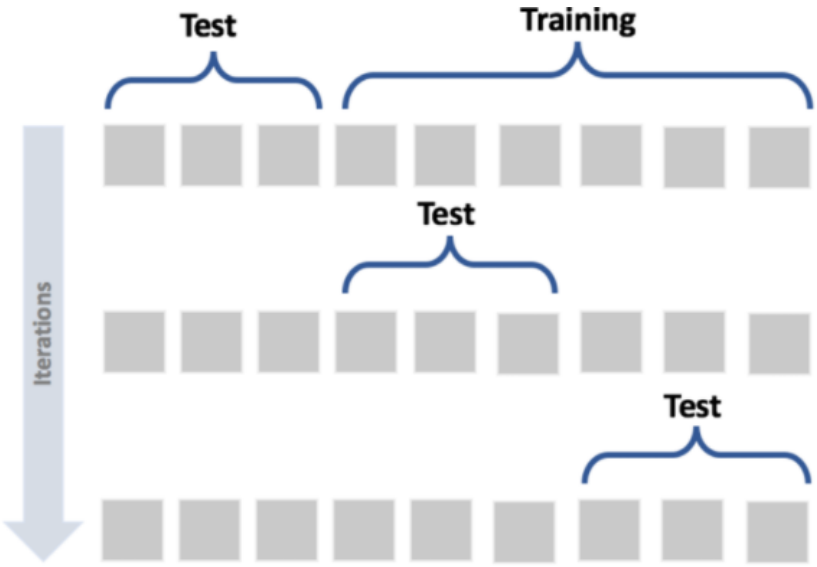


Figure 36: k-**fold cross-validation** testing iterations. From [16].

For this project, the most mainstream and widely used classification metrics were examined [64, 88]:

- **Confusion Matrix** - Also known as an error matrix, a confusion matrix is a square table layout that allows for the visualization of the performance of an algorithm. Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class. Evidently there are as many rows and columns as there are classes, and the diagonal represents the instances that were properly classified. The name stems from the fact that it makes it easy to see whether the system is confusing two classes (i.e. commonly mislabeling one as another);
- **Accuracy Score (acc)** - The accuracy metric measures the ratio of correct predictions over the total number of instances evaluated:

$$acc = \frac{tp + tn}{tp + fp + tn + fn}; \tag{29}$$

- **Precision Score (p)** - Ratio of correctly predicted positive observations to the total number of positive observations. Intuitively, it is the ability of the classifier not to label as positive a sample that is negative:

$$p = \frac{tp}{tp + fp}; \tag{30}$$

– **Recall Score (r)** - Ratio of correctly predicted positive observations to all observations in actual class. Intuitively, it is the ability of the classifier to find all the positive samples:

$$r = \frac{tp}{tp + fn}; \tag{31}$$

– **F1 Score (F1)** - Also known as balanced F-score or F-measure, is a weighted average of recall and precision, that is a parameter of great importance when having an uneven class distribution in data:

$$F1 = \frac{2 \times precision \times recall}{precision + recall}; \tag{32}$$

Despite having a fairly balanced class distribution, it is not perfectly balanced, so measuring the F1 score is valid.

The symbols *tp, fp, tn* and *fn* signify *true positive, false positive, true negative* and *false negative* respectively. All classification scores are calculated by averaging the results for each class, and using the range from 0 being the worst possible and 1 being a perfect score.

The metrics are expected not only to evaluate the algorithms' performance, but are also advantageous to assess if the feature selection process was successful, by fitting (training) each of the models with each of the data sets (plate with holes / adhesive joint), split in a way that can assess the features' impact on classification: the **full** set, containing all the successfully features initially extracted, the **selected** set, containing the features selected by the `tsfresh` selection module and the **top** set, containing the features considered relevant for all classes simultaneously. Needless to say, the score metrics are easily displayable in a table, as shown in Table 37, but when it comes to the confusion matrices, there is just no way of presenting all of them.

	0	2	6	10
0	31	0	0	0
2	0	49	0	0
6	0	21	23	3
10	0	9	40	4

Figure 37: Confusion Matrix - Plate with holes, selected feature subset - `Gaussian Naïve Bayes` classifier prediction

The confusion matrix on Figure 37, representing the `Gaussian Naïve Bayes` model predicting the classes of the plate with holes, with a selected feature data testing set, was

chosen as an example because it illustrates some interesting and diversified metrics, so that they can be understood: **class 0** has perfect accuracy, precision and recall, all tests from **class 0** were correctly classified as such, and no tests from other classes were inaccurately predicted as part of this class; **class 2** has perfect recall, as no tests from other classes were incorrectly predicted as **class 2**, but has a lower precision $p = 49/(0+49+21+9) = 0.62$, since a number of false positives appear, that is instances where tests belonging to that class were mistakenly predicted as other classes; in **class 6** and **class 10** the metrics fall down steeply, as not even the majority of the tests were predicted as their respective classes.

Table 5: Classification algorithms' metrics - Hole size prediction on aluminium plate - 70/30 data split

		Aluminium Plate Data Set		
Models		Full Features	Selected Features	Top Features
Gaussian Naïve Bayes	`acc`	0.594	0.594	0.583
	`p`	0.639	0.639	0.696
	`r`	0.641	0.641	0.618
	`F1`	0.579	0.579	0.569
k-Nearest Neighbors	`acc`	0.483	0.483	0.461
	`p`	0.533	0.533	0.507
	`r`	0.531	0.531	0.511
	`F1`	0.531	0.531	0.508
Gradient Boosting	`acc`	0.944	0.950	0.911
	`p`	0.952	0.956	0.926
	`r`	0.952	0.956	0.922
	`F1`	0.950	0.955	0.920
Random Forest	`acc`	0.900	0.939	0.895
	`p`	0.917	0.948	0.905
	`r`	0.907	0.943	0.906
	`F1`	0.908	0.944	0.905

Looking at the model results on Table 5, that present all the metrics for each model in the aluminium plate data set, it is clear that the ensemble methods, `Gradient Boosting` and `Random Forest` are head and shoulders above the `Gaussian Naïve Bayes` and `k-Nearest Neighbors` in all accounts, showing metrics consistently above 90%, which are great results, for models that were not even fine-tuned as exhaustively as they would be if they were to be fully implemented in some real-life project. The `Gaussian Naïve Bayes` and `k-Nearest Neighbors` might be suffering from the feature sets being too large which is a known disadvantage of both, specially for Bayesian models. The KNN's per-

formance could also be impaired by a poor choice of the k value, as it is a very sensitive parameter for the model [89].

Focusing on the ensemble methods, in terms of the feature subset differences, there is a steady improvement from the full to the selected set, across all metrics, which is a good sign. It means that the selection is effective in disposing of redundant and insignificant features that might be hurting the algorithms' performance. When it comes to the top features, there is a slight decline across the board as well, understandable considering that the number of significant features was reduced, and so, inevitably, relevant information was lost. Even if it was only relevant to classify one or two of the four classes, those features could - and did - make the difference in some cases. In any case, if time/computational power was a concern, this reduced number of features (447 as opposed to 2328) could be the only set initially extracted, avoiding a long and drawn-out extraction and selection process, and still yielding a more than satisfactory classification, specially because the errors will most likely occur, as Figure 38 indicates, among the classes 6 and 10, that is, those that represent meaningful damage, so a red flag would be raised in either case.

	0	2	6	10
0	31	0	0	0
2	0	49	0	0
6	0	0	39	8
10	0	0	3	50

Figure 38: Confusion Matrix - Plate with holes, selected feature subset - `Random Forest` classifier prediction

The results of the ensemble models were validated through the k-fold cross-validation method, in order to make sure they were legitimate. Table 6 presents the values for accuracy and F1 scores for both the `Gradient Boosting` and `Random Forest` models.

The metrics are solid, including one instance in each model, where 100% classification accuracy is reached. It is also very common for both of the models to have similar, almost identical metrics in the k-fold instances.

To finalize the classification procedures, the adhesive joint strength classification results are shown in Table 7, that are undoubtedly great results. The ensemble algorithms are utilizing the available features perfectly, the k-fold cross-validation shows 98-100% accuracy on all the 10 running instances, so they are pretty much as successful as one can be with this data set. More interesting is to evaluate the `Gaussian Naïve Bayes` and `k-Nearest Neighbors` models, as the first one shows a major improvement if compared

Table 6: `Gradient Boosting` and `Random Forest` k-fold cross-validation metrics (acc/F1)

Gradient Boosting		**Random Forest**	
`accuracy`	`F1 score`	`accuracy`	`F1 score`
0.967	0.964	0.933	0.939
0.950	0.957	0.967	0.975
0.950	0.956	0.950	0.958
0.983	0.982	0.983	0.982
0.967	0.971	1.00	1.00
0.967	0.961	0.983	0.984
1.00	1.00	0.900	0.891
0.967	0.975	0.983	0.980
0.983	0.984	0.983	0.982
0.933	0.946	0.967	0.973

with the aluminium plate data set. Note that the model parameters are exactly the same, so the features are definitely responsible for that improvement, either they represent the time series better, or maybe the fact that the simulations themselves represent even steps of the adhesion strength progression, it is easier to predict a given time series range. As for the KNN, it shows major improvement as well, but only with the top features subset, that contains 244 features. It is possible that some sort of "maximum features" threshold was preventing the model from scoring higher.

Be that as it may, it is fair to say that the classification tasks were a success, but there is always room for improving and strengthening of the pipeline, be it through fine tuning of the ML algorithms parameters, or further selection of the feature set employed.

Table 7: Classification algorithms' metrics - Adhesion strength prediction on single lap joint - 70/30 data split

		Adhesive Joint Data Set		
Models		Full Features	Selected Features	Top Features
Gaussian Naïve Bayes	acc	0.948	0.948	0.981
	p	0.952	0.952	0.981
	r	0.945	0.945	0.982
	F1	0.948	0.948	0.982
k-Nearest Neighbors	acc	0.552	0.552	0.982
	p	0.566	0.566	0.981
	r	0.550	0.550	0.980
	F1	0.555	0.555	0.982
Gradient Boosting	acc	0.996	1.00	0.993
	p	0.997	1.00	0.993
	r	0.996	1.00	0.993
	F1	0.996	1.00	0.993
Random Forest	acc	1.00	1.00	1.00
	p	1.00	1.00	1.00
	r	1.00	1.00	1.00
	F1	1.00	1.00	1.00

6 Summary of appended paper

In recent years the aeronautical industry has grown with new materials and fabrication methods being developed. This has caused the use of Structural Health Monitoring methods (SHM), more specifically Lamb Waves, to become more prevalent as regulations become more severe [1]. Methods utilizing time series sensor data for damage detection have shown great promise in classifying the extent of damage present in plate structures when used in union with machine learning algorithms. Despite this success, there is still a lack of a robust method for choosing features that optimize the learning process to classify any damage [2]. In this paper a powerful time-series specialized feature extraction method is implemented. Initially over 75 different prominent features and their variations are extracted from each sensor's raw data. Then, by utilizing the Benjamini-Hochberg procedure some are selected as relevant for a damage classification problem. After the initial selection, the features are inserted into supervised machine learning methods such as Random Forest and Naïve Bayes classifiers, where not only is it possible to achieve high classification metrics using all features, but also reveal and isolate which features allow the best differentiation of each damage class. This selection methodology accounts for robustness by utilizing different layers of selection and classification, validating the feature relevance in relation to the appropriate set of classes. As such, different damage types and ranges can be utilized in this multi-class classification pipeline.

7 Conclusions and Future work

A Structural Health Monitoring method, based on a Machine Learning classification pipeline, was successfully assembled, taking as inputs raw vibration signals from sensors, in a time series format, that originated from Finite Element Model simulations of a Lamb Wave propagating through a solid medium: either an aluminium plate with a defect (a hole of variable diameter and position), or a single-lap adhesive joint (with variable adhesion strength). After preprocessing, the relation between the time series was visualized through the employment of dimensionality reduction techniques.

Those time series signals were then subject to an automated feature extraction and selection procedure, where a large pool of features - attributes that characterize those signals in various domains (time, frequency, time-frequency, statistical) - were derived from the time series, and filtered through statistical methods, according to their relevance to the classification task.

Finally, those features were used to train Supervised Machine Learning algorithms, making them capable of predicting the diameter of the hole on the aluminium plate, or the approximate adhesion strength of the single-lap adhesive joint, with great accuracy.

The principal conclusions drawn from this project are:

- Lamb Waves are a very promising tool for Structural Health Monitoring/Non-Destructive Testing purposes;
- The feature extraction process was successful, capable of originating features that properly represent the sensor data in a low dimensional space;
- Feature selection is a crucial part of any Machine Learning project that involves feature engineering, as the best results will emerge from the algorithms when the best possible set of features is available;
- More training data generally means better performance of the algorithms, so the working data set should be as big as possible;
- Ensemble methods are the best off-the-shelf ML algorithms in terms of classification performance, specially if applied along with feature engineering.

Despite the project success revealed by the classification metrics, there is still an extensive margin for improvement, not only of this ML pipeline as it stands, through additional advanced feature selection and fine tuning of the implemented classification algorithms, but also through the expansion of the project, using this established framework as the cornerstone for growth, expanding the type of data used, adding other types of ML algorithms, and eventually test them with real data. Some possible approaches could be:

- The addition of noise to the signals, to approximate them to real-life sensor data;

- Alternating the excitation signal/position of the sensors in the simulations and evaluating their impact on the data and classification results;
- Using regression algorithms to predict the position of the hole on the aluminium plate, not only its diameter. Using regression to predict the exact value of the adhesion strength;
- Merging the two simulations into one: that is, a single-lap joint with variable adhesion strength, that contains a void (hole) of variable size and position. Predicting those characteristics;
- Experimental testing, with data obtained in laboratorial context, ideally preparing the pipeline for real life deployment applications.

References

[1] Lucas F.M. Da Silva, Andreas Öchsner, and Robert D. Adams. *Handbook of adhesion technology*. Springer Science & Business Media, 2011.

[2] Anthony J. Kinloch. *Adhesion and adhesives: science and technology*. Springer Science & Business Media, 2012.

[3] Mohamad Droubi, C. Fosbrooke, J. McConnachie, and Nadimul Faisal. Indentation based strength analysis of adhesively bonded leading-edge composite joints in wind turbine blades. *SN Applied Sciences*, 1:691, 2019.

[4] Robert D. Adams, Robert D. Adams, John Comyn, William Charles Wake, and W.C. Wake. *Structural adhesive joints in engineering*. Springer Science & Business Media, 1997.

[5] Firas Awaja, Michael Gilbert, Georgina Kelly, Bronwyn Fox, and Paul J. Pigram. Adhesion of polymers. *Progress in polymer science*, 34(9):948–968, 2009.

[6] Tasha Graciano. *Dynamic mechanical analysis of kissing bonds in bonded joints*. PhD thesis, The University of Wisconsin-Milwaukee, 2016.

[7] Ryan Marks, Alastair Clarke, Carol Featherston, Christophe Paget, and Rhys Pullin. Lamb wave interaction with adhesively bonded stiffeners and disbonds using 3D vibrometry. *Applied Sciences*, 6(1), 2016.

[8] Qingnan Xie, Chenyin Ni, and Zhonghua Shen. Defects detection and localization in underwater plates using laser laterally generated pure non-dispersive S_0 mode. *Applied Sciences*, 9(3), 2019.

[9] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R.J. Nelson, Eric Jones, Robert Kern, Eric Larson, C.J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[10] Vincentius Ewald, Roger Groves, and R. Benedictus. DeepSHM: A deep learning approach for structural health monitoring based on guided lamb wave techniques. 2019.

[11] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W. Kempa-Liehr. Time series feature extraction on basis of scalable hypothesis tests (tsfresh – a python package). *Neurocomputing*, 307:72–77, 2018.

[12] COGNUB Decision Solutions. `https://www.cognub.com/index.php/cognitive-platform/`, 2021. Accessed: 26-08-2021.

[13] Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman and Hall/CRC, 2019.

[14] Lorraine Li. `https://towardsdatascience.com/https-medium-com-lorrli-classification-and-regression-analysis-with-decision-trees-c43cdbc58054`, 2019. Accessed: 27-08-2021.

[15] Ahmet Taspinar. `https://ataspinar.com/2016/12/22/the-perceptron/`, 2016. Accessed: 12-08-2021.

[16] Giuseppe Bonaccorso. *Machine Learning Algorithms*. Packt Publishing Ltd, 2017.

[17] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society: Series B (Methodological)*, 57(1):289–300, 1995.

[18] Ming Hong, Zhongqing Su, Ye Lu, Hoon Sohn, and Xinlin Qing. Locating fatigue damage using temporal signal features of nonlinear lamb waves. *Mechanical Systems and Signal Processing*, 60:182–197, 2015.

[19] Z. Chen, R.D. Adams, and Lucas F.M. Da Silva. Prediction of crack initiation and propagation of adhesive lap joints using an energy failure criterion. *Engineering Fracture Mechanics*, 78(6):990–1007, 2011.

[20] David N. Alleyne. The nondestructive testing of plates using ultrasound lamb waves. 1991.

[21] Michał Mańka, Mateusz Rosiek, Adam Martowicz, Tadeusz Stepinski, and Tadeusz Uhl. Pzt based tunable interdigital transducer for lamb waves based ndt and shm. *Mechanical Systems and Signal Processing*, 78:71–83, 2016.

[22] L.B. Hunt. Bartholomaeus anglicus on gold. *Gold Bulletin*, 8:22–27, 1975.

[23] D.E. Packham, B.C. Cope, A.J. Kinloch, D.W. Aubrey, M.W. Pascoe, B. Kneafsey, D.G. Dixon, D.M. Brewis, G.W. Critchlow, D.A. Dillard, S. Millington, and G.J. Curtis. In D.E. Packham, editor, *Handbook of Adhesion*. John Wiley & Sons, Ltd, 2005.

[24] Kohzo Ito. Novel cross-linking concept of polymer network: Synthesis, structure, and properties of slide-ring gels with freely movable junctions. *Polymer Journal*, 39:489–499, 2007.

[25] Saqlain Abbas, Fucai Li, and Qiu Jianxi. A review on SHM techniques and current challenges for characteristic investigation of damage in composite material components of aviation industry. *ASTM International*, 7, 2018.

[26] A. Güemes, A. Fernandez-Lopez, A. R. Pozo, and J. Sierra-Perez. Structural health monitoring for advanced composite structures: A review. *Journal of Composites Science*, 4:13, 2020.

[27] Peter Cawley. Structural health monitoring: Closing the gap between research and industrial deployment. *Structural Health Monitoring: An International Journal*, 17, 2018.

[28] Iowa State University Center for Nondestructive Evaluation (CNDE). Nondestructive evaluation techniques. `https://www.nde-ed.org/NDETechniques/index.xhtml`, 2020. Accessed: 16-07-2021.

[29] C.C.H. Guyott, P. Cawley, and R.D. Adams. The non-destructive testing of adhesively bonded structure: A review. *The Journal of Adhesion*, 20(2):129–159, 1986.

[30] Horace Lamb. On waves in an elastic plate. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 93(648):114–128, 1917.

[31] Lord Rayleigh. On waves propagated along the plane surface of an elastic solid. *Proceedings of the London Mathematical Society*, s1-17(1):4–11, 1885.

[32] Seth Kessler and S. Spearing. Structural health monitoring in composite materials using lamb wave methods. 2001.

[33] Victor Giurgiutiu and Adrian Cuc. Embedded non-destructive evaluation for structural health monitoring, damage detection, and failure prevention. *The Shock and Vibration Digest*, 37, 2005.

[34] J.C. Dodson and D.J. Inman. Thermal sensitivity of lamb waves for structural health monitoring applications. *Ultrasonics*, 53(3):677–685, 2013.

[35] Zhongqing Su, Lin Ye, and Ye Lu. Guided lamb waves for identification of damage in composite structures: A review. *Journal of Sound and Vibration*, 295(3):753–780, 2006.

[36] Victor Giurgiutiu. Tuned lamb wave excitation and detection with piezoelectric wafer active sensors for structural health monitoring. *Journal of Intelligent Material Systems and Structures*, 16:291 – 305, 2005.

[37] Akash Desarda. `https://towardsdatascience.com/getting-data-ready-for-modelling-feature-engineering-feature-selection-dimension-reduction-77f2b9fadc0b`, 2018. Accessed: 31-07-2021.

[38] Laurens Van Der Maaten, Eric Postma, Jaap Van den Herik, et al. Dimensionality reduction: a comparative review. *Journal of Machine Learning Research*, 10(66-71):13, 2009.

[39] Yoav Benjamini and Moshe Leshno. Statistical methods for data mining. In *Data mining and knowledge discovery handbook*, pages 565–587. Springer, 2005.

[40] Guozhu Dong and Huan Liu. *Feature engineering for machine learning and data analytics*. CRC Press, 2018.

[41] Sinan Ozdemir and Divya Susarla. *Feature Engineering Made Easy: Identify unique features from your dataset in order to build powerful machine learning systems*. Packt Publishing Ltd, 2018.

[42] Izzet Tunç. `https://www.kaggle.com/izzettunc/introduction-to-time-series-clustering`, 2021. Accessed: 03-08-2021.

[43] Jason Brownlee. `https://machinelearningmastery.com/basic-feature-engineering-time-series-data-python/`, 2019. Accessed: 03-08-2021.

[44] Emile (https://stats.stackexchange.com/users/16137/emile). `https://stats.stackexchange.com/questions/50807/features-for-time-series-classification`, 2014. Accessed: 05-08-2021.

[45] Yimin Xiong and Dit-Yan Yeung. Time series clustering with arma mixtures. *Pattern Recognition*, 37(8):1675–1689, 2004.

[46] Ahmet Taspinar. `https://ataspinar.com/2018/04/04/machine-learning-with-signal-processing-techniques/`, 2018. Accessed: 08-08-2021.

[47] Dirk Jan Struik. `https://www.britannica.com/biography/Joseph-Baron-Fourier`, 2021. Accessed: 09-08-2021.

[48] Paul Heckbert. Fourier transforms and the fast fourier transform (fft) algorithm. *Computer Graphics - Carnegie Mellon University*, 2:15–463, 1995.

[49] Sanjeev R. Kulkarni. Frequency domain and fourier transforms. *Lecture Notes - Introduction to Electrical Signals and Systems*, 2002.

[50] Nasser Kehtarnavaz. Chapter 7 - frequency domain processing. In Nasser Kehtarnavaz, editor, *Digital Signal Processing System Design (Second Edition)*, pages 175–196. Academic Press, Burlington, second edition edition, 2008.

[51] Bashar Rajoub. Chapter 2 - characterization of biomedical signals: Feature engineering and extraction. In Walid Zgallai, editor, *Biomedical Signal Processing and Artificial Intelligence in Healthcare*, Developments in Biomedical Engineering and Bioelectronics, pages 29–50. Academic Press, 2020.

[52] M. Kiymik, Inan Guler, Alper Dizibuyuk, and Mehmet Akin. Comparison of STFT and Wavelet transform methods in determining epileptic seizure activity in EEG signals for real-time application. *Computers in biology and medicine*, 35:603–16, 2005.

[53] N.J. Sairamya, L. Susmitha, S. Thomas George, and M.S.P. Subathra. Chapter 12 - hybrid approach for classification of electroencephalographic signals using time–frequency images with wavelets and texture features. In D. Jude Hemanth, Deepak Gupta, and Valentina Emilia Balas, editors, *Intelligent Data Analysis for Biomedical Applications*, Intelligent Data-Centric Systems, pages 253–273. Academic Press, 2019.

[54] The Editors of Encyclopaedia Britannica. `https://pyhht.readthedocs.io/en/latest/tutorials/hilbert_view_nonlinearity.html`, 2021. Accessed: 14-08-2021.

[55] Jaidev Deshpande. `https://www.britannica.com/science/uncertainty-principle`, 2015. Accessed: 15-08-2021.

[56] M. Sifuzzaman, M. Islam, and Mostafa Ali. Application of Wavelet Transform and its advantages compared to Fourier Transform. *Journal of Physical Science*, 13, 2009.

[57] Brani Vidakovic. `https://www2.isye.gatech.edu/isyebayes/bank/handout20.pdf`, 2005. Accessed: 15-08-2021.

[58] N. Huang, Zheng Shen, S. Long, Manli C. Wu, Hsing H. Shih, Q. Zheng, N. Yen, C. Tung, and Henry H. Liu. The empirical mode decomposition and the hilbert spectrum for nonlinear and non-stationary time series analysis. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454:903 – 995, 1998.

[59] Paulo E. Rauber, Alexandre X. Falcão, Alexandru C. Telea, et al. Visualizing time-dependent data using dynamic t-SNE. 2016.

[60] J.A. Tenreiro Machado and António M. Lopes. Fractional state space analysis of temperature time series. *Fractional Calculus and Applied Analysis*, 18(6):1518–1536, 2015.

[61] Wolfgang Aigner, Silvia Miksch, Wolfgang Müller, H. Schumann, and Christian Tominski. Visualizing time-oriented data - a systematic view. *Computers & Graphics*, 31:401–409, 2007.

[62] Luai Al Shalabi and Zyad Shaaban. Normalization as a preprocessing engine for data mining and the approach of preference matrix. In *2006 International Conference on Dependability of Computer Systems*, pages 207–214, 2006.

[63] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists*. O'Reilly Media, Inc, 2018.

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

[65] Forrest W. Young and David F. Harris. *Multidimensional Scaling*. L.L. Thurstone Psychometric Laboratory, University of North Carolina, 1983.

[66] Ingwer Borg and Patrick J.F. Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[67] Gregory R. Hancock, Ralph O. Mueller, and Laura M. Stapleton. *The reviewer's guide to quantitative methods in the social sciences*. Routledge, 2010.

[68] Leland Wilkinson et al. Multidimensional scaling. *Systat 10.2 Statistics II*, pages 119–145, 2002.

[69] Seyed Abolghasem Mohammadi and B.M. Prasanna. Analysis of genetic diversity in crop plants—salient statistical tools and considerations. *Crop science*, 43(4):1235–1248, 2003.

[70] J.A. Tenreiro Machado and António M. Lopes. On the mathematical modeling of soccer dynamics. *Communications in Nonlinear Science and Numerical Simulation*, 53:142–153, 2017.

[71] Geoffrey Hinton and Sam T. Roweis. Stochastic neighbor embedding. In *NIPS*, volume 15, pages 833–840. Citeseer, 2002.

[72] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11), 2008.

[73] Mehedi Hasan. `https://www.ubuntupit.com/best-python-libraries-and-packages-for-beginners/`, 2019. Accessed: 22-08-2021.

[74] J.D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[75] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

[76] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

[77] Marília Barandas, Duarte Folgado, Letícia Fernandes, Sara Santos, Mariana Abreu, Patrícia Bota, Hui Liu, Tanja Schultz, and Hugo Gamboa. TSFEL: Time Series Feature Extraction Library. *SoftwareX*, 11:100456, 2020.

[78] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W. Kempa-Liehr. `https://tsfresh.readthedocs.io/en/latest/index.html`, 2021. Accessed: 23-08-2021.

[79] Xue Ying. An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, 1168:022022, 2019.

[80] Rui Campos Guimarães and J.A. Sarsfield Cabral. *Estatística, 2ª Edição*. Verlag Dashöfer, 2010.

[81] Lisa Sullivan. `https://sphweb.bumc.bu.edu/otlt/mph-modules/bs/bs704_hypothesistest-means-proportions/bs704_hypothesistest-means-proportions_print.html`, 2017. Accessed: 25-08-2021.

[82] Joseph P. Romano, Azeem M. Shaikh, Michael Wolf, et al. Multiple testing. *The New Palgrave Dictionary of Economics*, 2010.

[83] Yoav Benjamini and Daniel Yekutieli. The control of the false discovery rate in multiple testing under dependency. *Annals of statistics*, pages 1165–1188, 2001.

[84] Forrest W. Young and Robert M. Hamer. *Multidimensional Scaling: History, Theory, and Applications*. Psychology Press, 2013.

[85] Mohssen Mohammed, Muhammad Badruddin Khan, and Eihab Bashier Mohammed Bashier. *Machine Learning: Algorithms and Applications*. CRC Press, 2016.

[86] Pariwat Ongsulee. Artificial intelligence, machine learning and deep learning. In *2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE)*, pages 1–6. IEEE, 2017.

[87] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[88] Mohammad Hossin and M. Nasir Sulaiman. A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, 5(2):1, 2015.

[89] Amanpreet Singh, Narina Thakur, and Aakanksha Sharma. A review of supervised machine learning algorithms. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 1310–1315. IEEE, 2016.

Appendices

A `tsfresh` List of Feature Calculators

`tsfresh.feature_extraction.feature_calculators` module	
`feature_calculator`	Designation
abs_energy(x)	Returns the absolute energy of the time series which is the sum over the squared values.
absolute_maximum(x)	Calculates the highest absolute value of the time series x.
absolute_sum_of_changes(x)	Returns the sum over the absolute value of consecutive changes in the series x.
agg_autocorrelation(x, param)	Descriptive statistics on the autocorrelation of the time series.
agg_linear_trend(x, param)	Calculates a linear least-squares regression for values of the time series that were aggregated over chunks versus the sequence from 0 up to the number of chunks minus one.
approximate_entropy(x, m, r)	Implements a vectorized Approximate entropy algorithm.
ar_coefficient(x, param)	This feature calculator fits the unconditional maximum likelihood of an autoregressive AR(k) process. The k parameter is the maximum lag of the process.
augmented_dickey_fuller(x, param)	Does the time series have a unit root? The Augmented Dickey-Fuller test is a hypothesis test which checks whether a unit root is present in a time series sample. This feature calculator returns the value of the respective test statistic.
autocorrelation(x, lag)	Calculates the autocorrelation of the specified lag.
benford_correlation(x)	Useful for anomaly detection applications. Returns the correlation from first digit distribution when compared to the Newcomb-Benford's Law distribution.
binned_entropy(x, max_bins)	First bins the values of x into max_bins equidistant bins.
c3(x, lag)	Uses c3 statistics to measure non linearity in the time series.

change_quantiles(x, ql, qh, isabs, f_agg)	First fixes a corridor given by the quantiles ql and qh of the distribution of x.
cid_ce(x, normalize)	This function calculator is an estimate for a time series complexity (a more complex time series has more peaks, valleys etc.).
count_above(x, t)	Returns the percentage of values in x that are higher than t.
count_above_mean(x)	Returns the number of values in x that are higher than the mean of x.
count_below(x)	Returns the percentage of values in x that are lower than t.
count_below_mean(x)	Returns the number of values in x that are lower than the mean of x.
cwt_coefficients(x, param)	Calculates a Continuous Wavelet Transform for the Ricker wavelet family. This feature calculator takes three different parameter: widths, coeff and w. The feature calculator takes all the different widths arrays and then calculates the cwt one time for each different width array. Then the values for the different coefficient for coeff and width w are returned.
energy_ratio_by_chunks(x, param)	Calculates the sum of squares of chunk i out of N chunks expressed as a ratio with the sum of squares over the whole series.
fft_aggregated(x, param)	Returns the spectral centroid (mean), variance, skew, and kurtosis of the absolute Fourier transform spectrum.
fft_coefficient(x, param)	Calculates the Fourier coefficients of the one-dimensional discrete Fourier Transform for real input by Fast Fourier Transformation algorithm. The resulting coefficients will be complex, this feature calculator can return the real part (attr==”real”), the imaginary part (attr==”imag), the absolute value (attr=””abs) and the angle in degrees (attr==”angle).
first_location_of_maximum(x)	Returns the first location of the maximum value of x, relatively to the length of x.
first_location_of_minimum(x)	Returns the first location of the minimal value of x, relatively to the length of x.

fourier_entropy(x, bins)	Calculate the binned entropy of the power spectral density of the time series (using the welch method).
friedrich_coefficients(x, param)	Coefficients of polynomial h(x), which has been fitted to the deterministic dynamics of Langevin model.
has_duplicate(x)	Checks if any value in x occurs more than once.
has_duplicate_max(x)	Checks if the maximum value of x is observed more than once.
has_duplicate_min(x)	Checks if the minimal value of x is observed more than once.
index_mass_quantile(x)	Calculates the relative index i of time series x where q% of the mass of x lies left of i. For example for q = 50% this feature calculator will return the mass center of the time series.
kurtosis(x)	Returns the kurtosis of x (calculated with the adjusted Fisher-Pearson standardized moment coefficient G2).
large_standard_deviation(x, r)	Does time series have large standard deviation? Boolean variable denoting if the standard dev of x is higher than 'r' times the range = difference between max and min of x. According to a rule of the thumb, the standard deviation should be a forth of the range of the values.
last_location_of_maximum(x)	Returns the relative last location of the maximum value of x. The position is calculated relatively to the length of x.
last_location_of_minimum(x)	Returns the last location of the minimal value of x. The position is calculated relatively to the length of x.
lempel_ziv_complexity(x, bins)	Calculate a complexity estimate based on the Lempel-Ziv compression algorithm.
length(x)	Returns the length of x.

linear_trend(x, param)	Calculate a linear least-squares regression for the values of the time series versus the sequence from 0 to length of the time series minus one. This feature assumes the signal to be uniformly sampled. It will not use the time stamps to fit the model. The parameters control which of the characteristics are returned.
linear_trend_timewise(x, param)	Calculate a linear least-squares regression for the values of the time series versus the sequence from 0 to length of the time series minus one. This feature uses the index of the time series to fit the model, which must be of a `datetime dtype`. The parameters control which of the characteristics are returned.
longest_strike_above_mean(x)	Returns the length of the longest consecutive subsequence in x that is bigger than the mean of x.
longest_strike_below_mean(x)	Returns the length of the longest consecutive subsequence in x that is smaller than the mean of x.
matrix_profile(x, param)	Calculates the 1-D Matrix Profile[1] and returns Tukey's Five Number Set plus the mean of that Matrix Profile.
max_langevin_fixed_point(x, r, m)	Largest fixed point of dynamics estimated from polynomial h(x), which has been fitted to the deterministic dynamics of Langevin mode.
maximum(x)	Calculates the highest value of the time series x.
mean(x)	Returns the mean of x.
mean_abs_change(x)	Average over first differences. Returns the mean over the absolute differences between subsequent time series values.
mean_change(x)	Average over time series differences. Returns the mean over the differences between subsequent time series values which.
mean_n_absolute_max(x,number_of_maxima)	Calculates the arithmetic mean of the n absolute maximum values of the time series.

mean_second_derivative_central(x)	Returns the mean value of a central approximation of the second derivative.
median(x)	Returns the median of x.
minimum(x)	Calculates the lowest value of the time series x.
number_crossing_m(x, m)	Calculates the number of crossings of x on m. A crossing is defined as two sequential values where the first value is lower than m and the next is greater, or vice-versa. If you set m to zero, you will get the number of zero crossings.
number_cwt_peaks(x, n)	Number of different peaks in x. To estimamte the numbers of peaks, x is smoothed by a Ricker Wavelet for widths ranging from 1 to n. This feature calculator returns the number of peaks that occur at enough width scales and with sufficiently high Signal-to-Noise-Ratio (SNR)
number_peaks(x, n)	Calculates the number of peaks of at least support n in the time series x. A peak of support n is defined as a subsequence of x where a value occurs, which is bigger than its n neighbours to the left and to the right.
partial_autocorrelation(x, param)	Calculates the value of the partial autocorrelation function at the given lag.
percentage_of_reocurring_datapoints_ _to_all_datapoints(x)	Returns the percentage of non-unique data points. Non-unique means that they are contained another time in the time series again. The ratio is normalized to the number of data points in the time series.
percentage_of_reocurring_values_ _to_all_values(x)	Returns the percentage of values that are present in the time series more than once. The percentage is normalized to the number of unique values.

permutation_entropy(x, tau, dimension)	Calculate the permutation entropy, by chunking the data into sub-windows of length D starting every tau, replacing each D-window by the permutation that captures the ordinal ranking of the data, and counting the frequencies of every permutation, returning their entropy (by using log_e and not log_2).
quantile(x, q)	Calculates the q quantile of x. This is the value of x greater than q% of the ordered values from x.
query_similarity_count(x, param)	This feature calculator accepts an input query subsequence parameter, compares the query (under z-normalized Euclidean distance) to all subsequences within the time series, and returns a count of the number of times the query was found in the time series (within some predefined maximum distance threshold).
range_count(x, min, max)	Count observed values within the interval [min, max).
ratio_beyond_r_sigma(x, r)	Ratio of values that are more than r * std(x) (r times sigma) away from the mean of x.
ratio_value_number_to_time_series_lenght(x)	Returns a factor which is 1 if all values in the time series occur only once, and below one if this is not the case.
root_mean_square(x)	Returns the root mean square (rms) of the time series.
sample_entropy(x)	Calculate and return sample entropy of x.
set_property(key, values)	This method returns a decorator that sets the property key of the function to value.
skewness(x)	Returns the sample skewness of x (calculated with the adjusted Fisher-Pearson standardized moment coefficient G1).
spkt_welch_density(x, param)	This feature calculator estimates the cross power spectral density of the time series x at different frequencies. To do so, the time series is first shifted from the time domain to the frequency domain. Returns the power spectrum of the different frequencies.
standard_deviation(x)	Returns the standard deviation of x.

sum_of_reocurring_data_points(x)	Returns the sum of all data points, that are present in the time series more than once. Every reocurring value is counted as many times as it appears.
sum_of_reocurring_values(x)	Returns the sum of all values, that are present in the time series more than once. Each reocurring value is only counted once.
sum_values(x)	Calculates the sum over the time series values.
symmetry_looking(x, param)	Boolean variable denoting if the distribution of x looks symmetric.
time_reversal_asymmetry_statistic(x, lag)	Returns the time reversal asymmetry statistic.
value_count(x, value)	Count occurrences of value in time series x.
variance(x)	Returns the variance of x.
variance_larger_than_standard_deviation(x)	Is variance higher than the standard deviation? Boolean variable denoting if the variance of x is greater than its standard deviation. Is equal to variance of x being larger than 1.
variation_coefficient(x)	Returns the variation coefficient (standard error / mean, give relative value of variation around mean) of x.

For a more complete description of each procedure, please refer to the `tsfresh` documentation website (https://tsfresh.com).

B Python Code Script

```
##### General Libraries #####

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pandas as pd
import glob
import itertools

from scipy import interpolate
from scipy import signal
from scipy.fftpack import fft, rfft, fftshift, fftfreq, rfftfreq
from scipy.signal import find_peaks

import tsfresh
from tsfresh import extract_features, select_features

from sklearn.manifold import MDS, TSNE
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split, KFold
from sklearn.ensemble import (RandomForestClassifier,
                              GradientBoostingClassifier)
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

plt.style.use('seaborn')
plot_params = {'axes.facecolor': 'white',
               'axes.grid': True,
               'grid.alpha': 0.75,
               'grid.color': "#cccccc",
               'legend.loc': "upper right",
               'legend.title_fontsize': 16,
               'axes.titlesize': 20,
               'axes.labelsize': 18,
               'legend.fontsize': 14,
               'xtick.labelsize': 13,
               'ytick.labelsize': 13}
plt.rcParams.update(plot_params)


##### Functions #####

def download_data (path):

    all_files = glob.glob(path + "/*.csv")
    li = []

    headerNames = np.arange(0, 90000, 1)
    for filename in all_files:
        df = pd.read_csv(filename, index_col=None, sep=',' ,
                         header= None, na_filter=True, engine= 'python',
                         names=list(headerNames))
        li.append(df)

    dataset=pd.concat(li, axis=0, ignore_index=True)

    return (dataset)

def create_data_timeseries(dataset):

    dataset=np.array(dataset)
    hole_information=dataset[:,0:4]
    sensor_information=dataset[:,4:]

    return(hole_information, sensor_information)
```

```
def downsample_data_excel(dt,tempo_total,tempo_desejado,input_information):

    input_without_Nan=[]
    for i in range(0,len(input_information)):
        input_without_Nan.append(input_information[i][np.logical_not(np.isnan(input_information[i]))])

    data_interpolated=[]
    for i in range(0,len( input_without_Nan),4):
        for j in range(0,4):
            f = interpolate.interp1d(input_without_Nan[i],
                                     input_without_Nan[i+j],
                                     kind='cubic',
                                     fill_value="extrapolate")
            data_interpolated.append(f(tempo_desejado))

    return (data_interpolated)

def downsample_data_holes(dt,tempo_total,tempo_desejado,input_information):

    input_without_Nan=[]
    for i in range(0,len(input_information)):
        input_without_Nan.append(input_information[i][np.logical_not(np.isnan(input_information[i]))])

    input_without_Nan=np.array(input_without_Nan)

    inputs_without_repetition=[]
    for i in range(0,len(input_without_Nan)-4,4):
        x3, ind = np.unique(input_without_Nan[i], return_index = True)
        inputs_without_repetition.append(input_without_Nan[i][ind])
        x4 = input_without_Nan[i+1][ind]
        x5 = input_without_Nan[i+2][ind]
        x6 = input_without_Nan[i+3][ind]

        inputs_without_repetition.append(x4)
        inputs_without_repetition.append(x5)
        inputs_without_repetition.append(x6)

    novo=[]
    for i in range(0,len(inputs_without_repetition),4):
        for j in range(1,4):
            f = interpolate.interp1d(inputs_without_repetition[i],
                                     inputs_without_repetition[i+j],
                                     kind='cubic',
                                     fill_value="extrapolate")
            novo.append(f(tempo_desejado))

    return (novo)

def downsample_data_adhesives(dt,tempo_total,tempo_desejado,input_information):

    input_without_Nan=[]
    for i in range(0,len(input_information)):
        input_without_Nan.append(input_information[i][np.logical_not(np.isnan(input_information[i]))])


    input_without_Nan=np.array(input_without_Nan)

    inputs_without_repetition=[]
    for i in range(0,len(input_without_Nan)-1,2):
        x3, ind = np.unique(input_without_Nan[i], return_index = True)
        inputs_without_repetition.append(input_without_Nan[i][ind])
        x4 = input_without_Nan[i+1][ind]
        inputs_without_repetition.append(x4)

    novo=[]
    for i in range(0,len(inputs_without_repetition),2):
        f = interpolate.interp1d(inputs_without_repetition[i],
                                 inputs_without_repetition[i+1],
                                 kind='cubic',
                                 fill_value="extrapolate")
        novo.append(f(tempo_desejado))

    return (novo)
```

```
def create_ts_dataframe (input_information):

    name_array = ["pzt1", "pzt2", "pzt3"] * 600 # must be equal to the number of tests
    tests = list(itertools.chain.from_iterable([[x] * 3 for x in np.arange(0,600)]))
    test_list = list(zip(*[tests, name_array]))
    test_multindex = pd.DataFrame(input_information)
    sensor_dataframe = test_multindex.set_index(pd.MultiIndex.from_tuples
                                                (test_list, names=["test", "series"]))

    return (sensor_dataframe)

def get_fft_values(y_values, N, T, t):

    fft_amplitude = rfft(y_values)
    yf = np.abs(fft_amplitude)
    xf = rfftfreq(N,T)

    Y = fftshift(fft(y_values))
    #f = fftshift(fftfreq(len(t)))
    f = fftshift(fftfreq(N,T))

    p = np.unwrap(np.angle(Y))

    return xf, yf, f, p

def mds_data_visualization_2D(input_information):

    X = input_information
    scaler = MinMaxScaler()
    X_scaled = scaler.fit_transform(X)

    mds_2D = MDS(n_components = 2, random_state=0)
    X_2D = mds_2D.fit_transform(X_scaled)
    #X_2D_not_scaled = mds.fit_transform(X)

    return (X_2D)

def mds_data_visualization_3D(input_information):

    X = input_information
    scaler = MinMaxScaler()
    X_scaled = scaler.fit_transform(X)

    mds_3D = MDS(n_components = 3, random_state=0)
    X_3D = mds_3D.fit_transform(X_scaled)

    return (X_3D)

def tsne_data_visualization_2D(input_information):

    X = input_information
    scaler = MinMaxScaler()
    X_scaled = scaler.fit_transform(X)

    tsne = TSNE(n_components = 2, random_state=0)
    X_2D = tsne.fit_transform(X_scaled)

    return (X_2D)

def tsne_data_visualization_3D(input_information):

    X = input_information
    scaler = MinMaxScaler()
    X_scaled = scaler.fit_transform(X)

    tsne = TSNE(n_components = 3, random_state=0)
    X_3D = tsne.fit_transform(X_scaled)

    return (X_3D)
```

```
##### Program Body #####

### DATA/VARIABLES PREPARATION ###


# Data Gathering
path = r'C:\Users\vlore\Desktop\Workbox\5º Ano\Tese\TESTDATA\PYTHON_PATH_TESTS'
raw_data = download_data(path)
hole_info, test_sensor_data = create_data_timeseries(raw_data)
adhesion_info = hole_info # same info placement, depending on the input data
                          # (holes/adhesives) placed in the path folder

# Save DataFrame to .csv
#  .to_csv(r'C:\Users\vlore\Desktop\Workbox\5º Ano\Tese\TESTDATA\raw_data.csv', header = False)

# Interpolation Specs
dt=10**-7
total_time=0.0005
desired_time = np.arange(0, total_time, dt)
t = desired_time
# Frequency Analysis / FFT Values
sampfreq = 1 / dt
N = int(sampfreq * total_time)
T = 1 / sampfreq

# Data Downsampled & Dataframed
# Holes
input_data_downsampled = (downsample_data_holes(dt,total_time,desired_time,test_sensor_data))
input_data_downsampled = input_data_downsampled[0:1800] # only the first 1800 tests were used
                                                        # for feature extraction , 600 for each sensor
ts_dataframe = create_ts_dataframe(input_data_downsampled)
y_filtered = (np.array(ts_dataframe.iloc[0,:]),
              np.array(ts_dataframe.iloc[1,:]),
              np.array(ts_dataframe.iloc[2,:]))
# Adhesives
input_data_downsampled = (downsample_data_adhesives(dt,total_time,desired_time,test_sensor_data))
ts_dataframe = pd.DataFrame(input_data_downsampled)
y_filtered = (np.array(ts_dataframe.iloc[2,:]),
              np.array(ts_dataframe.iloc[333,:]),
              np.array(ts_dataframe.iloc[650,:]))



### PART 1 - SIGNAL VISUALIZATION AND MANUAL FEATURE EXTRACTION ###

# For holes, it gets the data of each of the 3 sensors, for each test.
# For adhesives, it gets the single sensor data, for 3 tests.


# Time Series Peaks
time_peaks_index = (find_peaks(y_filtered[0])[0],
                    find_peaks(y_filtered[1])[0],
                    find_peaks(y_filtered[2])[0])
time_peaks_values = (y_filtered[0][time_peaks_index[0]],
                     y_filtered[1][time_peaks_index[1]],
                     y_filtered[2][time_peaks_index[2]])
max_time_peaks_values = (time_peaks_values[0][np.argsort(time_peaks_values[0])[-10:]],
                         time_peaks_values[1][np.argsort(time_peaks_values[1])[-10:]],
                         time_peaks_values[2][np.argsort(time_peaks_values[2])[-10:]])
max_time_peaks_index = ([x[0] for x in [np.where(y_filtered[0]==x)[0] for x in
max_time_peaks_values[0]]],
                        [x[0] for x in [np.where(y_filtered[1]==x)[0] for x in
max_time_peaks_values[1]]],
                        [x[0] for x in [np.where(y_filtered[2]==x)[0] for x in
max_time_peaks_values[2]]])
max_time_peaks_time = (t[max_time_peaks_index[0]],
                       t[max_time_peaks_index[1]],
                       t[max_time_peaks_index[2]])
# Fast Fourier Transform (FFT)
fft_values = (get_fft_values(y_filtered[0], N, T, t),
              get_fft_values(y_filtered[1], N, T, t),
              get_fft_values(y_filtered[2], N, T, t))
```

```
# FFT Peaks
fft_peaks_index = (find_peaks(fft_values[0][1])[0],
                   find_peaks(fft_values[1][1])[0],
                   find_peaks(fft_values[2][1])[0])
fft_peaks_values = (fft_values[0][1][fft_peaks_index[0]],
                    fft_values[1][1][fft_peaks_index[1]],
                    fft_values[2][1][fft_peaks_index[2]])
max_fft_peaks_values = (fft_peaks_values[0][np.argsort(fft_peaks_values[0])[-5:]],
                        fft_peaks_values[1][np.argsort(fft_peaks_values[1])[-5:]],
                        fft_peaks_values[2][np.argsort(fft_peaks_values[2])[-5:]])
max_fft_peaks_index = ([x[0] for x in [np.where(fft_values[0][1]==x)[0] for x in
max_fft_peaks_values[0]]],
                       [x[0] for x in [np.where(fft_values[1][1]==x)[0] for x in
max_fft_peaks_values[1]]],
                       [x[0] for x in [np.where(fft_values[2][1]==x)[0] for x in
max_fft_peaks_values[2]]])
max_fft_peaks_time =(fft_values[0][0][max_fft_peaks_index[0]],
                     fft_values[1][0][max_fft_peaks_index[1]],
                     fft_values[2][0][max_fft_peaks_index[2]])
# Wavelet
widths = np.arange(1,50)
wav_pzt = (signal.cwt(y_filtered[0], signal.ricker, widths),
           signal.cwt(y_filtered[1], signal.ricker, widths),
           signal.cwt(y_filtered[2], signal.ricker, widths))

# Plots (Holes)
# Timeseries PZT Sensors
plt.figure()
plt.subplot(3, 1, 1)
plt.plot(t, y_filtered[0])
plt.title('Time Series - Original PZT Data Plotted')
plt.ylim([-np.nanmax(y_filtered[0])-(np.nanmax(y_filtered[0])/3),
          np.nanmax(y_filtered[0])+(np.nanmax(y_filtered[0]))/3])
plt.ylabel('PZT1 Displ. [m]')
plt.plot([max_time_peaks_time[0]], [max_time_peaks_values[0]], marker='X', mfc = 'darkred', ms='10')
plt.subplot(3, 1, 2)
plt.plot(t, y_filtered[1])
plt.ylim([-np.nanmax(y_filtered[1])-(np.nanmax(y_filtered[1])/3),
          np.nanmax(y_filtered[1])+(np.nanmax(y_filtered[1]))/3])
plt.ylabel('PZT2 Displ. [m]')
plt.plot([max_time_peaks_time[1]], [max_time_peaks_values[1]], marker='X', mfc = 'darkred', ms='10')
plt.subplot(3, 1, 3)
plt.plot(t, y_filtered[2])
plt.ylim([-np.nanmax(y_filtered[2])-(np.nanmax(y_filtered[2])/3),
          np.nanmax(y_filtered[2])+(np.nanmax(y_filtered[2]))/3])
plt.ylabel('PZT3 Displ. [m]')
plt.xlabel('Time [s]')
plt.plot([max_time_peaks_time[2]], [max_time_peaks_values[2]], marker='X', mfc = 'darkred', ms='10')
plt.tight_layout()
# FFT PZT Sensors
plt.figure()
plt.plot(fft_values[0][0], fft_values[0][1])
plt.ylim([0, np.nanmax(fft_values[0][1])+(np.nanmax(fft_values[0][1])/6)])
plt.xlim([0,200000])
plt.title('Fast Fourier Transform')
plt.ylabel('PZT1 Amplitude [m]')
plt.xlabel('Frequency [Hz]')
plt.plot([max_fft_peaks_time[0]], [max_fft_peaks_values[0]], marker='X', mfc = 'goldenrod', ms='10')
plt.figure()
plt.plot(fft_values[0][2] * (np.pi / 180), fft_values[0][3])
plt.ylabel('PZT Phase [rad]')
plt.xlabel('Frequency [Hz]')
plt.tight_layout()
# Wavelet
plt.figure()
plt.subplot(3, 1, 1)
plt.title('Wavelet')
plt.ylabel('PZT1')
plt.imshow(wav_pzt[0], extent=[-1, 1, 1, 50], cmap='PRGn', aspect='auto',
           vmax=abs(wav_pzt[0]).max(), vmin=-abs(wav_pzt[0]).max())
plt.subplot(3, 1, 2)
plt.ylabel('PZT2')
plt.imshow(wav_pzt[1], extent=[-1, 1, 1, 50], cmap='PRGn', aspect='auto',
           vmax=abs(wav_pzt[1]).max(), vmin=-abs(wav_pzt[1]).max())
plt.subplot(3, 1, 3)
plt.ylabel('PZT3')
plt.imshow(wav_pzt[2], extent=[-1, 1, 1, 50], cmap='PRGn', aspect='auto',
           vmax=abs(wav_pzt[1]).max(), vmin=-abs(wav_pzt[1]).max())
plt.tight_layout()
```

```
# Plots (Adhesives)
# Timeseries PZT Sensor
plt.figure()
plt.plot(t, y_filtered[2])
plt.title('Time Series - Adhesives Data Plotted')
plt.ylim([-np.nanmax(y_filtered[2])-(np.nanmax(y_filtered[2])/6),
          np.nanmax(y_filtered[2])+(np.nanmax(y_filtered[2]))/6])
plt.ylabel('PZT Displacement [m]')
plt.plot([max_time_peaks_time[2]], [max_time_peaks_values[2]], marker='X', mfc = 'darkred', ms='10')
plt.xlabel('Time [s]')
plt.tight_layout()
# FFT PZT Sensors
plt.figure()
plt.plot(fft_values[1][0], fft_values[1][1])
plt.xlim([0, 200000])
plt.ylim([0, np.nanmax(fft_values[1][1])+(np.nanmax(fft_values[1][1])/6)])
plt.title('Fast Fourier Transform - Adhesives Data')
plt.ylabel('PZT Amplitude [m]')
plt.plot([max_fft_peaks_time[1]], [max_fft_peaks_values[1]], marker='X', mfc = 'goldenrod', ms='10')
plt.xlabel('Frequency [Hz]')
plt.figure()
plt.plot(fft_values[1][2] * (np.pi/180), fft_values[1][3])
plt.ylabel('PZT Phase [rad]')
plt.xlabel('Frequency [Hz]')
plt.tight_layout()
# Wavelet
plt.figure()
plt.title('Wavelet Transform - Adhesives Data')
plt.imshow(wav_pzt[0], extent=[-1, 1, 1, 50], cmap='PRGn', aspect='auto',
           vmax=abs(wav_pzt[0]).max(), vmin=-abs(wav_pzt[0]).max())
plt.tight_layout()

#SHOWPLOTS
plt.show()



### PART 2 - AUTOMATIC FEATURE EXTRACTION / SELECTION ###


# tsfresh
# File preparation (Holes)
timearray = [desired_time] * 1800
timearray = np.concatenate(timearray)
ts_df = ts_dataframe.copy()
ts_df = ts_df.reset_index()
ts_df["timeseries"] = ts_df["test"].astype(str) + ts_df["series"]
ts_df = ts_df.drop(['test', 'series'], axis=1)
ts_df = ts_df.set_index('timeseries')
ts_df_temp = ts_df.stack()
ts_df_temp = ts_df_temp.reset_index()
ts_df_temp = ts_df_temp.drop(['level_1'], axis=1)
ts_df_tsfresh_index = ts_df_temp.iloc[:,0].str.split(pat = 'p', n = 0, expand = True)
ts_df_temp = ts_df_temp.drop(['timeseries'], axis=1)
ts_df_tsfresh_time = pd.DataFrame(timearray, columns = ['time'])
ts_df_tsfresh_values = ts_df_temp.rename(columns = {0: 'value'}, inplace = False)
ts_df_tsfresh_index = ts_df_tsfresh_index.rename(columns = {0: 'id', 1: 'kind'}, inplace = False)
ts_df_tsfresh = pd.concat([ts_df_tsfresh_index, ts_df_tsfresh_time, ts_df_tsfresh_values], axis = 1)

# File preparation (Adhesives)
timearray = [desired_time] * 900
timearray = np.concatenate(timearray)
ts_df = ts_dataframe.copy()
ts_df_temp = ts_df.stack()
ts_df_temp = pd.DataFrame(ts_df_temp)
ts_df_tsfresh_index = ts_df_temp.index.get_level_values(0)
ts_df_tsfresh_index = pd.DataFrame(ts_df_tsfresh_index, columns = ['id'])
ts_df_tsfresh_time = pd.DataFrame(timearray, columns = ['time'])
ts_df_tsfresh_values = ts_df_temp.rename(columns = {0: 'value'}, inplace = False)
ts_df_tsfresh_values = ts_df_tsfresh_values.droplevel(0)
ts_df_tsfresh_values = ts_df_tsfresh_values.set_index(ts_df_tsfresh_time.index)
ts_df_tsfresh = pd.concat([ts_df_tsfresh_index, ts_df_tsfresh_time, ts_df_tsfresh_values], axis = 1)
```

```
# Extract Features
# Holes
tsfresh_testdata = ts_df_tsfresh.copy()
tsfresh_features = extract_features(tsfresh_testdata,
                                    column_id="id",
                                    column_sort="time",
                                    column_kind="kind",
                                    column_value="value")
# Adhesives
tsfresh_testdata = ts_df_tsfresh.copy()
tsfresh_features = extract_features(tsfresh_testdata,
                                    column_id="id",
                                    column_sort="time",
                                    column_kind=None,
                                    column_value=None)

# Save to .csv after extracting
tsfresh_features.to_csv(r'C:\Users\GRamalho\Desktop\
                        PYTHON_PATH_FEATURES\features_verdadeiras_furos_FULL_LIVE.csv')

# Feature Loading (Holes/Adhesives)
readcsv_var_holes = pd.read_csv(r'C:\Users\vlore\Desktop\Workbox\5º Ano\Tese\python code\
                                Features\LIVE\features_no_damage_furos_FULL_LIVE.csv')
readcsv_var_adhesives = pd.read_csv(r'C:\Users\vlore\Desktop\Workbox\5º Ano\Tese\python code\
                                    Features\LIVE\features_verdadeiras_adesivos_FULL_LIVE.csv')

# Complete Feature File Processing

# Holes
tsfresh_fs = readcsv_var_holes.copy()
tsfresh_fs.iloc[:,0] = tsfresh_fs.iloc[:,0].astype(int)
tsfresh_fs = tsfresh_fs.sort_values(by = "index", axis = 0)
tsfresh_fs_index_list = [[hole_info[i][3]] for i in np.arange(0,2400,4)]
tsfresh_fs_index_list =  [item for subl in tsfresh_fs_index_list for item in subl]
tsfresh_fs_target = pd.Series(tsfresh_fs_index_list, index = tsfresh_fs.iloc[:,0])
tsfresh_fs = tsfresh_fs.set_index(['index'])
tsfresh_fs.index.name = 'ID'
# Drops (Failed Features)
tsfresh_fs = tsfresh_fs.drop(['zt1__query_similarity_count__query_None__threshold_0.0'], axis = 1)
tsfresh_fs = tsfresh_fs.drop(['zt2__query_similarity_count__query_None__threshold_0.0'], axis = 1)
tsfresh_fs = tsfresh_fs.drop(['zt3__query_similarity_count__query_None__threshold_0.0'], axis = 1)
for i in range(1,4) :
    for j in range(0,10):
        tsfresh_fs = tsfresh_fs.drop(['zt{}__autocorrelation__lag_{}'.format(i,j)], axis = 1)

# Adhesives
tsfresh_fs = readcsv_var_adhesives.copy()
tsfresh_fs.iloc[:,0] = tsfresh_fs.iloc[:,0].astype(int)
tsfresh_fs = tsfresh_fs.sort_values(by = "Unnamed: 0", axis = 0)
tsfresh_fs_index_list = [[adhesion_info[i][3]] for i in np.arange(0,1800,2)]
tsfresh_fs_index_list = [item for subl in tsfresh_fs_index_list for item in subl]
tsfresh_fs_index_list = np.array(tsfresh_fs_index_list)
adh_class_count = 5
min_target = min(tsfresh_fs_index_list)
max_target = max(tsfresh_fs_index_list)
target_range = (max_target - min_target)/adh_class_count
tsfresh_fs_target = np.array(tsfresh_fs_index_list)
for i in range(adh_class_count):
    tsfresh_fs_target[np.where((tsfresh_fs_index_list >= target_range * (i) + min_target) &
                               (tsfresh_fs_index_list < target_range * (1+i) + min_target))] = (i+1)
tsfresh_fs_target[-1:] = adh_class_count
tsfresh_fs_target = pd.Series(tsfresh_fs_target, index = tsfresh_fs.iloc[:,0])
tsfresh_fs = tsfresh_fs.set_index(['Unnamed: 0'])
tsfresh_fs.index.name = 'ID'
# Drops (Failed Features)
tsfresh_fs = tsfresh_fs.drop(['value__query_similarity_count__query_None__threshold_0.0'], axis = 1)
for i in range(0,10):
    tsfresh_fs = tsfresh_fs.drop(['value__autocorrelation__lag_{}'.format(i)], axis = 1)
```

```
# Feature Selection for Classification Task
tsfresh_selected_features = select_features(tsfresh_fs,
                                            tsfresh_fs_target,
                                            ml_task = 'classification',
                                            multiclass = True, fdr_level = 0.05)
p_values_table = tsfresh.feature_selection.relevance.calculate_relevance_table(tsfresh_fs,
                                                                               tsfresh_fs_target,
                                                                               ml_task = 'classification',
                                                                               multiclass = True,
                                                                               fdr_level = 0.05)
p_values_table = p_values_table.reset_index(drop = True)

# Features considered relevant for all classes (Holes)
top_features_index = p_values_table.iloc[np.where(p_values_table.loc[:,'n_significant']==4)[0], 0]
top_features_index = set(top_features_index.values)
top_features_index = list(top_features_index)
tsfresh_top_features = tsfresh_selected_features.loc[: , top_features_index]
# Features considered relevant for all classes (Adhesives)
top_features_index = p_values_table.iloc[np.where(p_values_table.loc[:,'n_significant']==5)[0], 0]
top_features_index = set(top_features_index.values)
top_features_index = list(top_features_index)
tsfresh_top_features = tsfresh_selected_features.loc[: , top_features_index]

# Save to .csv after selecting
tsfresh_selected_features.to_csv(r'C:\Users\vlore\Desktop\Workbox\5º Ano\Tese\python code\
                                 Features\LIVE\selected_no_damage_furos_FULL_LIVE.csv')
tsfresh_top_features.to_csv(r'C:\Users\vlore\Desktop\Workbox\5º Ano\Tese\python code\
                            Features\LIVE\top_no_damage_furos_FULL_LIVE.csv')

# Selected Features Analysis
# Holes
# Target well distributed (check)
print("Number of tests with no holes = " + str(len(np.where(tsfresh_fs_target==0)[0])))
print("Number of tests with 2mm holes = " + str(len(np.where(tsfresh_fs_target==2)[0])))
print("Number of tests with 6mm holes = " + str(len(np.where(tsfresh_fs_target==6)[0])))
print("Number of tests with 10mm holes = " + str(len(np.where(tsfresh_fs_target==10)[0])))
# Number of features with p-values < 0,05
print("Number of features with p-values < 0.05 (no holes) = "
      + str(len(np.where(p_values_table.loc[:,'p_value_0.0']<0.05)[0])))
print("Number of features with p-values < 0.05 (2mm holes) = "
      + str(len(np.where(p_values_table.loc[:,'p_value_2.0']<0.05)[0])))
print("Number of features with p-values < 0.05 (6mm holes) = "
      + str(len(np.where(p_values_table.loc[:,'p_value_6.0']<0.05)[0])))
print("Number of features with p-values < 0.05 (10mm holes) = "
      + str(len(np.where(p_values_table.loc[:,'p_value_10.0']<0.05)[0])))
# p-values histograms
plt.figure()
plt.subplot(1, 4, 1)
x_0 = plt.hist(p_values_table.loc[:,'p_value_0.0'].values,
               bins = 25, alpha = 0.5, color = 'y', histtype = 'bar',
               label = 'no damage', edgecolor="black")
x_0_contour = plt.hist(p_values_table.loc[:,'p_value_0.0'].values,
                       bins = 25, color = 'y', histtype = 'step', edgecolor="black")
plt.legend(loc='upper right')
plt.xlabel("p-value")
plt.ylabel("Count")
axes = plt.gca()
axes.set_ylim([None,np.max(x_0[0])+25])
plt.subplot(1, 4, 2)
x_2 = plt.hist(p_values_table.loc[:,'p_value_2.0'].values,
               bins = 25, alpha = 0.5, color = 'b', histtype = 'bar', label = '2mm',
edgecolor="black")
x_2_contour = plt.hist(p_values_table.loc[:,'p_value_2.0'].values,
                       bins = 25, color = 'b', histtype = 'step', edgecolor="black")
plt.legend(loc='upper right')
plt.xlabel("p-value")
plt.ylabel("Count")
axes = plt.gca()
axes.set_ylim([None,np.max(x_0[0])+25])
plt.subplot(1, 4, 3)
x_6 = plt.hist(p_values_table.loc[:,'p_value_6.0'].values,
               bins = 25, alpha = 0.5, color = 'r', histtype = 'bar', label = '6mm',
edgecolor="black")
x_6_contour = plt.hist(p_values_table.loc[:,'p_value_6.0'].values,
                       bins = 25,  color = 'r', histtype = 'step', edgecolor="black")
plt.legend(loc='upper right')
plt.xlabel("p-value")
axes = plt.gca()
axes.set_ylim([None,np.max(x_0[0])+25])
```

```
plt.subplot(1, 4, 4)
x_10 = plt.hist(p_values_table.loc[:,'p_value_10.0'].values,
                bins = 25, alpha = 0.5, color = 'g', histtype = 'bar', label = '10mm',
edgecolor="black")
x_10_contour = plt.hist(p_values_table.loc[:,'p_value_10.0'].values,
                        bins = 25, color = 'g', histtype = 'step', edgecolor="black")
plt.legend(loc='upper right')
plt.xlabel("p-value")
axes = plt.gca()
axes.set_ylim([None,np.max(x_0[0])+25])
plt.tight_layout()
# Number of relevant features (after Benji-Hoch)
print("Number of features considered relevant (no holes) = "
      + str(len(np.where(p_values_table.loc[:,'relevant_0.0']==True)[0])))
print("Number of features considered relevant (2mm holes) = "
      + str(len(np.where(p_values_table.loc[:,'relevant_2.0']==True)[0])))
print("Number of features considered relevant (6mm holes) = "
      + str(len(np.where(p_values_table.loc[:,'relevant_6.0']==True)[0])))
print("Number of features considered relevant (10mm holes) = "
      + str(len(np.where(p_values_table.loc[:,'relevant_10.0']==True)[0])))

# Adhesives (5 classes)
# Target well distributed (check)
print("Number of tests with Adhesion Class 1 = " + str(len(np.where(tsfresh_fs_target==1)[0])))
print("Number of tests with Adhesion Class 2 = " + str(len(np.where(tsfresh_fs_target==2)[0])))
print("Number of tests with Adhesion Class 3 = " + str(len(np.where(tsfresh_fs_target==3)[0])))
print("Number of tests with Adhesion Class 4 = " + str(len(np.where(tsfresh_fs_target==4)[0])))
print("Number of tests with Adhesion Class 5 = " + str(len(np.where(tsfresh_fs_target==5)[0])))
# Number of features with p-values < 0,05
print("Number of features with p-values < 0.05 (Adhesion Class 1) = "
      + str(len(np.where(p_values_table.loc[:,'p_value_1.0']<0.05)[0])))
print("Number of features with p-values < 0.05 (Adhesion Class 2) = "
      + str(len(np.where(p_values_table.loc[:,'p_value_2.0']<0.05)[0])))
print("Number of features with p-values < 0.05 (Adhesion Class 3) = "
      + str(len(np.where(p_values_table.loc[:,'p_value_3.0']<0.05)[0])))
print("Number of features with p-values < 0.05 (Adhesion Class 4) = "
      + str(len(np.where(p_values_table.loc[:,'p_value_4.0']<0.05)[0])))
print("Number of features with p-values < 0.05 (Adhesion Class 5) = "
      + str(len(np.where(p_values_table.loc[:,'p_value_5.0']<0.05)[0])))
# p-values histograms
plt.figure()
plt.subplot(1, 5, 1)
x_1 = plt.hist(p_values_table.loc[:,'p_value_1.0'].values,
               bins = 25, alpha = 0.5, color = 'b', histtype = 'bar',
               label = 'Class 1', edgecolor="black")
x_1_contour = plt.hist(p_values_table.loc[:,'p_value_1.0'].values,
                       bins = 25, color = 'b', histtype = 'step', edgecolor="black")
plt.legend(loc='upper right')
plt.xlabel("p-value")
plt.ylabel("Count")
axes = plt.gca()
axes.set_ylim([None,np.max(x_1[0])+25])
plt.subplot(1, 5, 2)
x_2 = plt.hist(p_values_table.loc[:,'p_value_2.0'].values,
               bins = 25, alpha = 0.5, color = 'r', histtype = 'bar',
               label = 'Class 2', edgecolor="black")
x_2_contour = plt.hist(p_values_table.loc[:,'p_value_2.0'].values,
                       bins = 25,  color = 'r', histtype = 'step', edgecolor="black")
plt.legend(loc='upper right')
plt.xlabel("p-value")
axes = plt.gca()
axes.set_ylim([None,np.max(x_1[0])+25])
plt.subplot(1, 5, 3)
x_3 = plt.hist(p_values_table.loc[:,'p_value_3.0'].values,
               bins = 25, alpha = 0.5, color = 'g', histtype = 'bar',
               label = 'Class 3', edgecolor="black")
x_3_contour = plt.hist(p_values_table.loc[:,'p_value_3.0'].values,
                       bins = 25, color = 'g', histtype = 'step', edgecolor="black")
plt.legend(loc='upper right')
plt.xlabel("p-value")
axes = plt.gca()
axes.set_ylim([None,np.max(x_1[0])+25])
```

```
plt.subplot(1, 5, 4)
x_4 = plt.hist(p_values_table.loc[:,'p_value_4.0'].values,
               bins = 25, alpha = 0.5, color = 'y', histtype = 'bar',
               label = 'Class 4', edgecolor="black")
x_4_contour = plt.hist(p_values_table.loc[:,'p_value_4.0'].values,
                       bins = 25,  color = 'y', histtype = 'step', edgecolor="black")
plt.legend(loc='upper right')
plt.xlabel("p-value")
axes = plt.gca()
axes.set_ylim([None,np.max(x_1[0])+25])
plt.subplot(1, 5, 5)
x_5 = plt.hist(p_values_table.loc[:,'p_value_5.0'].values,
               bins = 25, alpha = 0.5, color = 'm', histtype = 'bar',
               label = 'Class 5', edgecolor="black")
x_5_contour = plt.hist(p_values_table.loc[:,'p_value_5.0'].values,
                       bins = 25, color = 'm', histtype = 'step', edgecolor="black")
plt.legend(loc='upper right')
plt.xlabel("p-value")
axes = plt.gca()
axes.set_ylim([None,np.max(x_1[0])+25])
# Number of relevant features (after Benji-Hoch)
print("Number of features considered relevant (Adhesion Class 1) = "
      + str(len(np.where(p_values_table.loc[:,'relevant_1.0']==True)[0])))
print("Number of features considered relevant (Adhesion Class 2) = "
      + str(len(np.where(p_values_table.loc[:,'relevant_2.0']==True)[0])))
print("Number of features considered relevant (Adhesion Class 3) = "
      + str(len(np.where(p_values_table.loc[:,'relevant_3.0']==True)[0])))
print("Number of features considered relevant (Adhesion Class 4) = "
      + str(len(np.where(p_values_table.loc[:,'relevant_4.0']==True)[0])))
print("Number of features considered relevant (Adhesion Class 5) = "
      + str(len(np.where(p_values_table.loc[:,'relevant_5.0']==True)[0])))



### PART 3 - SIGNAL / FEATURE VISUALIZATION ###


# Feature Loading

# Selected Feature Loading Holes
readcsv_var_selected = pd.read_csv(r'C:\Users\vlore\Desktop\Workbox\5º Ano\Tese\python code\
                                   Features\LIVE\selected_no_damage_furos_FULL_LIVE.csv')
sel_feature_data = readcsv_var_selected.set_index('ID')
# Top Feature Loading Holes
readcsv_var_top = pd.read_csv(r'C:\Users\vlore\Desktop\Workbox\5º Ano\Tese\python code\
                              Features\LIVE\top_no_damage_furos_FULL_LIVE.csv')
top_feature_data = readcsv_var_top.set_index('ID')

# Selected Feature Loading Adhesives
readcsv_var_selected = pd.read_csv(r'C:\Users\vlore\Desktop\Workbox\5º Ano\Tese\python code\
                                   Features\LIVE\selected_features_adesivos_FULL_LIVE.csv')
sel_feature_data = readcsv_var_selected.set_index('Unnamed: 0')
sel_feature_data.index.name = 'ID'
# Top Feature Loading Adhesives
readcsv_var_top = pd.read_csv(r'C:\Users\vlore\Desktop\Workbox\5º Ano\Tese\python code\
                              Features\LIVE\top_features_adesivos_FULL_LIVE.csv')
top_feature_data = readcsv_var_top.set_index('Unnamed: 0')
top_feature_data.index.name = 'ID'
# Adhesion Strength Classes Definition
# For adhesive data, the data set can be presented by adhesion strenght value,
# or by adhesion class (very weak, weak, medium, strong, very strong)
adhesion_strength = list(adhesion_info[i][3] for i in np.arange(1,1800,2))
tsfresh_fs_index_list = [[adhesion_info[i][3]] for i in np.arange(0,1800,2)]
tsfresh_fs_index_list = [item for subl in tsfresh_fs_index_list for item in subl]
tsfresh_fs_index_list = np.array(tsfresh_fs_index_list)
adh_class_count = 5
min_target = min(tsfresh_fs_index_list)
max_target = max(tsfresh_fs_index_list)
target_range = (max_target - min_target)/adh_class_count
tsfresh_fs_target = np.array(tsfresh_fs_index_list)
for i in range(adh_class_count):
    tsfresh_fs_target[np.where((tsfresh_fs_index_list >= target_range * (i) + min_target) &
                               (tsfresh_fs_index_list < target_range * (1+i) + min_target))] = (i+1)
tsfresh_fs_target[-1:] = adh_class_count
tsfresh_fs_target = pd.Series(tsfresh_fs_target, index = tsfresh_fs.iloc[:,0])
adhesion_class = tsfresh_fs_target.copy()
```

```
# Feature Data
# To select different features for presentation, change feat_viz variable,
# the index number at the feature_data.iloc

# Holes
# MDS 3D Visualization - selected features
fig_feat_sel = plt.figure()
ax_feat_sel = Axes3D(fig_feat_sel)
plt.title('MDS 3D - Selected Feature')
feat_viz_sel = np.array(sel_feature_data.iloc[:,23]).reshape(-1,1)
hole_sizes = list(hole_info[i][3] for i in np.arange(1,2400,4))
mds_feat_viz_sel = mds_data_visualization_3D(feat_viz_sel)
feat_mds3d_sel = ax_feat_sel.scatter(mds_feat_viz_sel[:,0],mds_feat_viz_sel[:,1],
mds_feat_viz_sel[:,2],
                                     c=hole_sizes, cmap = 'jet')
ax_feat_sel.scatter(mds_feat_viz_sel[:,0],mds_feat_viz_sel[:,1], mds_feat_viz_sel[:,2],
                    c=hole_sizes, cmap = 'jet')
ax_feat_sel.legend(*feat_mds3d_sel.legend_elements(), loc = 'upper right', title = 'Classes')
# MDS 3D Visualization - top features
fig_feat_top = plt.figure()
ax_feat_top = Axes3D(fig_feat_top)
plt.title('MDS 3D - Selected Feature')
feat_viz_top = np.array(top_feature_data.iloc[:,23]).reshape(-1,1)
hole_sizes = list(hole_info[i][3] for i in np.arange(1,2400,4))
mds_feat_viz_top = mds_data_visualization_3D(feat_viz_top)
feat_mds3d_top = ax_feat_top.scatter(mds_feat_viz_top[:,0],mds_feat_viz_top[:,1],
mds_feat_viz_top[:,2],
                                     c=hole_sizes, cmap = 'jet')
ax_feat_top.scatter(mds_feat_viz_top[:,0],mds_feat_viz_top[:,1], mds_feat_viz_top[:,2],
                    c=hole_sizes, cmap = 'jet')
ax_feat_top.legend(*feat_mds3d_top.legend_elements(), loc = 'upper right', title = 'Classes')
# t-SNE 3D Visualization - top features
fig_feat_top_tsne = plt.figure()
ax_feat_top_tsne = Axes3D(fig_feat_top_tsne)
plt.title('t-SNE - PZT Selected Feature')
feat_viz_top_tsne = np.array(top_feature_data.iloc[:,183]).reshape(-1,1)
hole_sizes = list(hole_info[i][3] for i in np.arange(1,2400,4))
tsne_feat_viz_top = tsne_data_visualization_3D(feat_viz_top_tsne)
feat_tsne3d_top = ax_feat_top_tsne.scatter(
    tsne_feat_viz_top[:,0],tsne_feat_viz_top[:,1], tsne_feat_viz_top[:,2],
    c=hole_sizes, cmap = 'jet')
ax_feat_top_tsne.scatter(tsne_feat_viz_top[:,0],tsne_feat_viz_top[:,1], tsne_feat_viz_top[:,2],
                         c=hole_sizes, cmap = 'jet')
ax_feat_top_tsne.legend(*feat_tsne3d_top.legend_elements(), loc = 'upper right', title = 'Classes')

# Adhesives
# MDS 3D Visualization - selected features
fig_feat_sel = plt.figure()
ax_feat_sel = Axes3D(fig_feat_sel)
plt.title('MDS 3D - Selected Feature')
feat_viz_sel = np.array(sel_feature_data.iloc[:,1]).reshape(-1,1)
mds_feat_viz_sel = mds_data_visualization_3D(feat_viz_sel)
ax_feat_sel.scatter(mds_feat_viz_sel[:,0],mds_feat_viz_sel[:,1], mds_feat_viz_sel[:,2],
                    c=adhesion_strength, cmap = 'jet')
feat_mds3d_sel = ax_feat_sel.scatter(mds_feat_viz_sel[:,0],mds_feat_viz_sel[:,1],
mds_feat_viz_sel[:,2],
                                     c=adhesion_class, cmap = 'jet')
ax_feat_sel.scatter(mds_feat_viz_sel[:,0],mds_feat_viz_sel[:,1], mds_feat_viz_sel[:,2],
                    c=adhesion_class, cmap = 'jet')
ax_feat_sel.legend(*feat_mds3d_sel.legend_elements(), loc = 'upper right', title = 'Classes')
# MDS 3D Visualization - top features
fig_feat_top = plt.figure()
ax_feat_top = Axes3D(fig_feat_top)
plt.title('MDS 3D - Selected Feature')
feat_viz_top = np.array(top_feature_data.iloc[:,4]).reshape(-1,1)
mds_feat_viz_top = mds_data_visualization_3D(feat_viz_top)
ax_feat_top.scatter(mds_feat_viz_top[:,0],mds_feat_viz_top[:,1], mds_feat_viz_top[:,2],
                    c=adhesion_strength, cmap = 'jet')
feat_mds3d_top = ax_feat_top.scatter(mds_feat_viz_top[:,0],mds_feat_viz_top[:,1],
mds_feat_viz_top[:,2],
                                     c=adhesion_class, cmap = 'jet')
ax_feat_top.scatter(mds_feat_viz_top[:,0],mds_feat_viz_top[:,1], mds_feat_viz_top[:,2],
                    c=adhesion_class, cmap = 'jet')
ax_feat_top.legend(*feat_mds3d_top.legend_elements(), loc = 'upper right', title = 'Classes')
```

```
# t-SNE 3D Visualization - top features
fig_feat_top_tsne = plt.figure()
ax_feat_top_tsne = Axes3D(fig_feat_top_tsne)
plt.title('t-SNE 3D - PZT Selected Feature')
feat_viz_top_tsne = np.array(top_feature_data.iloc[:,8]).reshape(-1,1)
tsne_feat_viz_top = tsne_data_visualization_3D(feat_viz_top_tsne)
ax_feat_top_tsne.scatter(tsne_feat_viz_top[:,0],tsne_feat_viz_top[:,1], tsne_feat_viz_top[:,2],
                         c=adhesion_strength, cmap = 'jet')
feat_tsne3d_top = ax_feat_top_tsne.scatter(
    tsne_feat_viz_top[:,0],tsne_feat_viz_top[:,1], tsne_feat_viz_top[:,2], c=adhesion_class, cmap =
'jet')
ax_feat_top_tsne.scatter(tsne_feat_viz_top[:,0],tsne_feat_viz_top[:,1], tsne_feat_viz_top[:,2],
                         c=adhesion_class, cmap = 'jet')
ax_feat_top_tsne.legend(*feat_tsne3d_top.legend_elements(), loc = 'upper right', title = 'Classes')


# Time Series Data

# MDS 2D Visualization - time-series (Holes)
# PZT1
plt.figure()
plt.title('MDS 2D - PZT1 Signal Time Series')
pzt1_data = list(input_data_downsampled[i] for i in np.arange(0,1799,3))
pzt1_hole_size = list(hole_info[i][3] for i in np.arange(1,2400,4))
mds_pzt1 = mds_data_visualization_2D(pzt1_data)
tsh_mds2d_pzt1 = plt.scatter(mds_pzt1[:,0],mds_pzt1[:,1], c=pzt1_hole_size, cmap = 'jet')
plt.scatter(mds_pzt1[:,0],mds_pzt1[:,1], c=pzt1_hole_size, cmap = 'jet')
plt.legend(*tsh_mds2d_pzt1.legend_elements(), loc = 'upper right', title = 'Classes')
# PZT2
plt.figure()
plt.title('MDS 2D - PZT2 Signal Time Series')
pzt2_data = list(input_data_downsampled[i] for i in np.arange(1,1800,3))
pzt2_hole_size = list(hole_info[i][3] for i in np.arange(2,2400,4))
mds_pzt2 = mds_data_visualization_2D(pzt2_data)
tsh_mds2d_pzt2 = plt.scatter(mds_pzt2[:,0],mds_pzt2[:,1], c=pzt2_hole_size, cmap = 'tab10')
plt.scatter(mds_pzt2[:,0],mds_pzt2[:,1], c=pzt2_hole_size, cmap = 'tab10')
plt.legend(*tsh_mds2d_pzt2.legend_elements(), loc = 'upper right', title = 'Classes')
# PZT3
plt.figure()
plt.title('MDS 2D - PZT3 Signal Time Series')
pzt3_data = list(input_data_downsampled[i] for i in np.arange(2,1801,3))
pzt3_hole_size = list(hole_info[i][3] for i in np.arange(3,2400,4))
mds_pzt3 = mds_data_visualization_2D(pzt3_data)
tsh_mds2d_pzt3 = plt.scatter(mds_pzt3[:,0],mds_pzt3[:,1], c=pzt3_hole_size, cmap = 'jet')
plt.scatter(mds_pzt3[:,0],mds_pzt3[:,1], c=pzt3_hole_size, cmap = 'jet')
plt.legend(*tsh_mds2d_pzt3.legend_elements(), loc = 'upper right', title = 'Classes')

# MDS 2D Visualization - time-series (Adhesives)
plt.figure()
plt.title('MDS 2D - PZT Signal Time Series')
pzt_data = list(input_data_downsampled[i] for i in np.arange(0,900))
mds_pzt = mds_data_visualization_2D(pzt_data)
plt.scatter(mds_pzt[:,0],mds_pzt[:,1], c=adhesion_strength, cmap = 'jet')
tsa_mds2d = plt.scatter(mds_pzt[:,0],mds_pzt[:,1], c=adhesion_strength, cmap = 'jet')
plt.scatter(mds_pzt[:,0],mds_pzt[:,1], c=adhesion_class, cmap = 'jet')
plt.legend(*tsa_mds2d.legend_elements(), loc = 'upper right', title = 'Classes')

# MDS 3D Visualization - time-series (Holes)
# PZT1
fig1 = plt.figure()
ax1 = Axes3D(fig1)
plt.title('MDS 3D - PZT1 Signal Time Series')
pzt1_data = list(input_data_downsampled[i] for i in np.arange(0,1799,3))
pzt1_hole_size = list(hole_info[i][3] for i in np.arange(1,2400,4))
mds_3D_pzt1 = mds_data_visualization_3D(pzt1_data)
tsh_mds3d_pzt1 = ax1.scatter(mds_3D_pzt1[:,0],mds_3D_pzt1[:,1], c=pzt1_hole_size, cmap = 'jet')
ax1.scatter(mds_3D_pzt1[:,0],mds_3D_pzt1[:,1], c=pzt1_hole_size, cmap = 'jet')
ax1.legend(*tsh_mds3d_pzt1.legend_elements(), loc = 'upper right', title = 'Classes')
# PZT2
fig2 = plt.figure()
ax2 = Axes3D(fig2)
plt.title('MDS 3D - PZT2 Signal Time Series')
pzt2_data = list(input_data_downsampled[i] for i in np.arange(1,1800,3))
pzt2_hole_size = list(hole_info[i][3] for i in np.arange(2,2400,4))
mds_3D_pzt2 = mds_data_visualization_3D(pzt2_data)
tsh_mds3d_pzt2 = ax2.scatter(mds_3D_pzt2[:,0],mds_3D_pzt2[:,1], c=pzt2_hole_size, cmap = 'jet')
ax2.scatter(mds_3D_pzt2[:,0],mds_3D_pzt2[:,1], c=pzt2_hole_size, cmap = 'jet')
ax2.legend(*tsh_mds3d_pzt2.legend_elements(), loc = 'upper right', title = 'Classes')
```

```
# PZT3
fig3 = plt.figure()
ax3 = Axes3D(fig3)
plt.title('MDS 3D - PZT3 Signal Time Series')
pzt3_data = list(input_data_downsampled[i] for i in np.arange(2,1801,3))
pzt3_hole_size = list(hole_info[i][3] for i in np.arange(3,2400,4))
mds_3D_pzt3 = mds_data_visualization_3D(pzt3_data)
tsh_mds3d_pzt3 = ax3.scatter(mds_3D_pzt3[:,0],mds_3D_pzt3[:,1], c=pzt3_hole_size, cmap = 'jet')
ax3.scatter(mds_3D_pzt3[:,0],mds_3D_pzt3[:,1], c=pzt3_hole_size, cmap = 'jet')
ax3.legend(*tsh_mds3d_pzt3.legend_elements(), loc = 'upper right', title = 'Classes')

# MDS 3D Visualization - time-series (Adhesives)
fig = plt.figure()
ax = Axes3D(fig)
plt.title('MDS 3D - PZT Signal Time Series')
pzt_data = list(input_data_downsampled[i] for i in np.arange(0,900))
mds_3D_pzt = mds_data_visualization_3D(pzt_data)
ax.scatter(mds_3D_pzt[:,0],mds_3D_pzt[:,1],mds_3D_pzt[:,2],c=adhesion_strength, cmap = 'jet')
tsa_mds3d = ax.scatter(mds_3D_pzt[:,0],mds_3D_pzt[:,1], c=adhesion_class, cmap = 'jet')
ax.scatter(mds_3D_pzt[:,0],mds_3D_pzt[:,1], c=adhesion_class, cmap = 'jet')
ax.legend(*tsa_mds3d.legend_elements(), loc = 'upper right', title = 'Classes')

# t-SNE Visualization - time-series (Holes)
# PZT1
plt.figure()
plt.title('t-SNE - PZT1 Signal Time Series')
pzt1_data = list(input_data_downsampled[i] for i in np.arange(0,1799,3))
pzt1_hole_size = list(hole_info[i][3] for i in np.arange(1,2400,4))
tsne_pzt1 = tsne_data_visualization_2D(pzt1_data)
tsh_tsne_pzt1 = plt.scatter(tsne_pzt1[:,0],tsne_pzt1[:,1], c=pzt1_hole_size, cmap = 'jet')
plt.scatter(tsne_pzt1[:,0],tsne_pzt1[:,1], c=pzt1_hole_size, cmap = 'jet')
plt.legend(*tsh_tsne_pzt1.legend_elements(), loc = 'upper right', title = 'Classes')
# PZT2
plt.figure()
plt.title('t-SNE - PZT2 Signal Time Series')
pzt2_data = list(input_data_downsampled[i] for i in np.arange(1,1800,3))
pzt2_hole_size = list(hole_info[i][3] for i in np.arange(2,2400,4))
tsne_pzt2 = tsne_data_visualization_2D(pzt2_data)
tsh_tsne_pzt2 = plt.scatter(tsne_pzt2[:,0],tsne_pzt2[:,1], c=pzt2_hole_size, cmap = 'jet')
plt.scatter(tsne_pzt2[:,0],tsne_pzt2[:,1], c=pzt2_hole_size, cmap = 'jet')
plt.legend(*tsh_tsne_pzt2.legend_elements(), loc = 'upper right', title = 'Classes')
# PZT3
plt.figure()
plt.title('t-SNE - PZT3 Signal Time Series')
pzt3_data = list(input_data_downsampled[i] for i in np.arange(2,1801,3))
pzt3_hole_size = list(hole_info[i][3] for i in np.arange(3,2400,4))
tsne_pzt3 = tsne_data_visualization_2D(pzt3_data)
tsh_tsne_pzt3 = plt.scatter(tsne_pzt3[:,0],tsne_pzt3[:,1], c=pzt3_hole_size, cmap = 'tab10')
plt.scatter(tsne_pzt3[:,0],tsne_pzt3[:,1], c=pzt3_hole_size, cmap = 'tab10')
plt.legend(*tsh_tsne_pzt3.legend_elements(), loc = 'upper right', title = 'Classes')

# t-SNE Visualization - time-series (Adhesives)
plt.figure()
plt.title('t-SNE - PZT Signal Time Series')
pzt_data = list(input_data_downsampled[i] for i in np.arange(0,900))
tsne_pzt = tsne_data_visualization_2D(pzt_data)
plt.scatter(tsne_pzt[:,0],tsne_pzt[:,1], c=adhesion_strength, cmap = 'jet')
tsa_tsne = plt.scatter(tsne_pzt[:,0],tsne_pzt[:,1], c=adhesion_class, cmap = 'jet')
plt.scatter(tsne_pzt[:,0],tsne_pzt[:,1], c=adhesion_class, cmap = 'jet')
plt.legend(*tsa_tsne.legend_elements(), loc = 'upper right', title = 'Classes')

#SHOWPLOTS
plt.show()
```

```
# Train/Test Splits

# Full Features 70/30 Split
clf_train_full, clf_test_full, tsfresh_fs_full_target_train, tsfresh_fs_full_target_test =
train_test_split(
    tsfresh_fs, tsfresh_fs_target, test_size=0.30, random_state = 1)
# Selected Features 70/30 Split
clf_train_sel, clf_test_sel, tsfresh_fs_sel_target_train, tsfresh_fs_sel_target_test =
train_test_split(
    sel_feature_data, tsfresh_fs_target, test_size=0.30, random_state = 1)
# Top Features 70/30 Split
clf_train_top, clf_test_top, tsfresh_fs_top_target_train, tsfresh_fs_top_target_test =
train_test_split(
    top_feature_data, tsfresh_fs_target, test_size=0.30, random_state = 1)

# KFold Split
kf = KFold(n_splits = 10, shuffle = True)

# KFold Split Classification Selected Features (RF / GBoost)
acc_kf_rf = []
acc_kf_gboost = []
f1_kf_rf = []
f1_kf_gboost = []
for train_index, test_index in kf.split(sel_feature_data):
        rf_kf_clf = RandomForestClassifier(max_depth = 10)
        gboost_kf_clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1)
        kf_train, kf_test = sel_feature_data.iloc[train_index,:], sel_feature_data.iloc[test_index,:]
        kf_target_train, kf_target_test = tsfresh_fs_target.iloc[train_index],
        tsfresh_fs_target.iloc[test_index]
        # Random Forest
        kf_rf_fit = rf_kf_clf.fit(kf_train, kf_target_train)
        kf_rf_predict = kf_rf_fit.predict(kf_test)
        acc_kf_rf += [metrics.accuracy_score(kf_target_test, kf_rf_predict)]
        f1_kf_rf += [metrics.f1_score(kf_target_test, kf_rf_predict, average = 'macro')]
        # Gradient Boosting
        kf_gboost_fit = gboost_kf_clf.fit(kf_train, kf_target_train)
        kf_gboost_predict = kf_gboost_fit.predict(kf_test)
        acc_kf_gboost += [metrics.accuracy_score(kf_target_test, kf_rf_predict)]
        f1_kf_gboost += [metrics.f1_score(kf_target_test, kf_rf_predict, average = 'macro')]

# Random Forest
rf_clf_full = RandomForestClassifier(max_depth = 10, random_state=1)
rf_clf_sel = RandomForestClassifier(max_depth = 10, random_state=1)
rf_clf_top = RandomForestClassifier(max_depth = 10, random_state=1)
# Using the FULL split
rf_clf_full = rf_clf_full.fit(clf_train_full, tsfresh_fs_full_target_train)
rf_predict_full = rf_clf_full.predict(clf_test_full)
# Predict on training SELECT split (verification)
rf_clf_sel = rf_clf_sel.fit(clf_train_sel, tsfresh_fs_full_target_train)
rf_predict_train = rf_clf_sel.predict(clf_train_sel)
# Predict on testing data SELECT split (live)
rf_predict_sel = rf_clf_sel.predict(clf_test_sel)
# Using the TOP split
rf_clf_top = rf_clf_top.fit(clf_train_top, tsfresh_fs_full_target_train)
rf_predict_top = rf_clf_top.predict(clf_test_top)

# K-Nearest Neighbors
knn_clf_full = KNeighborsClassifier(n_neighbors=5)
knn_clf_sel = KNeighborsClassifier(n_neighbors=5)
knn_clf_top = KNeighborsClassifier(n_neighbors=5)
# Using the FULL split
knn_clf_full = knn_clf_full.fit(clf_train_full, tsfresh_fs_full_target_train)
knn_predict_full = knn_clf_full.predict(clf_test_full)
# Predict on training SELECT split (verification)
knn_clf_sel = knn_clf_sel.fit(clf_train_sel, tsfresh_fs_full_target_train)
knn_predict_train = knn_clf_sel.predict(clf_train_sel)
# Predict on testing data SELECT split (live)
knn_predict_sel = knn_clf_sel.predict(clf_test_sel)
# Using the TOP split
knn_clf_top = knn_clf_top.fit(clf_train_top, tsfresh_fs_full_target_train)
knn_predict_top = knn_clf_top.predict(clf_test_top)
```

```
# Gaussian Naïve Bayes
gnb_clf_full = GaussianNB()
gnb_clf_sel = GaussianNB()
gnb_clf_top = GaussianNB()
# Using the FULL split
gnb_clf_full = gnb_clf_full.fit(clf_train_full, tsfresh_fs_full_target_train)
gnb_predict_full = gnb_clf_full.predict(clf_test_full)
# Predict on training SELECT split (verification)
gnb_clf_sel = gnb_clf_sel.fit(clf_train_sel, tsfresh_fs_full_target_train)
gnb_predict_train = gnb_clf_sel.predict(clf_train_sel)
# Predict on testing data SELECT split (live)
gnb_predict_sel = gnb_clf_sel.predict(clf_test_sel)
# Using the TOP split
gnb_clf_top = gnb_clf_top.fit(clf_train_top, tsfresh_fs_full_target_train)
gnb_predict_top = gnb_clf_top.predict(clf_test_top)

# Gradient Boosting
gboost_clf_full = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1)
gboost_clf_sel = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1)
gboost_clf_top = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1)
# Using the FULL split
gboost_clf_full = gboost_clf_full.fit(clf_train_full, tsfresh_fs_full_target_train)
gboost_predict_full = gboost_clf_full.predict(clf_test_full)
# Predict on training SELECT split (verification)
gboost_clf_sel = gboost_clf_sel.fit(clf_train_sel, tsfresh_fs_full_target_train)
gboost_predict_train = gboost_clf_sel.predict(clf_train_sel)
# Predict on testing data SELECT split (live)
gboost_predict_sel = gboost_clf_sel.predict(clf_test_sel)
# Using the TOP split
gboost_clf_top = gboost_clf_top.fit(clf_train_top, tsfresh_fs_full_target_train)
gboost_predict_top = gboost_clf_top.predict(clf_test_top)



# SKLearn - Metrics

# Confusion Matrices
# Random Forest
mx_rf_verify = metrics.confusion_matrix(tsfresh_fs_full_target_train, rf_predict_train)
mx_rf_pred_full = metrics.confusion_matrix(tsfresh_fs_full_target_test, rf_predict_full)
mx_rf_pred_sel = metrics.confusion_matrix(tsfresh_fs_full_target_test, rf_predict_sel)
mx_rf_pred_top = metrics.confusion_matrix(tsfresh_fs_full_target_test, rf_predict_top)
# KNN
mx_knn_verify = metrics.confusion_matrix(tsfresh_fs_full_target_train, knn_predict_train)
mx_knn_pred_full = metrics.confusion_matrix(tsfresh_fs_full_target_test, knn_predict_full)
mx_knn_pred_sel = metrics.confusion_matrix(tsfresh_fs_full_target_test, knn_predict_sel)
mx_nknn_pred_top = metrics.confusion_matrix(tsfresh_fs_full_target_test, knn_predict_top)
# Naïve Bayes
mx_gnb_verify = metrics.confusion_matrix(tsfresh_fs_full_target_train, gnb_predict_train)
mx_gnb_pred_full = metrics.confusion_matrix(tsfresh_fs_full_target_test, gnb_predict_full)
mx_gnb_pred_sel = metrics.confusion_matrix(tsfresh_fs_full_target_test, gnb_predict_sel)
mx_gnb_pred_top = metrics.confusion_matrix(tsfresh_fs_full_target_test, gnb_predict_top)
# Gradient Boosting
mx_gboost_verify = metrics.confusion_matrix(tsfresh_fs_full_target_train, gboost_predict_train)
mx_gboost_pred_full = metrics.confusion_matrix(tsfresh_fs_full_target_test, gboost_predict_full)
mx_gboost_pred_sel = metrics.confusion_matrix(tsfresh_fs_full_target_test, gboost_predict_sel)
mx_gboost_pred_top = metrics.confusion_matrix(tsfresh_fs_full_target_test, gboost_predict_top)
```

```
# Precision Score
# Random Forest
prc_rf_full = metrics.precision_score(tsfresh_fs_full_target_test, rf_predict_full, average = 'macro')
prc_rf_sel = metrics.precision_score(tsfresh_fs_full_target_test, rf_predict_sel, average = 'macro')
prc_rf_top = metrics.precision_score(tsfresh_fs_full_target_test, rf_predict_top, average = 'macro')
# KNN
prc_knn_full = metrics.precision_score(tsfresh_fs_full_target_test, knn_predict_full, average =
'macro')
prc_knn_sel = metrics.precision_score(tsfresh_fs_full_target_test, knn_predict_sel, average = 'macro')
prc_knn_top = metrics.precision_score(tsfresh_fs_full_target_test, knn_predict_top, average = 'macro')
# Naïve Bayes
prc_gnb_full = metrics.precision_score(tsfresh_fs_full_target_test, gnb_predict_full, average =
'macro')
prc_gnb_sel = metrics.precision_score(tsfresh_fs_full_target_test, gnb_predict_sel, average = 'macro')
prc_gnb_top = metrics.precision_score(tsfresh_fs_full_target_test, gnb_predict_top, average = 'macro')
# Gradient Boosting
prc_gboost_full = metrics.precision_score(tsfresh_fs_full_target_test, gboost_predict_full, average =
'macro')
prc_gboost_sel = metrics.precision_score(tsfresh_fs_full_target_test, gboost_predict_sel, average =
'macro')
prc_gboost_top = metrics.precision_score(tsfresh_fs_full_target_test, gboost_predict_top, average =
'macro')

# Recall Score
# Random Forest
rec_rf_full = metrics.recall_score(tsfresh_fs_full_target_test, rf_predict_full, average = 'macro')
rec_rf_sel = metrics.recall_score(tsfresh_fs_full_target_test, rf_predict_sel, average = 'macro')
rec_rf_top = metrics.recall_score(tsfresh_fs_full_target_test, rf_predict_top, average = 'macro')
# KNN
rec_knn_full = metrics.recall_score(tsfresh_fs_full_target_test, knn_predict_full, average = 'macro')
rec_knn_sel = metrics.recall_score(tsfresh_fs_full_target_test, knn_predict_sel, average = 'macro')
rec_knn_top = metrics.recall_score(tsfresh_fs_full_target_test, knn_predict_top, average = 'macro')
# Naïve Bayes
rec_gnb_full = metrics.recall_score(tsfresh_fs_full_target_test, gnb_predict_full, average = 'macro')
rec_gnb_sel = metrics.recall_score(tsfresh_fs_full_target_test, gnb_predict_sel, average = 'macro')
rec_gnb_top = metrics.recall_score(tsfresh_fs_full_target_test, gnb_predict_top, average = 'macro')
# Gradient Boosting
rec_gboost_full = metrics.recall_score(tsfresh_fs_full_target_test, gboost_predict_full, average =
'macro')
rec_gboost_sel = metrics.recall_score(tsfresh_fs_full_target_test, gboost_predict_sel, average =
'macro')
rec_gboost_top = metrics.recall_score(tsfresh_fs_full_target_test, gboost_predict_top, average =
'macro')

# F1 Score
# Random Forest
f1_rf_full = metrics.f1_score(tsfresh_fs_full_target_test, rf_predict_full, average = 'macro')
f1_rf_sel = metrics.f1_score(tsfresh_fs_full_target_test, rf_predict_sel, average = 'macro')
f1_rf_top = metrics.f1_score(tsfresh_fs_full_target_test, rf_predict_top, average = 'macro')
# KNN
f1_knn_full = metrics.f1_score(tsfresh_fs_full_target_test, knn_predict_full, average = 'macro')
f1_knn_sel = metrics.f1_score(tsfresh_fs_full_target_test, knn_predict_sel, average = 'macro')
f1_knn_top = metrics.f1_score(tsfresh_fs_full_target_test, knn_predict_top, average = 'macro')
# Naïve Bayes
f1_gnb_full = metrics.f1_score(tsfresh_fs_full_target_test, gnb_predict_full, average = 'macro')
f1_gnb_sel = metrics.f1_score(tsfresh_fs_full_target_test, gnb_predict_sel, average = 'macro')
f1_gnb_top = metrics.f1_score(tsfresh_fs_full_target_test, gnb_predict_top, average = 'macro')
# Gradient Boosting
f1_gboost_full = metrics.f1_score(tsfresh_fs_full_target_test, gboost_predict_full, average = 'macro')
f1_gboost_sel = metrics.f1_score(tsfresh_fs_full_target_test, gboost_predict_sel, average = 'macro')
f1_gboost_top = metrics.f1_score(tsfresh_fs_full_target_test, gboost_predict_top, average = 'macro')


### END ###
```

C Paper

Journal Title
XX(X):1–14

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Feature Extraction and Visualization for Damage Detection on Adhesive Joints, Utilizing Lamb Waves and Supervised Machine Learning Algorithms

Vasco F.F. Loreiro[1] and Gabriel M. F. Ramalho[1] and A. Francisco G. Tenreiro[2] and António M. Lopes[1,2] and Lucas F. M. da Silva[1,2]

Abstract

In recent years, the aeronautical industry has grown with new materials and fabrication methods being developed. This has caused the use of Structural Health Monitoring methods, more specifically Lamb Waves, to become more prevalent as regulations become more severe. Methods utilizing time series sensor data for damage detection have shown great promise in classifying the extent of damage present in plate structures when used in union with machine learning algorithms. Despite this success, there is still a lack of a robust method for choosing features that optimize the learning process to classify any damage.

In this paper a powerful time series specialized feature extraction method is implemented. Initially over 75 different prominent features and their variations are extracted from each sensor's raw data. Then, by utilizing the Benjamini-Hochberg procedure some are selected as relevant for a damage classification problem. After the initial selection, the features are inserted into supervised machine learning methods, such as Random Forest and Naïve Bayes classifiers, where not only is it possible to achieve high classification metrics using all features, but also reveal and isolate which features allow the best differentiation of each damage class. This selection methodology accounts for robustness by utilizing different layers of selection and classification, validating the feature relevance in relation to the appropriate set of classes. As such, different damage types and ranges can be utilized in this multi-class classification pipeline.

Keywords

Lamb Waves, Adhesive Joint, Weak Adhesion, Structural Health Monitoring, Nondestructive Testing

Introduction

Research on LW - an elastic disturbance that propagates on thin plate structures with shallow to no curvature - and their relevance to this application has been carried out throughout the last few years, in a collective effort to increase the reliability, integrity and durability of adhesive bonded structures (1), which are present and growing in popularity in different industries, such as the aeronautical and automotive industries/sectors, due to their potential for reduction of weight and cost in relation to other joining methods, such as screwed, riveted or welded connections (2). As component reliability and performance is paramount in these fields, it is necessary to be able to identify and locate defects without causing the destruction of the structure or component under test/examination/inspection, which ultimately is the aim of all NDT (3).

LW are a prime candidate for NDT, since they propagate over long distances with minimal attenuation and have the capability of interacting with various types of material discontinuities and defects (4). However, the intrinsic nature of LW propagation makes the interpretation of their characteristics more difficult, since they invariably excite more than one propagation mode at any given testing frequency (3), and interaction with defects present in the medium results in very complicated time-based response signals arising from the sensors. Hence, the opportunity to apply Machine Learning (ML) to the time series signals should be explored, by extracting meaningful features and inputting the information into classification algorithms, in order to train them for damage detection.

The paper is divided into five parts: firstly the introduction, secondly a literature review of the subjects addressed in this paper, namely LW, adhesive joints and defects associated with them and data visualization/machine learning techniques. Then an explanation of the project's structure and development, followed by the presentation and discussion of the obtained results, ending with the conclusion.

The main objective of this paper is to present the development of a machine learning pipeline designed to take raw LW response vibration signals measured in a single-lap adhesive joint, and, through feature engineering and machine learning algorithms, predict the degree of adhesion strenght in that joint.

[1]Department of Mechanical Engineering, Faculty of Engineering, University of Porto, Portugal
[2]LAETA/INEGI, Faculty of Engineering, University of Porto, Portugal

Corresponding author:
António M. Lopes, University of Porto, Portugal
Email: aml@fe.up.pt

Literature Review

This section is an overview of the topics covered in this paper, namely the theoretical support and definition of LW, a review on adhesion technology and considerations on the data visualization techniques and ML algorithms to be implemented, including the mathematical concepts and tools involved.

Lamb Waves

It was 1916 when Horace Lamb. published the article *"On Waves in an Elastic Plate"*, in which he presents his considerations on the problem of two-dimensional waves in a solid bounded by parallel planes, first approached by Lord Rayleigh in 1889 (5). In fact, both Rayleigh and Lamb mention each other on different papers on the subject of vibrations around this time (6), but in this paper in 1916, Lamb presents the equations that characterize the propagation of this type of waves, that ended up with his name.

LW techniques have proven capabilities to provide information about damage type, severity and location ever since they were first used to detect damage in 1960 by Worlton of the General Eletric Company. Since then, they have been employed in a variety of fashions, from research conducted at NASA that demonstrated the possibility of using LW to detect delamination in composite beams, to different groups at Imperial College, working to optimize the generation of directional LW, among many others (7).

Assuming an infinite solid medium, elastic waves can propagate in two basic modes: pressure (P) waves and shear (S) waves. Yet, if the medium is bounded, wave reflections occur at the boundary, giving way for more complicated wave patterns. Guided Waves are particularly interesting because they remain contained in a wave guide and can travel very large distances with little amplitude attenuation. As such, they are excellent for damage detection due to the full cross-section interrogation of the material, assuring that the wave will interact with any possible defect. The sensitivity to different defects will depend on mode type and the location of the defect in the thickness of the structure. Examples of Guided Waves are LW, and others, such as Love Waves, which travel in layered materials, or Rayleigh Waves, which travel in a constrained manner the surface (8; 9). These are actually the typical seismic waves that propagate on the surface of the earth during an earthquake.

LW in specific are a form of elastic perturbation that propagates in a solid thin plate with parallel free boundaries, but can also occur on shell-like structures with shallow curvature, and they are made up of a superposition of pressure modes, whose characteristics vary with entry angle, excitation and structural geometry (10). They have two fundamental propagation modes: Symmetric (Sn) and Anti-symmetric (An).

This symmetry or anti-symmetry happens with respect to the plate's mid-plane. Considering a plate with stress-free upper and lower surface, the outline of the equations for a LW propagation, following Giurgiutiu, et al. and Su, et al., is presented(8; 10; 11). Consider the equation of motion for an isotropic elastic medium, that describes the displacement field by satisfying Navier's displacement equation:

$$\mu\nabla^2\mathbf{u} + (\lambda+\mu)\nabla\nabla \bullet \mathbf{u} = \rho\frac{\partial^2\mathbf{u}}{\partial t^2} \tag{1}$$

where λ and μ are the Lamé constants, which are two material-dependent quantities that arise from the study of elastic stress-strain relationships, ρ is the mass density, and $\mathbf{u}$ is the displacement vector, given by:

$$\mathbf{u} = \nabla\Phi + \nabla\times\Psi \tag{2}$$

where Φ and Ψ are the potential functions.

The wave equations can be written as a function of this potential functions, the mass density, the Lamé constants, and both the longitudinal (L) wavespeed, given by $c_L^2 = (\lambda+2\mu)/\rho$, and the transversal (T) wavespeed, given by $c_T^2 = \mu/\rho$:

$$\begin{aligned}\frac{\partial^2\Phi}{\partial x^2} + \frac{\partial^2\Phi}{\partial y^2} + \frac{\omega^2}{c_L^2}\Phi = 0\\ \frac{\partial^2\Psi}{\partial x^2} + \frac{\partial^2\Psi}{\partial y^2} + \frac{\omega^2}{c_T^2}\Psi = 0\end{aligned} \tag{3}$$

The time dependency for these waves is assumed harmonic, in the form $e^{i\omega t}$, bringing the general solution to equation (3) as:

$$\begin{aligned}\Phi &= (A_1\sin py + A_2\cos py)e^{i(\xi x-\omega t)}\\ \Psi &= (B_1\sin qy + B_2\cos qy)e^{i(\xi x-\omega t)}\end{aligned} \tag{4}$$

where $\xi = \omega/c$ is the wavenumber and:

$$p^2 = \frac{\omega^2}{c_L^2} - \xi^2 \quad , \quad q^2 = \frac{\omega^2}{c_T^2} - \xi^2 \tag{5}$$

The four integration constants, A_1, A_2, B_1, B_2 are to be obtained from the boundary conditions. Getting the relations between the potential functions and the displacements, stresses and strains:

$$u_x = \frac{\partial\Phi}{\partial x} + \frac{\partial\Psi}{\partial y} \tag{6a}$$

$$\tau_{yx} = \mu\left(2\frac{\partial^2\Phi}{\partial x\partial y} - \frac{\partial^2\Psi}{\partial x^2} + \frac{\partial^2\Psi}{\partial y^2}\right) \tag{6b}$$

$$u_y = \frac{\partial\Phi}{\partial y} + \frac{\partial\Psi}{\partial x} \tag{6c}$$

$$\tau_{yy} = \lambda\left(\frac{\partial^2\Phi}{\partial x^2} + \frac{\partial^2\Phi}{\partial y^2}\right) + 2\mu\left(\frac{\partial^2\Phi}{\partial x^2} - \frac{\partial^2\Psi}{\partial x\partial y}\right) \tag{6d}$$

$$\epsilon_x = \frac{\partial u_x}{\partial x} \tag{6e}$$

and plugging them into the general solution equations gets:

$$\begin{aligned}u_x = [&(A_2 i\xi\cos py + B_1 q\cos qy) +\\ &(A_1 i\xi\sin py - B_2 q\sin qy)]e^{i(\xi x-\omega t)}\end{aligned} \tag{7}$$

$$\begin{aligned}u_y = [&-(A_2 p\sin py + B_1 i\xi\sin qy) +\\ &(A_1 p\cos py + B_2 i\xi\cos qy)]e^{i(\xi x-\omega t)}\end{aligned} \tag{8}$$

For free wave motion, the homogeneous solution is derived by applying the stress-free boundary conditions at the upper and lower surfaces ($y = \pm d$), where d is half of the plate thickness, obtaining the characteristic equations:

$$D_S = (\xi^2 - q^2)^2 \cos pd \sin qd + 4\xi^2 pq \sin pd \cos qd = 0 \quad (9a)$$

$$D_A = (\xi^2 - q^2)^2 \sin pd \cos qd + 4\xi^2 pq \cos pd \sin qd = 0 \quad (9b)$$

And finally, equations (9a) and (9b) can be rewritten in the more compact form as the Rayleigh-Lamb equation:

$$\frac{\tan pd}{\tan qd} = -\left[\frac{4\xi^2 pq}{(\xi^2 - q^2)^2}\right]^{\pm 1} \quad (10)$$

where the exponent +1 corresponds to symmetric (S) motion and -1 to anti-symmetric (A) motion. Equations (9a) and (9b) accept a number of eigenvalues, $\xi_0^S, \xi_1^S, \xi_2^S, ...,$ and $\xi_0^A, \xi_1^A, \xi_2^A, ...,$ respectively. Each parameter corresponds to a set of eigencoefficients (A_2, B_1) for the symmetric case and (A_1, B_2) for the anti-symmetric one, that can be plugged into equations (7) and (8) and yield the corresponding modes: $S_0, S_1, S_2, ..., S_n$ and $A_0, A_1, A_2, ..., A_n$.

The coefficients p and q in equations (9a) and (9b) are dependant on the angular frequency, ω, and, consequently, the eigenvalues ξ_i^S and ξ_i^A will change accordingly, and since the wavespeeds correspond to $c_i = \omega/\xi_i$, they will also change with frequency, and this change produces the so-called wave dispersion.

LW are highly dispersive, meaning that the fundamental way to concretely describe their propagation in a material is through their dispersion curves, that plot the phase and group velocities against the excitation frequency (often shown as a product with thickness), since, for each frequency-thickness product, and each solution of the Rayleigh-Lamb equation, one finds a corresponding LW mode (7; 11).

A proper LW mode for damage detection should feature non-dispersion, low attenuation, high sensitivity, easy excitability and good detectability. The best way to prevent wave dispersal is to have an input signal with a narrow bandwidth, such as a windowed toneburst which is a more frequently adopted input signal.

The generation of LW can be done through a variety of instruments, roughly grouped under five categories, ultrasonic probes, laser, interdigital transducers, optical fibre, piezoeletric elements (10). All of these methods have strengths and weaknesses. Piezoeletric, or lead zirconate titanate (PZT) elements have advantages, since they can be used for both LW generation and acquisition, delivering excellent performance, allied to their neglectable mass/volume, effortless integration, outstanding mechanical strength, wide frequency response range, low power consumption and acoustic impedance, and low cost, making them particularly suitable for SHM applications as an in-situ generator/sensor. On the other hand, some nonlinear behaviour and hysteresis under large strains/voltages, or at high temperatures should be accounted for, and their brittleness and low fatigue life may cause concerns or limit some applications. Importantly, PZT-generated LW unavoidably excite multiple modes that generate complex response signals, requiring sophisticated signal processing to successfully utilize them for damage detection (10).

Adhesive Joints and Defects

An adhesive is formally defined as a material which, when applied to surfaces, can join them together and resist separation. Adhesive is, therefore, the general term which covers materials like glue, cement, paste, among others (12). An adhesive joint is the finished connection of two surfaces, made of similar or different materials, through the use of adhesives. The materials that are bonded by the adhesive, before bonding, are called substrates, and after bonding, they are referred to as adherends (12). The region linking the adherend and the adhesive is the interphase, whose chemical and physical characteristics critically influence the mechanical properties of the adhesive bond itself. Not to be confused with the interface also known as the boundary layer, that resides within the interphase (where various interfaces connecting different materials can exist), which is the plane defined by the contact between the surface and the two materials, where, during the formation stages of the bond, the intimate molecular contact is created (13).

Adhesive joints provide a wide variety of advantages when compared to conventional mechanical fasteners, according to (13): more uniform stress distribution along the bonded area, that leads to a good resistance to dynamic solicitation as well as load transmission and higher stiffness, as well as the reduction of the weight on the structure and consequently, the cost; The possibility of automating the adhesive application is also appealing.

However, some disadvantages should also be considered, like the need to avoid peel and cleavage stress because they create a load concentration in a small area, resulting in low strength in that area, limited resistance to extreme environmental conditions, such as high temperature and humidity, and the difficult quality control, since there is no expeditious way to assess the integrity of the bond - that could change with non-destructive testing techniques (12; 13).

Even if a certain adhesive is the perfect choice for a given application, its durability is a function of the entire bonding system, that is, adhesion defects and other problems can still arise and lead the joint to failures of different types when subjected to mechanical stress, be it because of the lack of proper surface preparation, bad chemical formulation of the adhesive, incomplete curing cycle or other environmental disturbances, such as high humidity.

Structural bonds, in particular, are expected to undergo some form of loading for a significant part of their service life, and so, an understanding of the bond failure and failure modes is critically important. It can occur in a number of places within a bonded joint or structure, as reported by (13; 14), cohesively within the adherends, interfacially between the adherend and the adhesive layer or cohesively within the adhesive. The bond might also contain "isolated phenomena" that compromises its structural integrity, such as voids, debonds, porosity or cracks within the adhesive

layer, and these should be detected as soon as possible to avoid serious damage or failure of the bond.

Data Visualization Techniques

Data visualization regards the manipulation of sampled and computed data for comprehensive display, conducive to a deeper understanding of the data and highlighting the underlying physical laws and properties. It is an important tool that helps the data scientist in different stages of a project, by pointing out useless, incorrect or inconsistent information that can be eliminated, and assessing the results, to expose dynamics and help define the focus points and regions of interest (15).

Dimensionality reduction is applied as a procedure to create projections of the data, mapping it into a low dimensional space, in order to enable an insightful view of how data instances relate to each other, namely the appearance of structures such as high local density, interesting relations between observations and the presence of clusters (16; 17).

Time-oriented data visualization is a widely researched topic, and according to Aigner, et al. (18), visualization techniques can be categorized in many ways, such as the structure of time itself (linear, cyclic or branched), the frame of reference (spatial or abstract), the number of variables that are time-dependant, and the dimensionality that is to be obtained (2D or 3D, since those are intuitively understood by the human brain).

It is a common procedure in data science to transform the raw data formats into more suitable and consistent ones before applying any further techniques, through processes like smoothing, generalization and normalization. Normalization is likely to improve accuracy and efficiency of classification algorithms, and for distance-based methods in particular, it prevents attributes with initially large value ranges from outweighing those with smaller ranges, uniforming the importance of any given testing instance, while keeping the intrinsic information intact. Among others, data normalization methods include Min-Max normalization, Z-Score normalization, l^2 normalization and normalization by decimal scaling (19; 20).

Min-Max normalization performs a linear transformation on the original data. If an attribute *A* has a range of values of $[a_{min}, a_{max}]$, Min-Max normalization maps a value x of *A*, to its counterpart *x'* in the new predetermined range $[a'_{min}, a'_{max}]$ by computing: (19)

$$x' = \frac{x - a_{min}}{a_{max} - a_{min}} \times (a'_{max} - a'_{min}) + a'_{min} \qquad (11)$$

where, in this case, the new selected range was $[a'_{min}, a'_{max}] = [0, 1]$.

Two of the most widespread visualization techniques, Multidimensional Scaling (MDS), and t-distributed Stochastic Neighbor Embedding (t-SNE), are implemented for the projection and visualization of the time series themselves and also to visualize some features later on.

Multidimensional Scaling (MDS) MDS is a multivariate statistical method that represents measurements of proximity/similarity (or dissimilarity) among pairs of objects geometrically, as distances between points of a low-dimensional space. Torgerson proposed the first MDS method and coined its name, Multidimensional Scaling, that can also be known as Principal Coordinates Analysis (PCoA) or even Torgerson Scaling (21).

MDS shows the correlations among instances of the data, displaying each of these as a point on a plane, so that the closer together the points are, the more correlated the respective instances are, turning the data from immense arrays of numbers to an accessible visual representation, for easy inspection and exploration (22).

The application of MDS is exploratory, so as to uncover the data elements accounting for the proximity of the data, rather than to test *a priori* hypothesis about the existence and properties of those elements (23). The vastly used family of procedures known as *Principal Components Analysis* (PCA), which should not be mistaken with PCoA, is closely related to MDS in function, but differs in some key aspects, the principal being the fact that MDS starts with a matrix of similarities between a set of individuals, while PCA starts directly with the initial data matrix (24; 25).

MDS models are defined by the similarity or dissimilarity of data - the proximity indexes p_{ij} between pairs (i, j) of objects, that construct an $n \times n$ matrix $\mathbf{C}$, being *n* the total number of objects - and how those proximity indexes are mapped into distances of an *m*-dimensional MDS space configuration: $\mathbf{X}$. In classic MDS, $\mathbf{C}$ is symmetric, with $p_{ij} > 0$ for $i \neq j$ and its main diagonal is composed of "1" (17).

The mapping is given by a representation function $f(p_{ij})$, that specifies how the proximities should be related to the distances $d_{ij}(\mathbf{X})$, seeking the configuration (in a given dimensionality *m*) whose distances satisfy f as closely as possible. The condition "as closely as possible" is quantified by a badness-of-fit measure, also called a *loss function* - an expression that aggregates the representation errors : $e_{ij} = f(p_{ij}) - d_{ij}(\mathbf{X})$. The most common loss function in MDS is named raw-Stress (ρ), also known as "Kruskal stress" (24). It varies between 0 and 1, with values near 0 evidently indicating a better fit (22; 26).

$$\rho = [f(p_{ij}) - d_{ij}]^2, i, j = 1, ..., n \qquad (12)$$

Shepard plots are also used to evaluate the fit of the mapping, by comparing d_{ij} versus p_{ij} for a given value of *m*. A narrow scatter of points, resembling a smooth straight line without sudden steps indicates a successful representation (22; 24; 26).

The *m*-dimensional MDS space configuration always refers to a coordinate system, customarily a set of *m* directed axes, perpendicular to each other and intersecting in one point, the origin. If the value for *m* is chosen to be 2, for example, this would define a Cartesian plane. Since the MDS interpretation is based on the emerging clusters and distances between points in the mapping, rather than on their absolute coordinates, the units of the axes are meaningless, and so, the MDS map can be rotated and translated, as the distances between points remain the same (17; 22). As for the distances' definition, the most natural and frequently used is the Euclidean distance, corresponding to the length of the straight line segment connecting *i* and *j*, computed by the formula (22):

$$d_{ij}(\mathbf{X}) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2} \quad (13)$$

Thus, $d_{ij}(\mathbf{X})$ equals the square root of the sum of the intradimensional differences $x_{ia} - x_{ja}$, which, in plain terms, is the Pythagorean theorem for the length of the hypotenuse of a right triangle. Generalizing to the *m*-dimensional case, yields:

$$d_{ij}(\mathbf{X}) = \left[\sum_{a=1}^{m}(x_{ia} - x_{ja})^2\right]^{1/2} \quad (14)$$

The dissimilarity matrix $\mathbf{C}$ can adopt different measures, such as the so-called Canberra and Manhattan distances, among others. It should be noted that the use of alternative measure methods within the MDS is a common procedure, often having distinct representations of the same data set, that view phenomena with different perspectives, opening up the possibility of choosing the MDS charts that yield better visualizations (26).

t-distributed Stochastic Neighbor Embedding Stochastic Neighbor Embedding (SNE), as originally presented by Hinton and Roweis (27), is a probabilistic approach to the task of placing objects, from high-dimensional vectors or by pairwise dissimilarities, in a low-dimensional space in such a way that preserves neighbor identities. It serves the same purpose as MDS, dimensionality reduction, but unlike MDS, SNE makes use of a Gaussian distribution, centered on each object of the high dimensional space, and the densities under this distribution are used to define a probability distribution over all the potential neighbors of the object (24; 27).

Briefly, it starts by converting the high-dimensional Euclidean distances between datapoints, x_j and x_i into conditional probabilities $p_{j|i}$, representing the probability that x_i would pick x_j as its neighbor, if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i. This probability will be high for nearby points, and almost infinitesimal for widely separated ones.

$$p_{j|i} = \frac{\exp\left(-||x_i - x_j||^2/2\sigma_i^2\right)}{\sum_{k\neq i}\exp\left(-||x_i - x_k||^2/\sigma_i^2\right)} \quad (15)$$

where σ_i is the variance of the Gaussian distribution centered around x_i. Now for their low-dimensional counterparts, y_i and y_j, a similar conditional probability is computed (but with a fixed variance), denoted by $q_{j|i}$. If these conditional probabilities are equal, in other words, if the distributions are matched, the model is correctly mapping the data (27; 28).

So, the aim of the embedding is to match the distributions as well as possible, which is also achieved by minimizing a cost function. A natural measure of the faithfulness with which the probabilities match is the Kullback-Leibler divergence. Therefore a cost function *C* is defined by a sum of the Kullback-Leibler divergences between the original ($p_{j|i}$) and the induced ($q_{j|i}$) distributions over neighbors for each object. SNE minimizes *C* using a gradient descent method (27; 28).

$$C = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \quad (16)$$

The t-SNE is a variation of SNE, with two important differences, which aim to solve some shortcomings of the SNE method (27; 28).

Firstly, since the Kullback-Leibler divergence is not symmetric, different types of error in the pairwise distances present in the low-dimensional map are not weighted equally. For instance, there is a large cost for using widely spaced map points to represent nearby datapoints, but a small cost for using neighboring map points to represent datapoints. In t-SNE, conditions to this function are applied to ensure that it is symmetrized, eliminating that problem. It is said to be symmetric because it has the property that $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$, $\forall i, j$, hereby named joint probabilities. The new cost function C comes:

$$C = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \quad (17)$$

and this cost function is minimized by the gradient descent method given by:

$$Cy_i = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1} \quad (18)$$

The issue known as the "Crowding Problem", lies in the fact that if small distances are to be accurately mapped, the moderately distanced datapoints will be placed much too far away in the representation. As a solution, in t-SNE, instead of using a Gaussian distribution, it is swapped by the heavy-tailed Student t-distribution with one degree of freedom (also known as Cauchy distribution) on the low-dimensional map, q_{ij}. The mismatched tails on this distribution compensate the mismatched dimensionalities on the data.

There are various parameters that can be tinkered with in a t-SNE application to optimize the cost function gradient descent, some of which are the number of iterations, early exaggeration, learning rate and perplexity, interpreted as a smooth measure of the effective number of neighbors; so adjusting all these parameters yields different results even for the same data set (28).

Machine Learning and Features

ML is the subfield of AI that gives "computers the ability to learn without being explicitly programmed" (29). Evolved from the study of pattern recognition, ML explores the construction and optimization of algorithms that can learn from, and make predictions on data. They operate by building and improving models from sample inputs, making data-driven decisions instead of following strict program instructions. It is closely related to (and often overlaps with) computational statistics, making regular use of mathematical and statistical methods.

Even with the powerful tools that ML has to offer, it is difficult for ML algorithms to produce proper results if the input is not preprocessed. There could exist outliers that need to be discarded, missing values on tests that should be eliminated or appended with accordingly interpolated values (30). This preprocessing is fundamental, even more so if the data is obtained from real life sensors, that might contain noise and interference effects that should be minimized, or malfunctions that should be accounted for. Such is not the case in the present project, as all the data is obtained from

simulations, so the preprocessing stage has a lighter overall significance.

If a large number of observations are registered, the whole data set turns into a high-dimensional one. It is important to reduce this dimensionality in order to handle the data adequately, and use it successfully in ML algorithms. Ideally, the reduced representation should have a dimensionality that corresponds to the intrinsic dimensionality of the data, the minimum number of parameters needed to account for the observed properties (31).

However, one needs to differentiate between the number of cases (observations) in a large data set, and the number of variables available for each case - these variables comprise specific information about the data and are referred to as *"features"*. These features are not present in the original measurements, but signal processing tools can extract these from the raw data, enabling a more efficient use of ML algorithms. This procedure can be viewed as "nontrivial extraction of implicit, previously unknown and potentially useful information from data, or the search for relationships and global patterns that exist in databases" (32).

Feature Engineering plays a vital role in ML algorithms and big data analytics, encompassing the generation, extraction, transformation, selection, analysis and evaluation of features. Indeed, little can be achieved if there is a short amount of features to represent the underlying data objects, and the quality of the results obtained in those algorithms will reflect the quality of the available features themselves (33; 34).

Feature Engineering is often data specific and application dependent, which means that different data types - text, images, streaming data, social media data - require specialized techniques (33).

It is useful to distinguish three main types of features, besides simple **Statistical Features** (which can also be significant), based on the domain their information is in, namely **Time Domain**, **Frequency Domain** and **Time-Frequency Domain**:

Statistical features are the simplest form of features, obtained directly from the signal by statistical analysis, and represent basic information about the data. Some examples are the Mean, Variance, Standard Deviation, Skewness, Kurtosis and higher order moments, and even Maximum and Minimum values (Peaks). Even though they do not result from complex analysis and advanced methods, their information can be valuable (35).

Time domain features, are typically used to predict future values for signals using time-based models like the Autoregressive (AR), Integrated (I) and Moving Average (MA) models, that can be fused together in the ARIMA model first introduced by Box and Jenkins (36). However, some orders/parameters of these models can be used as a feature in itself, as well as the Auto-Correlation dimensions of the signal (where it is compared to itself with a small delay). The Auto-Correlation coefficients produced a by the vibration of a healthy structure could be different than those from a faulty one. From the ML algorithms viewpoint, these features are just arrays of organized information to be used in a specific task, with no connection to the time-based models that originated them, or their purpose;

Frequency domain features are based on the frequency domain analysis, which is is arguably the most far-reaching set of mathematical tools utilized in engineering, specially signal analysis, and its cornerstone is the Fourier Transform (FT). It is based on the premise that every function - no matter how complex it looks - can be decomposed into a sum of simpler functions, a concept proposed by Joseph Fourier in 1822 (37; 38).

Taking $f(t)$ as a time-dependant input signal, that may be composed by harmonic and/or periodic elements, its Fourier Transform $F(\omega)$ is called the signal's *spectrum*, and can be viewed as the frequency response - a transformation of the time signal into a sum of basis functions (sinusoidal) of various frequencies, which the original signal contains as periodic components, where (37; 39; 40):

$$F(\omega) = \int_{t=-\infty}^{\infty} f(t) e^{-j\omega t}\, dt \tag{19}$$

with $j = \sqrt{-1}$ and the complex exponentials as the sinusoids: $e^{j\theta} = \cos\theta + j\sin\theta$. A Fourier Transform pair is often written $f(t) \leftrightarrow F(\omega)$. The Inverse Fourier Transform is also applicable, changing the content from the frequency to the time domain:

$$f(t) = \frac{1}{2\pi} \int_{\omega=-\infty}^{\infty} F(\omega) e^{j\omega t}\, d\omega \tag{20}$$

$F(\omega)$, which is written as a complex number in terms of its magnitude and phase, ultimately tells how much content the original signal has at any frequency ω;

The necessity for **Time-Frequency domain features** arises from the fact the the Fourier Transform is of great use to transform a signal into the frequency domain, where it has great resolution, at the expense of the time domain information, as it contains none. In other words, it is known at which frequencies the signal oscillates, but not at which time these oscillations occur. So if a signal has a dynamic frequency spectrum, i.e. the frequency content changing over time or frequencies appearing abruptly for a short period of time, the Fourier Transform will not expose them. For that, the Time-Frequency domain is approached, specially through the Short-Time Fourier Transform (STFT), the Wavelet Transform or the Hilbert-Huang Transform, among others (37).

The STFT can be used as a way of quantifying the change of a non-stationary signal's frequency and phase content over time, by dividing a time-based signal into shorter segments of equal length and then computing the Fourier Transform in each of those segments separately, therefore keeping in the results of the magnitude and phase content for each point in time. The use the Fast Fourier Transform on these segments yields the discrete-time STFT, expressed as:

$$STFT_{x(k)}(m,n) = \sum_{k=0}^{L-1} x(k) g(k-m) e^{-j2\pi nk/L} \tag{21}$$

where $x(k)$ denotes the discrete time signal, that is multiplied by $g(k)$, an L-point window function with a fixed size n, that will divide the signal into chunks as it slides through the signal with m amount of shift. The FFT is then applied

to each of these chunks to compute the STFT (41; 42). The Hamming window, a smooth, "bell-shaped" curve is a popular choice for the window function employed. The STFT can also be called a Gabor transform, if the window used is a Gaussian function (43).

The STFT is utilized to construct what is called the Spectrogram, a 2-D visualization that plots, in a logarithmic scale, the normalized, squared magnitude of the STFT coefficients (41; 44).

Machine Learning pipeline

Acquisition of simulation data

LW based NDT methods can detect incipient damage, often unnoticed by other techniques, with the help of suitable electric signals applied to a PZT actuator which induces the propagation of this type of waves on the structure or specimen. Other PZT sensors, placed strategically in the specimen, measure the vibrations and output electric signals, with distinct amplitude and phase from the input, and whose characteristics will depend on the existence and type of damage in the structure.

Data processing algorithms based on ML require large volumes of data to be trained and, therefore, it would be impracticable to use experimental data in their development. The alternative is to use simulation data from numerical models, generated with finite element method (FEM) software - ABAQUS. For the purpose of this project, an adhesive joint simulated specimen was modeled.

An adhesive joint among two aluminium plates, with a single-lap joint design with varying degrees of adhesion strength on each test (ranging from 600 to 270000 kPa), with 1 PZT actuator on the top plate and 1 PZT sensor on the lower plate. The aluminium plate has a density $\rho = 2500\ \mathrm{kg/m^3}$, Poisson ratio $\nu = 0.33$ and Young modulus $E = 72.4$ GPa. The chosen adhesive to simulate was a 0.2 mm layer of Nagase T-836/R-810, as it has great potential for industrial applications, specifically aeronautical and automotive. The simulated adhesive joint is presented on Figure 1.

This model was run 900 times, and since it measures only one sensor's displacement as a function of time, accounts for 900 time series. All signals obtained were made uniform as a vector of equal dimension, achieved by interpolating the whole original array of values, imposing a fixed time step of 5000 to fit an array of predetermined size. The simulation ran with a fixed time of 0.5 ms, as this is enough time to have the waves interact with the defects / adhesive joint and arrive at the sensor without receiving too many reflected waves from the opposite wall. The excitation signal chosen was a 5-cycle Hanning-windowed sinusoidal tone burst with the central frequency of 100 KHz, applied to the PZT elements.

As an example, two signals from different tests are shown in Figure 2, and 3, a test with a higher adhesion strength in the joint:

In Figure 2 only the top 10 peaks were marked directly on the time series plot, but the values of all the local minima and maxima of the entire domain are saved in an array, for each of the tests - a statistical feature.

It is perceptible that the second signal comes from a stronger, firmer and more consistent medium, as the LW

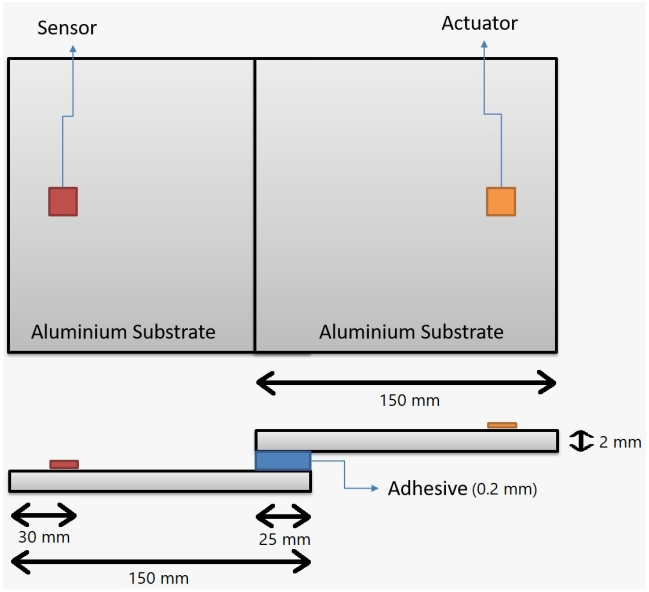


Figure 1. Model of the adhesive joint simulation setup.

propagates and arrives at the sensor more smoothly, creating a more balanced response. The average displacement measured on the sensor is higher, and all of the 10 peaks appear above the 1×10^{-5} m mark, while on the first test, the single highest peak falls short of that mark, being the only one that surpasses 0.75×10^{-5} m.

Automatic Feature Extraction

Viewed as one of the general topics of feature engineering, Automatic Feature Extraction is a methodology capable of automatically generating a large number of features from a data set and subsequently selecting the most effective subset of these features to be applied in the ML algorithms. The best way to differentiate the information within the signals is to have the highest possible number of features being initially extracted from the time series, in all the domains discussed in the previous chapter, and then choosing the best, most meaningful and adequate subset to the problem at hand, so that ML algorithms can use it in the most effective way possible to detect damage (33).

The automation of the feature extraction process was key, and for that effect, **tsfresh** - **T**ime **S**eries **F**eatu**R**e **E**xtraction on basis of **S**calable **H**ypothesis - a Python package developed by Christ et. al (45), was implemented, comprised of two main steps: the **feature extraction** and **feature selection**.

The first step is the feature extraction from the time series, resulting in a $M \times N$ matrix, where the M rows correspond to the time series, identified by their **id**, while each of the N columns correspond to the extracted features, characterized by the kind of time series (which sensor) that originated the data, followed by the feature calculator itself and lastly the key-value pairs of parameters configuring the respective feature calculator. A sample of the feature matrix is shown on Figure 4:

The second step, which can be run separately, in parallel with any other ongoing feature extraction thread, is the estimation of each feature's relevance to a given ML task, through hypothesis tests and calculation of the respective

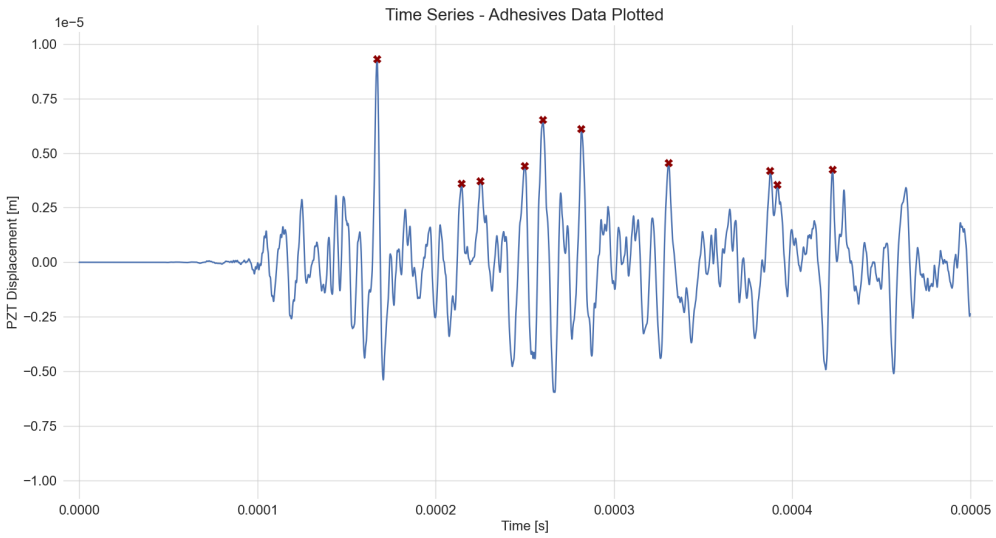


Figure 2. Representation of an adhesive joint test test, with a signal from the PZT Sensor and the top 10 peaks marked. Adhesion strength = 900 kPa.

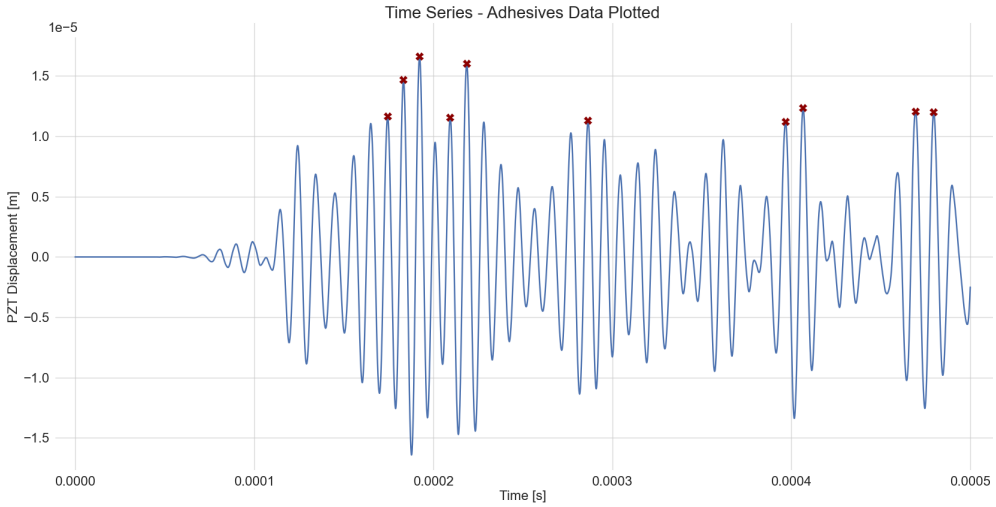


Figure 3. Representation of an adhesive joint test test, with a signal from the PZT Sensor and the top 10 peaks marked. Adhesion strength = 50100 kPa.

ID	zt1__fft_coefficient__attr_"angle"__coeff_64	zt2__change_quantiles__f_agg_"mean"__isabs_True__qh_0.6__ql_0.4	zt2__fft_coefficient__attr_"angle"__coeff_25	zt3__benford_correlation
0	-165.021	5.35231e-08	162.193	0.980251
1	-166.597	6.29235e-08	-159.883	0.97646
2	164.847	4.48842e-08	67.9614	0.965603
3	-171.061	6.10006e-08	158.88	0.969739
4	159.306	7.38881e-08	171.275	0.962898
5	158.972	9.23835e-08	133.348	0.912316
6	-173.121	5.3202e-08	155.487	0.975048
7	159.821	4.74425e-08	87.5884	0.91597
8	154.682	6.82938e-08	143.163	0.955324
9	161.466	4.52758e-08	102.138	0.964703
10	143.166	5.80039e-08	-152.86	0.959445
11	-69.0983	6.33425e-08	167.786	0.951394
12	-178.289	4.77109e-08	152.026	0.979217

Figure 4. **tsfresh** feature matrix sample.

p-values. The hypothesis tests are automatically configured depending on the type of supervised ML problem (classification/regression) and feature type (categorical/continuous). It also involves a multiple testing procedure, utilizing an instrument called *Benjamini-Hochberg Method*, which is designed to control the false discovery rate (FDR). This feature selection step is crucial, as the effectiveness of a feature is ultimately measured in terms of its performance to the ML task at hand, and whether or not it improves its metrics, in other words, whether or not the feature has relevance to the problem at hand (33).

Table 1. tsfresh classification-task oriented feature filtering steps results

	Adhesive Joint				
Classes	1	2	3	4	5
Extracted features	**788**				
Features with p-value ≤ 0.05	681	636	599	598	590
Benjamini-Hochberg Selected features	657	559	555	571	571
Top features	**244**				

Through this selection method, the pool of extracted features considered relevant by the univariate hypothesis testing (those that scored p-values lower or equal to 0.05) is filtered, leaving only the so called "selected features", which have remained relevant for the classification task of at least one class. Table 1 shows the feature filtering process results. The "Top" features are those considered relevant for all classes simultaneously. It is fair to mention that this automated feature selection module of **tsfresh** is just as appropriate and decisive as the feature extraction module itself. They work together seamlessly and elegantly, allowing for the whole direct data processing, from preprocessed time series to finely selected, ML-task oriented features.

Results and Discussion

Visualization of the data / Features

The results of the application of MDS and t-SNE is now presented, both to the signals' original time series and the extracted/selected features.

The chart in Figure 5 represents the MDS applied to the time series themselves. The absence of legend is due to the fact that the results are not shown per class, but directly using the adhesion strength value, from the weakest adhesion being the deepest blue to the strongest adhesion in the dark red, yielding the direct realization that the LW propagation signals are as closely related as the adhesion strength itself.

Figure 6 presents the t-SNE mapping of the data. It follows the color map convention used on the MDS visualization, and the result has the same overall character, with a steady evolution of distances following the adhesion strength increase in a smother trend than with the results obtained using MDS.

For the visualization of features, considering that the adhesion strength value varies from 600 to 270000 kPa, five classes were arbitrarily assumed, by dividing that range in five equal intervals, intended to illustrate qualitatively the data: **1** - Very Low Adhesion Force; **2** - Low Adhesion force; **3** - Medium Adhesion force; **4** - High Adhesion force; **5** - Very High Adhesion force; The target vectors are then composed by the corresponding time series id, that should be consistent with the features id, of its respective class.

In Figure 7, a feature calculated from a linear least-squares regression of the time series values, is represented in MDS mapping. The differentiation among classes is evident here, with the sole exception of class number 2 (low adhesion strength), that does not stand on its own, isolated from the others.

Supervised Machine Learning - Classification Algorithms

Naïve Bayes For each feature **x**, the Naïve Bayes approach to classification is to formulate a probabilistic model that estimates the posterior probability, $P(y|\mathbf{x})$, of the different classes, y, and to predict the one with the largest posterior probability. According to Bayes' Theorem, one has that:

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} \tag{22}$$

where P(y) can be estimated by counting the proportion of class y in the training set, and P(**x**) can be ignored since we are comparing different y on the same **x**; thus, the only component that needs to be considered is $P(\mathbf{x}|y)$. If an accurate estimate for $P(\mathbf{x}|y)$ is obtained, that will correspond to the **Bayes optimal classifier**, with the smallest theoretical error rate. Estimating $P(\mathbf{x}|y)$ is not straightforward, given that it involves the estimation of exponential numbers of joint-probabilities of the features. That can be avoided by assuming that, in each class label, the n features are independent of each other, and so we can estimate the conditional probability by:

$$P(\mathbf{x}|y) = \prod_{i=1}^{n} P(x_i|y) \tag{23}$$

In the training stage, these probabilities are estimated, and then in the testing stage, a feature **x** will be predicted as label y, if y leads to the largest value of:

$$P(y|\mathbf{x}) \propto P(y) \prod_{i=1}^{n} P(x_i|y) \tag{24}$$

The different Naïve Bayes classifiers differ mainly in the assumptions made regarding the distribution of $P(x_i|y)$. In this case, a Normal distribution is appointed (46).

k-Nearest Neighbors In the **k-Nearest Neighbors** (KNN) classification algorithm, one relies on the simple principle that objects similar in the input space will also be similar in the output space. It is considered a *lazy learning* approach, known as *instance based* or *non-generalizing learning*, as it does not have an explicit training process nor does it attempt to construct a general internal model. It merely stores the instances of the training set. In training, the KNN model will pick out the k instances from the training set that are closer to the test instance. Then, for classification, the test instance will be classified to the majority class among the k nearest instances, with k being a value specified by the user. The basic KNN uses uniform weights: the value assigned to a query point is computed from a simple majority vote of the nearest neighbors. Under some circumstances, it is better to weigh the neighbors such that the nearer neighbors contribute more to the fit (46; 47).

Ensemble methods Ensemble methods are state-of-the-art ML approaches that train multiple learners to solve the same problem and combine their results, significantly improving the accuracy compared to a single learner model. Ensemble learning is also called *committee-based learning* or *multiple classifier systems*. They are constructed in two

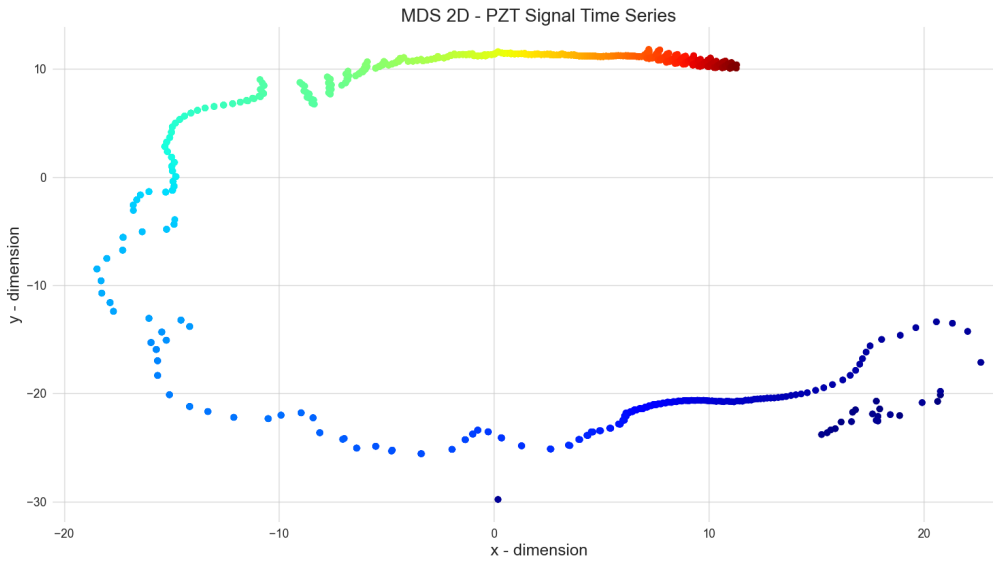


Figure 5. 2-D MDS Visualization of the adhesive joint simulation tests.

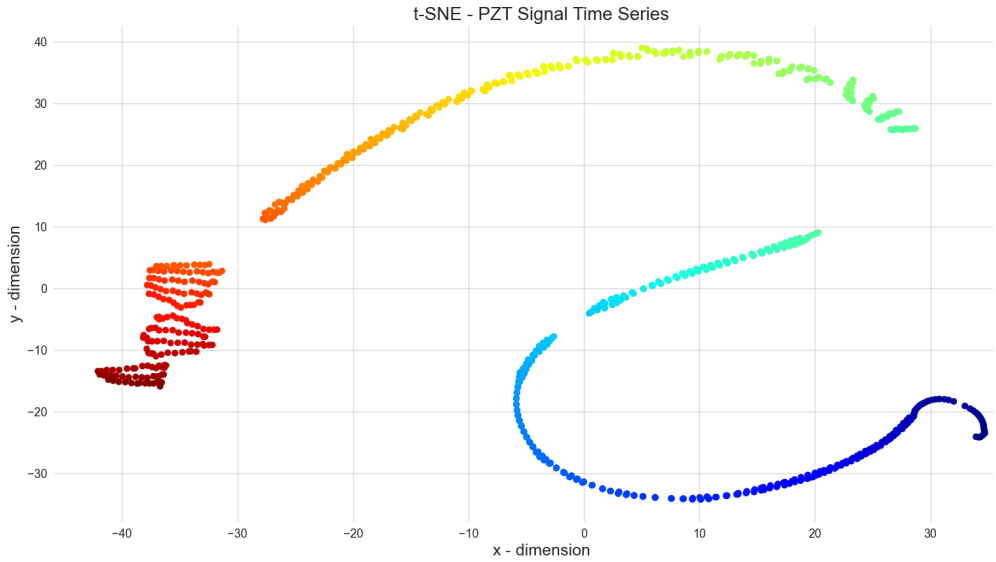


Figure 6. 2-D t-SNE Visualization of the adhesive joint simulation tests.

steps: generating the base models, that should be as accurate and diverse as possible, and then averaging their results (46).

The ensemble contains a number of learners called **base learners**, generated from the training data by a base algorithm, which can be decision trees, neural networks or other kinds of models. All of the three ensemble methods mentioned are based on the decision tree model, and by using a single base algorithm, they produce *homogenous* base learners, leading to **homogenous ensembles**. That implies that *heterogeneous* ensembles also exist, using various learning algorithms that get the name individual/component learners, instead of base learners (46). Before expanding on ensemble methods, defining the used base learner is in order.

A **Decision Tree** is a non-parametric model made up of tree-structured decision tests, working in a *divide-and-conquer* way, predicting the value of a target variable by learning simple decision rules inferred from the data features. It starts on the root node and from there each non-leaf node is associated with a feature test, also called a **split**. Data falling into the node will be split into subsets according to their different results on the feature test, eventually landing in a leaf node, associated with a classification label to which that instance will be assigned. **Decision Tree** learning algorithms are generally recursive processes, since, in each step, a data set is given and a split is selected, then acting as the given data set for the next step. The key of the decision tree algorithm is in the selection criteria of the splits. Typical split criteria include the "information gain" or the Gini criteria (46).

A final distinction among ensemble methods is required on how the base learners are generated, which can be divided in two paradigms (46):

– **Boosting** - Refers to a family of algorithms that convert "weak learners" to "strong learners" by having the weak models working together. Characterized by being sequential ensemble methods, where the base learners are generated in succession, exploiting the dependence between them and boosting overall performance in a residual-decreasing way. Gradient Boosting is a representative of this kind;

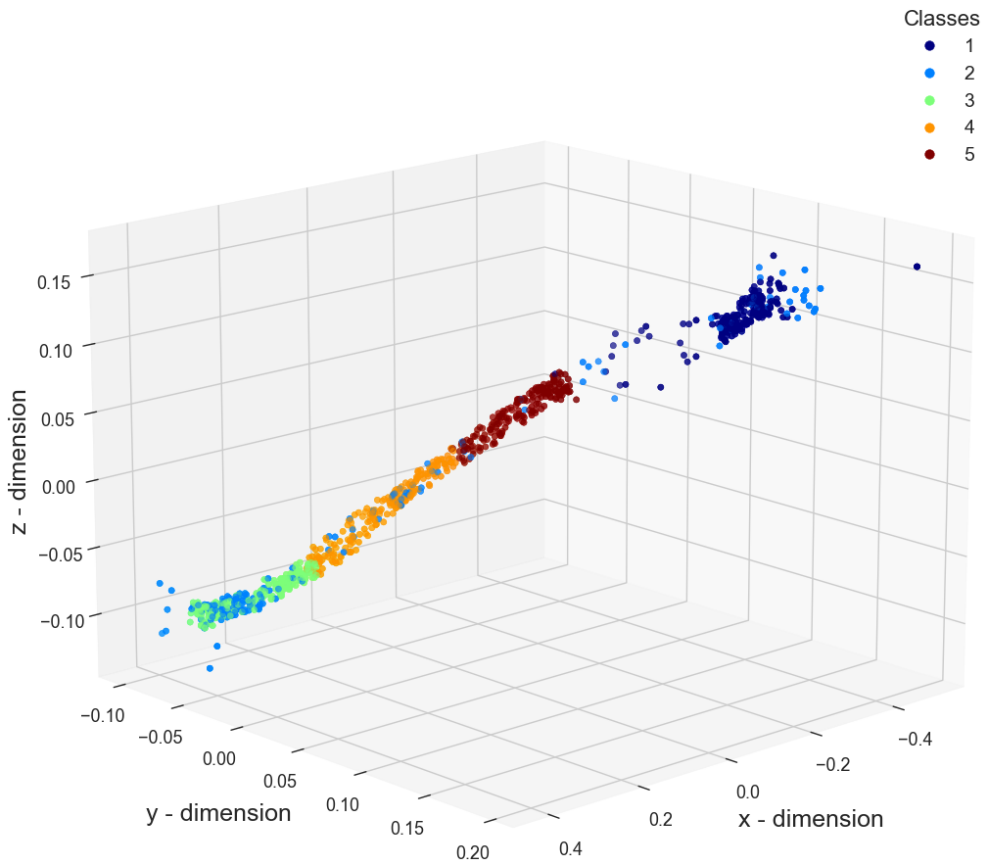


Figure 7. MDS 3-D Adhesive Joint Feature: "value_agg_linear_trend_attr_intercept_chunk_len_10_f_agg_var".

– **Bagging** - Exploits the independence between base learners by setting them up parallel to each other, reducing the error by combining them and averaging out the results. To have independent base learners, one possibility is to apply bootstrap sampling. That means training those base models with non-overlapped subsets of the training data pool. Not only does bagging improve performance, but also decreases variance and helps handling overfitting. Random Forest represents this kind.

According to Zhou (46), for many tasks, "the best off-the-shelf learning technique at present is an ensemble method such as **Random Forest**, facilitated with feature engineering which constructs/generates usually an overly large number of new features rather than simply working on the original features."

Classification Metrics and Evaluation

In ML, the algorithm's performance is presented and analysed through the results' **metrics**. Numerous metrics exist, specifically designed to evaluate certain aspects of each ML category (classification, regression, clustering, etc.), that can go from a broad assessment of the algorithms' behaviour, to very particular measurements that could be of special interest. The most mainstream and widely used classification metrics were examined, as presented by (47; 48): **Confusion Matrix**, also known as an error matrix, a confusion matrix is a square table layout that allows for the visualization of the performance of an algorithm. Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class. Evidently there are as many rows and columns as there are classes, and the diagonal represents the instances that were properly classified. The name stems from the fact that it makes it easy to see whether the system is confusing two classes (i.e. commonly mislabeling one as another). Figure 8 represents one of the confusion matrices obtained from the K-Nearest Neighbors algorithm, using the Top Features data subset. The accuracy is almost perfect, only missing the prediction of a few tests.

The **Accuracy Score (acc)** measures the ratio of correct predictions over the total number of instances evaluated:

$$acc = \frac{tp + tn}{tp + fp + tn + fn} \tag{25}$$

Precision Score (p) is the ratio of correctly predicted positive observations to the total number of positive observations. Intuitively, it is the ability of the classifier not to label as positive a sample that is negative:

$$p = \frac{tp}{tp + fp} \tag{26}$$

Recall Score (r) is the ratio of correctly predicted positive observations to all observations in actual class. Intuitively, it is the ability of the classifier to find all the positive samples:

$$r = \frac{tp}{tp + fn} \tag{27}$$

F1 Score (F1), also known as balanced F-score or F-measure, is a weighted average of recall and precision, that

is a parameter of great importance when having an uneven class distribution in data:

$$F1 = \frac{2 \times p \times r}{p + r} \tag{28}$$

Despite having a fairly balanced class distribution, the dataset is not perfectly balanced, so measuring the F1 score is valid.

The symbols *tp, fp, tn* and *fn* signify *true positive, false positive, true negative* and *false negative* respectively. All classification scores are calculated by averaging the results for each class.

	0	1	2	3	4
0	52	2	0	1	0
1	0	51	0	0	0
2	0	0	48	0	0
3	0	0	2	60	0
4	0	0	0	0	54

Figure 8. Confusion Matrix - KNN Classifier - Top Features data set.

Prior to the models being put into practice, the data set must be split into two parts: the **training / testing sets**. In this split, a ratio must be chosen between the data to which the model will be fitted to (trained on), and the data left aside to be used in the class prediction (tested on). In this work, the 70% training / 30% testing ratio was adopted.

While splitting, the properties of the original data set should be kept as much as possible, otherwise the results could be misleading (by chance of a really favorable or really poor split/sampling). To avoid that, a commonly used validation method is called *k*-**fold cross-validation**: the original data set is partitioned into k equal sized disjoint subsets, and then k runs of fitting/predicting are performed. For any iteration of this process, $k - 1$ subsets are randomly chosen as the training set, and testing of this fold is done with the remaining subset. The average results of the k runs' metrics are taken as results of the cross-validation (46; 49). A usual configuration is the 10-fold cross validation, which has been adopted in this work.

The adhesive joint strength classification results are shown in Table 2. The ensemble algorithms are utilizing the available features perfectly, the *k*-fold cross-validation shows 98-100% accuracy on all the 10 running instances, so they are as successful as one can be with this data set. Evaluating the **Gaussian Naïve Bayes** and **k-Nearest Neighbors** models, the first one shows a major excellent metrics also, although not as perfect as the **Random Forest Classifier**. As for the KNN, it shows major improvement on the top features subset, that contains 244 features. It is possible that some sort of feature number threshold was preventing the model from scoring higher.

Conclusions

A Structural Health Monitoring method, based on a Machine Learning classification pipeline, was successfully assembled, taking as inputs raw vibration signals from sensors, in a time series format. These signals originated from Finite Element Model simulations of a Lamb Wave propagating through a solid medium - a single-lap adhesive joint (with variable adhesion strength). After preprocessing, the relation between the time series was visualized through the employment of dimensionality reduction techniques.

Those time series signals were then subject to an automated feature extraction and selection procedure, where a large pool of features - attributes that characterize the signals in various domains (time, frequency, time-frequency, statistical) - were derived from the time series, and filtered through statistical methods, according to their relevance to the classification task.

Finally, those features were used to train Supervised Machine Learning algorithms, making them capable of predicting the approximate adhesion strength of the single-lap adhesive joint with great accuracy.

The principal conclusions drawn from this project are:

- Feature selection is a crucial part of any Machine Learning project that involves feature engineering, as the best results will emerge from the algorithms when the best possible set of features is available;
- More training data generally means better performance of the algorithms, so the working data set should be as big as possible;
- Ensemble methods are the best off-the-shelf ML algorithms in terms of classification performance, specially if applied along with feature engineering.

References

[1] M. Hong, Z. Su, Y. Lu, H. Sohn, X. Qing, Locating fatigue damage using temporal signal features of nonlinear lamb waves, Mechanical Systems and Signal Processing 60 (2015) 182–197.

[2] Z. Chen, R. Adams, L. F. Da Silva, Prediction of crack initiation and propagation of adhesive lap joints using an energy failure criterion, Engineering Fracture Mechanics 78 (6) (2011) 990–1007.

[3] D. N. Alleyne, The nondestructive testing of plates using ultrasound lamb waves. (1991).

[4] M. Mańka, M. Rosiek, A. Martowicz, T. Stepinski, T. Uhl, Pzt based tunable interdigital transducer for lamb waves based ndt and shm, Mechanical Systems and Signal Processing 78 (2016) 71–83.

[5] H. Lamb, On waves in an elastic plate, Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character 93 (648) (1917) 114–128.

[6] L. Rayleigh, On Waves Propagated along the Plane Surface of an Elastic Solid, Proceedings of the London Mathematical Society s1-17 (1) (1885) 4–11.

[7] S. Kessler, S. Spearing, Structural health monitoring in composite materials using lamb wave methods (2001).

[8] V. Giurgiutiu, A. Cuc, Embedded non-destructive evaluation for structural health monitoring, damage detection, and failure prevention, The Shock and Vibration Digest 37 (2005).

[9] J. Dodson, D. Inman, Thermal sensitivity of lamb waves for structural health monitoring applications, Ultrasonics 53 (3) (2013) 677–685.

Table 2. Classification algorithms' metrics - Adhesion Strength prediction on Single Lap Joint - 70/30 data split.

		Adhesive Joint Data Set		
Models		Full Features	Selected Features	Top Features
Naïve Bayes	**acc**	0.948	0.948	0.981
	p	0.952	0.952	0.981
	r	0.945	0.945	0.982
	F1	0.948	0.948	0.982
k-Nearest Neighbors	**acc**	0.552	0.552	0.982
	p	0.566	0.566	0.981
	r	0.550	0.550	0.980
	F1	0.555	0.555	0.982
Gradient Boosting	**acc**	0.996	1.00	0.993
	p	0.997	1.00	0.993
	r	0.996	1.00	0.993
	F1	0.996	1.00	0.993
Random Forest	**acc**	1.00	1.00	1.00
	p	1.00	1.00	1.00
	r	1.00	1.00	1.00
	F1	1.00	1.00	1.00

[10] Z. Su, L. Ye, Y. Lu, Guided lamb waves for identification of damage in composite structures: A review, Journal of Sound and Vibration 295 (3) (2006) 753–780.

[11] V. Giurgiutiu, Tuned lamb wave excitation and detection with piezoelectric wafer active sensors for structural health monitoring, Journal of Intelligent Material Systems and Structures 16 (2005) 291 – 305.

[12] R. D. Adams, R. D. Adams, J. Comyn, W. C. Wake, W. Wake, Structural adhesive joints in engineering, Springer Science & Business Media, 1997.

[13] L. F. Da Silva, A. Öchsner, R. D. Adams, Handbook of adhesion technology, Springer Science & Business Media, 2011.

[14] D. Packham, B. Cope, A. Kinloch, D. Aubrey, M. Pascoe, B. Kneafsey, D. Dixon, D. Brewis, G. Critchlow, D. Dillard, S. Millington, G. Curtis, Handbook of Adhesion, John Wiley & Sons, Ltd, 2005.

[15] C. B. (Ed.), C. Bajaj, C. F. J. Wiley, Data visualization techniques (1998).

[16] P. E. Rauber, A. X. Falcao, A. C. Telea, et al., Visualizing time-dependent data using dynamic t-sne. (2016).

[17] J. T. Machado, A. M. Lopes, Fractional state space analysis of temperature time series, Fractional Calculus and Applied Analysis 18 (6) (2015) 1518–1536.

[18] W. Aigner, S. Miksch, W. Müller, H. Schumann, C. Tominski, Visualizing time-oriented data – a systematic view, Computers Graphics 31 (2007) 401–409. doi:10.1016/j.cag.2007.01.030.

[19] L. Al Shalabi, Z. Shaaban, Normalization as a preprocessing engine for data mining and the approach of preference matrix, in: 2006 International Conference on Dependability of Computer Systems, 2006, pp. 207–214. doi:10.1109/DEPCOS-RELCOMEX.2006.38.

[20] A. Zheng, A. Casari, Feature engineering for machine learning: principles and techniques for data scientists, " O'Reilly Media, Inc.", 2018.

[21] F. W. Young, D. F. Harris, Multidimensional scaling, LL Thurstone Psychometric Laboratory, University of North Carolina, 1983.

[22] I. Borg, P. J. Groenen, Modern multidimensional scaling: Theory and applications, Springer Science & Business Media, 2005.

[23] G. R. Hancock, R. O. Mueller, L. M. Stapleton, The reviewer's guide to quantitative methods in the social sciences, Routledge, 2010.

[24] L. Wilkinson, et al., Multidimensional scaling, Systat 10.2 Statistics II (2002) 119–145.

[25] S. A. Mohammadi, B. Prasanna, Analysis of genetic diversity in crop plants—salient statistical tools and considerations, Crop science 43 (4) (2003) 1235–1248.

[26] J. T. Machado, A. M. Lopes, On the mathematical modeling of soccer dynamics, Communications in Nonlinear Science and Numerical Simulation 53 (2017) 142–153.

[27] G. Hinton, S. T. Roweis, Stochastic neighbor embedding, in: NIPS, Vol. 15, Citeseer, 2002, pp. 833–840.

[28] L. Van der Maaten, G. Hinton, Visualizing data using t-sne., Journal of machine learning research 9 (11) (2008).

[29] P. Ongsulee, Artificial intelligence, machine learning and deep learning, in: 2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE), IEEE, 2017, pp. 1–6.

[30] A. Desarda, https://towardsdatascience.com/getting-data-ready-for-modelling-feature-engineering-feature-selection-dimension-reduction-77f2b9fadc0b, accessed: 31-07-2021 (2018).

[31] L. Van Der Maaten, E. Postma, J. Van den Herik, et al., Dimensionality reduction: a comparative review, J Mach Learn Res 10 (66-71) (2009) 13.

[32] Y. Benjamini, M. Leshno, Statistical methods for data mining, in: Data mining and knowledge discovery handbook, Springer, 2005, pp. 565–587.

[33] G. Dong, H. Liu, Feature engineering for machine learning and data analytics, CRC Press, 2018.

[34] S. Ozdemir, D. Susarla, Feature Engineering Made Easy: Identify unique features from your dataset in order to build powerful machine learning systems, Packt Publishing Ltd,

2018.

[35] E. (https://stats.stackexchange.com/users/16137/emile), https://stats.stackexchange.com/questions/50807/features-for-time-series-classification, accessed: 05-08-2021 (2014).

[36] Y. Xiong, D.-Y. Yeung, Time series clustering with arma mixtures, Pattern Recognition 37 (8) (2004) 1675–1689.

[37] A. Taspinar, https://ataspinar.com/2018/04/04/machine-learning-with-signal-processing-techniques/, accessed: 08-08-2021 (2018).

[38] D. J. Struik, https://www.britannica.com/biography/Joseph-Baron-Fourier, accessed: 09-08-2021 (2021).

[39] P. Heckbert, Fourier transforms and the fast fourier transform (fft) algorithm, Computer Graphics - Carnegie Mellon University 2 (1995) 15–463.

[40] S. R. Kulkarni, Frequency domain and fourier transforms, Lecture Notes - Introduction to Electrical Signals and Systems (2002).

[41] N. Kehtarnavaz, Chapter 7 - frequency domain processing, in: N. Kehtarnavaz (Ed.), Digital Signal Processing System Design (Second Edition), second edition Edition, Academic Press, Burlington, 2008, pp. 175–196. doi:https://doi.org/10.1016/B978-0-12-374490-6.00007-6.
URL https://www.sciencedirect.com/science/article/pii/B9780123744906000076

[42] B. Rajoub, Chapter 2 - characterization of biomedical signals: Feature engineering and extraction, in: W. Zgallai (Ed.), Biomedical Signal Processing and Artificial Intelligence in Healthcare, Developments in Biomedical Engineering and Bioelectronics, Academic Press, 2020, pp. 29–50. doi:https://doi.org/10.1016/B978-0-12-818946-7.00002-0.
URL https://www.sciencedirect.com/science/article/pii/B9780128189467000020

[43] M. Kiymik, I. Guler, A. Dizibuyuk, M. Akin, Comparison of stft and wavelet transform methods in determining epileptic seizure activity in eeg signals for real-time application, Computers in biology and medicine 35 (2005) 603–16. doi:10.1016/j.compbiomed.2004.05.001.

[44] N. Sairamya, L. Susmitha, S. Thomas George, M. Subathra, Chapter 12 - hybrid approach for classification of electroencephalographic signals using time–frequency images with wavelets and texture features, in: D. J. Hemanth, D. Gupta, V. Emilia Balas (Eds.), Intelligent Data Analysis for Biomedical Applications, Intelligent Data-Centric Systems, Academic Press, 2019, pp. 253–273. doi:https://doi.org/10.1016/B978-0-12-815553-0.00013-6.
URL https://www.sciencedirect.com/science/article/pii/B9780128155530000136

[45] M. Christ, N. Braun, J. Neuffer, A. W. Kempa-Liehr, Time series feature extraction on basis of scalable hypothesis tests (tsfresh – a python package), Neurocomputing 307 (2018) 72–77. doi:https://doi.org/10.1016/j.neucom.2018.03.067.
URL https://www.sciencedirect.com/science/article/pii/S0925231218304843

[46] Z.-H. Zhou, Ensemble Methods: Foundations and Algorithms, Chapman and Hall/CRC, 2019.

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

[48] M. Hossin, M. N. Sulaiman, A review on evaluation metrics for data classification evaluations, International journal of data mining & knowledge management process 5 (2) (2015) 1.

[49] G. Chandrashekar, F. Sahin, A survey on feature selection methods, Computers & Electrical Engineering 40 (1) (2014) 16–28.