FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Non-Intrusive Appliance Load Monitoring using a Lightweight Disaggregation Algorithm

João Carlos Amaro Freixo

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Paulo José Lopes Machado Portugal

July 31, 2023

Resumo

Nos últimos anos, o consumo de energia elétrica está num nível mais alto do que nunca, com a procura global por energia continuando a aumentar à medida que as populações e as economias crescem. O crescente uso de tecnologia e equipamentos elétricos em casas e escritórios é um fator que contribui para os altos níveis de consumo de energia. Como resultado, há uma necessidade urgente de encontrar maneiras de reduzir o consumo de energia para reduzir a dependência de combustíveis fósseis, que são uma importante fonte de emissões de gases de efeito estufa e contribuem para o aquecimento global.

Um passo crucial na redução do consumo de energia de qualquer sistema elétrico, seja um edifício ou um sistema embarcado, é a monitorização e identificação de cargas elétricas. A monitorização de cargas é o processo de identificar a potência consumida por uma carga elétrica e pode ser realizada segundo duas perspetivas, intrusiva ou não intrusiva. O método intrusivo é inconveniente, pois exige que aparelhos de medição de potência sejam instalados em todos os equipamentos elétricos, e adicionar uma nova carga exigiria recalibração de todo o sistema. O método não intrusivo tem a simplicidade de permitir a monitorização de todos os equipamentos a partir de um único ponto, com as desvantagens de precisar de desagregar e identificar as diferentes cargas. Num sistema de monitorização de carga não intrusivo, a identificação de diferentes cargas pode ser feita com Aprendizagem Máquina, o que impõe o problema de necessitar de um poder computacional elevado nas plataformas onde os algoritmos são executados.

Os trabalhos mais recentes sobre este assunto concentram-se em aumentar a complexidade desses algoritmos de aprendizagem para aumentar a precisão de identificação das cargas, alojando-os em dispositivos caros ou recorrendo a técnicas como aplicações em *Cloud*. Esta dissertação concentra-se no desenvolvimento de um algoritmo de aprendizagem máquina computacionalmente leve e capaz de identificar com precisão o consumo de energia de cargas individuais por meio de seu comportamento elétrico.

Isto será alcançado recorrendo a uma técnica de aprendizagem superficial, o algoritmo Random Forest. Em estudos recentes, este algoritmo demonstrou ser capaz de atingir uma elevada precisão na desagregação de energia e identificação de equipamentos. O algoritmo funciona treinando múltiplas árvores de decisão em diferentes subconjuntos de dados, o que leva a um algoritmo preditivo menos tendencioso. A decisão final é tomada pela média da decisão de todas as árvores.

Ao testar o desempenho de desagregação desta regressão tanto em dados de carga simulados, obtidos num simulador de circuitos elétricos, como em dados reais obtidos através de um *dataset*, verificámos que a regressão consegue desagregar com precisão múltiplas cargas simples. O aumento da complexidade da carga e do número de cargas degrada significativamente a qualidade da desagregação, no entanto, *features* elétricas adicionais atenuam este problema, reduzindo o erro de previsão em todos os tipos de cargas.

Abstract

In recent years, energy consumption is at an all-time high, with the global demand for energy continuing to rise as populations and economies grow. The increasing use of technology and electrical equipment in homes and work offices are contributing factors to the high levels of energy consumption. As a result, there is a pressing need to find ways to reduce energy demands in order to reduce reliance on fossil fuels, which are a major source of greenhouse gas emissions and contribute to climate change.

A crucial step in lowering the energy consumption of any electrical system, whether it be a building or an embedded system, is the monitoring and identification of electrical loads. Load monitoring is the process of tracking the power consumed by an electrical load and can be done in two major ways, intrusive or non-intrusive. The intrusive method is inconvenient, as it requires power meters to be installed in all electrical equipment, and adding a new load would require re-calibration of the whole system. The non-intrusive method has the simplicity of allowing the monitoring of all equipment from a single point, with the drawbacks of needing to disaggregate and identify the different loads. In a non-intrusive load monitoring system, the identification of different loads can be done with Machine Learning, which imposes the problem of requiring a large computational power in the platforms where the algorithms are executed.

Most Recent articles on this matter focus on increasing the complexity of said learning algorithms in order to increase the identification accuracy of the loads, housing them in either expensive devices, or relying on techniques such as Cloud Applications. This dissertation focuses on developing a lightweight machine learning algorithm capable of accurately identifying the power consumption of individual loads through their electrical behavior.

This will be achieved by relying on a shallow learning technique, the Random Forest algorithm. In recent studies, this algorithm has shown that it is able to achieve high accuracy in energy disaggregation and appliance identification. The algorithm works by training multiple decision trees in different subsets of data, which leads to a less biased predictive algorithm. The final decision is made by averaging the decision of all trees.

Upon testing the disaggregation performance of this regression on both simulated load data, obtained on an electrical circuit simulator, and real data obtained via a dataset, we found that the regression can accurately disaggregate multiple simple loads. The increase in load complexity and number of loads significantly degrades the disaggregation quality, however, additional electrical features can mitigate this problem, reducing the prediction error on all types of loads.

Acknowledgements

I would like to express my heartfelt thanks to my parents and brother for their unwavering support and guidance throughout my life and academic journey. Your belief in my abilities and pushing me to achieve my goals have played a significant role in shaping me into the person I am today. From providing valuable insights and offering emotional support during challenging times, your presence has been an invaluable source of strength.

I am also immensely grateful to my uncles, cousins, and extended family for their unwavering support and encouragement and all the get-togethers and joy. My successes have been motivated by your conviction in my potential, and I am appreciative of our family's sense of harmony and support.

I can't express enough gratitude to my girlfriend Lara Anjos and my incredible circle of friends for their unwavering emotional support and motivation. Your presence has been a lifeline, providing comfort in trying times and a necessary diversion when required. Your friendship has truly been a gift.

Finally, I would like to express my deepest appreciation to my supervisor, Prof. Paulo José Lopes Machado Portugal. Your knowledge, advice, and tolerance have been extremely helpful to me at every step of the way. Your expertise, guidance, and patience have been invaluable throughout my entire journey. Your unwavering support, timely feedback, and willingness to go the extra mile have shaped my work and contributed significantly to its success. I am grateful for the mentorship and knowledge you have shared with me, which will undoubtedly leave a lasting impact on my future endeavors.

Thank you so much to each and every one of you. I am very lucky to have such amazing people in my life, and your continued support has been crucial to my success. I'm looking forward to reaching this milestone with you all and starting new journeys with your support and affection. With sincere appreciation,

João Freixo

"When life gives you lemons, don't make lemonade.
Make life take the lemons back!"

Cave Johnson

Contents

List of Figures

List of Tables

Abbreviations

AMPds	Almanac of Minutely Power Dataset
BLUED	Building-level Fully Labeled Dataset for Electricity Disaggregation
CNN	Convolutional Neural Networks
CSV	Comma Separated Values
CVD	Continuously Variable Device
CWT	Continuous Wavelet Transform
DNN	Deep Neural Networks
DRED	Dutch Residential Energy Dataset
DWT	Discrete Wavelet Transform
ECO	Electricity consumption and occupancy
FFNN	Feed-Forward Neural Networks
FFT	Fast Fourier Transform
FL	Federated Learning
FSM	Finite State Machine
iAWE	Indian Dataset for Ambient Water and Energy
IoT	Internet of Things
Imax	Max Current
Irms	Effective Current
KNN	K Nearest Neighbors
ML	Machine Learning
MSQ	Mean Square Error
NILM	Non-Intrusive Load Monitoring
PF	Power Factor
PLAID	Plug-Level Appliance Identification Dataset
P	Real Power
PQA	Power Quality Analysers
Q	Reactive Power
Q1	First Quartile
Q3	Third Quartile
RECAP	Real Time Recognition and Profiling of Appliances
REDD	Reference Energy Disaggregation Dataset
RFE	Recursive Feature Elimination
RNN	Recurrent Neural Networks
STFT	Short Time Fourier Transform
SynD	Synthetic Dataset
VI	Voltage and Current
Vmax	Max Voltage

Chapter 1

Introduction

This chapter aims to provide the context and motivation behind the proposal addressed in this thesis, as well as its objectives. In the same way, the last part of this chapter explains how the rest of the document is organized.

1.1 Context

With the recent trend of more people living in cities and using more electrical devices, energy demands are growing all the time. According to the work of Angelis et al. (2022) [4] and Schirmer and Mporas (2022) [5], energy production accounts for 75 percent of all greenhouse gasses emitted into the atmosphere, with households responsible for 25 percent of total energy consumption in Europe, rising to 35 percent in the United States of America.

In order to tackle this problem, one possible solution is to be mindful of which appliances have the highest power consumption in order to reduce said consumption through a process called *appliance load monitoring*. Appliance load monitoring is the process of measuring and analyzing the power consumption of each appliance in a building or home. This can be done for a variety of reasons, such as to reduce power usage, diagnose problems in the network, or monitor the performance of appliances over time.

There are two major methods to monitor appliance loads: intrusive and non-intrusive. Intrusive methods require the installation of energy meters on all appliances, allowing you to control and monitor the power usage of these appliances. This solution has some problems, such as the fact that the cost of installation goes up with the number of appliances plugged in, that the meters may not be able to be put on some appliances, such as those that are in hard-to-reach places, and that it is bothersome for the user to have to re-configure the system every time they want to add, move, or remove an appliance from the system ([5]).

The Non-Intrusive Load Monitoring (NILM) technique was created to address all of these problems. It lets every piece of electrical equipment in a network be monitored for how much power by measuring the aggregate consumption from a single point in an electrical network and then disaggregating said measured power to show how much power each appliance consumes. The

disaggregation of the overall power is a complex operation, and different methods have varying degrees of success, however, all methods use the appliance's features (voltage, current, etc.) in order to find which ones are consuming power at any moment.

According to Dash and Sahoo (2022) [1], the NILM implementation has the following benefits:

- NILM can help households and buildings reduce their power consumption by identifying specific appliances that are using excessive amounts.
- The disaggregation can infer daily habits, allowing us to monitor people who live in inaccessible areas.
- This implementation is independent of the number of appliances connected to the household's network, making its installation simpler.

1.2 Motivation

NILM implementations are able to identify different appliances and their respective power consumption through a process of disaggregation. The disaggregation process consists of breaking down the measured power consumption into the most probable combination of individual consumption for the loads in the network. Research on NILM mainly focuses on the development of Machine Learning (ML) algorithms that allow the disaggregation of said individual loads[1]. These algorithms are computationally heavy, so they require either a powerful device that can handle them or reliance on architectures such as the Internet of Things (IoT) and other such *Cloud* applications.

The motivation for this proposal is to find and evaluate the performance of an ML algorithm, so that we may implement the NILM architecture without relying on expensive hardware, making its cost reasonable for the average household.

1.3 Objectives

Considering the points presented in the previous section, this dissertation will focus on the development of a machine learning algorithm that could be deployed on a cost-effective device, such as a microprocessor. We will find the best possible machine learning algorithm and features in order to ensure an accurate disaggregation with the lowest possible memory and processing requirements so that it is as lightweight as possible. This accuracy will be verified via a scenario of tests to ensure that this accuracy isn't dependent on the specific data used to train said algorithm.

1.4 Organization of the Dissertation

Apart from this introduction, this paper will contain 6 more chapters, in the following structure:

- Chapter 2, Background: This chapter will present the core knowledge in non-intrusive load monitoring and machine learning needed in order to understand the concepts in further chapters.
- Chapter 3, State of the Art: In this chapter, we will go through the work that has already been done on the subjects relevant to this proposal. We will also discuss in further detail the researched solutions that can be applied to our problem.
- Chapter 4, Proposed Solution: This chapter contains the methodologies that will be used to tackle the problem in this thesis, along with how the said solution will be implemented, the hardware requirements, and how this solution will be evaluated.
- Chapter 5, Development: This chapter will present the specific technologies used to implement the techniques presented in chapter 4, as well as the steps used in the said implementation.
- Chapter 6, Experimental Results: In this chapter, we will perform a wide range of tests in both the selected dataset data and simulated data that will allow for an adequate evaluation of the performance of the developed algorithm.
- Chapter 7, Conclusion and Further Work: In this chapter, we will perform an overview of the work done, as well as ponder on the further work that needs to be done on the topic of this thesis.

Chapter 2

Background

In this chapter, we will go over the principles needed to understand the work that will be done for this proposal. We'll start by talking about what non-intrusive load monitoring is, and what stages are involved in this process. In another section, we will address machine learning in NILM, along with the various learning types that have been used in its architecture. Finally, we will go over how this knowledge is relevant in the following chapters.

2.1 Non intrusive Load Monitoring

Non-Intrusive Load Monitoring is a technique that involves estimating the power consumption of individual appliances in an electrical network while measuring the total power consumption of the network with a single meter. The goal of NILM is to figure out how much power each appliance uses by looking at how much power the whole network uses. This is typically done using algorithms that analyze the measured aggregate power consumption in an electrical network produced by the appliances, which can be used to identify the individual power consumption of the different appliances based on their unique electrical signatures. The results of NILM can be used to improve energy efficiency by identifying appliances that are using more energy than necessary and developing strategies for reducing energy consumption.

2.1.1 Mathematical expression

According to Dash and Sahoo (2022) [1], the NILM disaggregation problem can be expressed mathematically by the following expression:

$$P_{total}(t) = \sum_{n=1}^{N} p_n(t) + e(t)$$

In this equation, $P_{total}(t)$ represents the known measured aggregated power in any instant t, $p_n(t)$ is the predicted power that any appliance connected to the household's network n is consuming at the instant of time t and the error $e(t)$ represents the gap between the measured power and the sum of the predicted power consumed by all operating appliances. In their work, Dash and Sahoo

(2022) [1] explain that this error may be present " due to the combined effects of distribution line losses and measuring instrument operation under real-time conditions".The disaggregation problem involves finding the most probable combination of individual power consumption that minimizes the error value so that line losses and measuring errors are the only sources of error between the measured total power and the sum of the individual predicted power consumption.

2.1.2 Data acquisition

Data acquisition is one of the vital parts of a NILM, as it is the mechanism that allows the system to capture the individual patterns of the loads [6]. Usually, a NILM system has a "single/three phase voltage and current sensor module" [6] connected to a central point of the electrical network it is installed on so that it can measure the aggregated power consumption. In order to choose the right power meter for a system, it is necessary to know its specific application. At the time of the writing of this dissertation, based on the work of Gopinath et al. (2020) [6], there are two types of applicable architectures of power meters, digital meters and Power Quality Analyzers (PQA). The cheaper digital meters work on low sampling rates that are in the Hz range and, as such, lack the capability for analysis of frequency components and identification of higher order harmonics [6]. In contrast, the costly PQAs are able to process and condition the electrical signal more quickly, due to working on sampling rates in a range from 10kHZ to 100MHz [6]. PQAs also allow the time domain signal to be transformed into a frequency domain one, which is useful to identify power quality-related features [6]. This topic will be expanded on in chapter 3.

2.1.3 Appliance types and identification

Aside from figuring out how much power each appliance uses, the NILM architecture is also needed to figure out what appliance is using that power. People's homes may have a wide variety of appliances, such as those that draw a lot of power, like refrigerators and washing machines, and appliances that draw low power, such as toasters and coffee makers. A building may also have electronic devices, e.g. computers and smartphones, a lighting system, and heating, ventilation, and air conditioning systems. We also must consider that different appliances may have similar power behavior, making them hard to distinguish from one another.

Considering that these appliances will be operating simultaneously, a NILM device needs to be able to distinguish between them. Recent NILM studies ([4],[6]) classify appliances according to the following categories, with said categories demonstrated in figure 2.1:

- Type I: appliances that have only two states (ON/OFF). The defining characteristic of such appliances is that they only consume power during their ON state [6]. Kettles, light bulbs, and other resistive appliances are examples of type I appliances [4].

- Type II: finite state machines (FSM) with repeatable operational states. These appliances are characterized by having multiple states during their operation [4], with state transitions

being identified via rising or falling power usage edges [6]. Examples of such appliances are refrigerators and washing machines [6].

- Type III: Continuously Variable Devices (CVD). These appliances don't have a set number of states, with power consumption varying continuously during their operation. This type of behavior makes it hard for such appliances to be disaggregated from the measured total power [4]. Type III appliances include dimmable lights and drills [6].
- Type IV appliances run continuously for a long period of time. Examples of such appliances include cable TV receivers and smoke detectors [6].

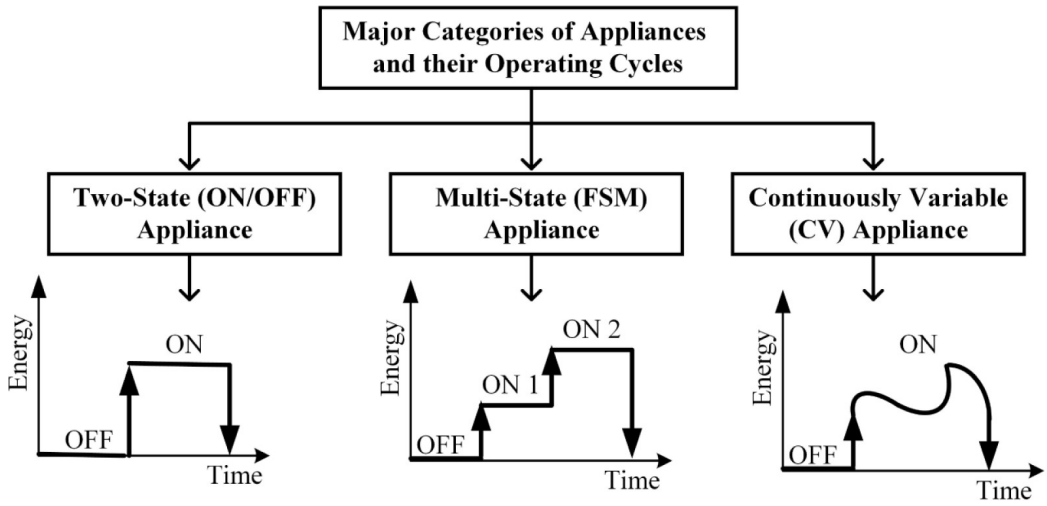


Figure 2.1: Appliance types categorized by operation style [1]

Each type of appliance requires different amounts and types of information/features to be properly disaggregated and identified from the measured power, as such, it is necessary to measure different types of features to implement a complete NILM system.

2.1.4 Electrical Features

In the NILM architecture, appliances are identified by their electrical behavior in a building's electrical network, this behavior is categorized into electrical features, and the combination of all these sampled electrical characteristics composes an individual electrical signature for all appliances. By looking at the behavior of these features we can then identify *Electrical Features* can be divided into three categories: Steady State, Transient, and External.

2.1.4.1 Steady State Features

Steady state features can be extracted when the appliances are operating at a steady state. These features may be extracted from data sampled at a low frequency of a few Hz in range, as there aren't many changes in power over time during steady state operation [6]. However, steady state features may not be enough to properly identify all appliances, as multiple machines may have the

same or similar behavior in the sampled features. It is also known that steady state features may not be able to properly identify and disaggregate type III appliances, considering their particular behavior over time. The work of Dash and Sahoo (2022) [1] also noted that steady state features may lead to faulty identification when the network experiences power disturbances. Examples of steady state features include voltage, current, real and reactive power, and power factor[6].

2.1.4.2 Transient features

Transient features are electrical features that can be extracted in the transient state of operation. Transient features allow for a more accurate appliance identification [6], as it is unlikely that different appliances possess similar transient features [1]. Transient features also have the advantage of allowing the estimation of an appliance's health. Considering the erratic behavior of a transient state system, in which its current state can rapidly change, transient features require sampled data in high frequencies (kHz range) in order to be extracted [6]. This high sampling rate increases the requirements of the NILM device, in order to be able to support computationally heavy calculations [1]. Examples of transient features include transient real and apparent power, fast-Fourier transform (FFT) and Short-time Fourier transform (STFT) features, and VI trajectories.

2.1.4.3 External features

Apart from the information extracted from the sampled data from the electrical network, additional external features can be extracted from the appliance usage over time [4], such as period of consecutive time it is used for, peak time usage and how often the appliance is used [6]. In order to use such features, it is necessary to encode them in order for them to be recognizable by a disaggregation algorithm [4]. As an example, Angelis et al. (2022) [4] provides an encoding system where temporal features are profiled as a 2D circle, with their behavior being expressed in the respective sine and cosine values.

2.2 Machine learning in NILM

Considering the disaggregation stage, there are several approaches to this step in recent NILM implementations ([4],[6]), but most rely on machine learning algorithms to learn the characteristics of the energy consumption of different appliances and identify these characteristics in the aggregate power. These algorithms can be trained on household power reading data and appliance labels. Once trained, the algorithms can use energy consumption patterns to identify which appliances are in use at any given time.

In the training phase, the device may need to capture the signatures of the appliances on the power line [6]. This training process can be performed using either a supervised or an unsupervised learning approach. In a supervised learning approach, consumers can manually label the appliances, allowing the NILM system to recognize and differentiate between them. However, an unsupervised approach does not require manual labeling and instead relies on the NILM system

to learn and identify the appliances independently. This process may take a few weeks, during which the consumer may need to confirm the accuracy of the appliance detection made by the unsupervised NILM device [6]. An unsupervised approach has the advantage of being able to adapt to frequent changes in the environment, such as the introduction of a new appliance to the household. In this case, the NILM system would need to identify the new appliance and capture its signatures over time in order to accurately estimate its energy consumption.

The work of Gopinath et al. (2020) [6], further divides these algorithms into two different categories, event-based and eventless.

2.2.1 Event based algorithms

An event-based NILM algorithm aims to identify the individual appliances based on changes to the aggregated measured power [1]. These algorithms work by looking for specific events, which are changes in the aggregated power due to an appliance switching state [6] and then extracting the features associated with those events in order to form an electric signature [1]. Once the event has been identified and the features extracted, the acquired signature is then assigned to an individual appliance, and the energy consumption of that appliance is estimated using the knowledge of the appliance's operation sequences. In an event-based ML identification algorithm, a classification algorithm is used to identify the appliance using the signatures obtained from the event. These algorithms either use the raw time series data for classification or try to extract features from the data and classify them using standard ML algorithms.

2.2.2 Eventless algorithms

Eventless NILM algorithms are methods used to estimate the energy consumption of individual appliances without the need for detecting specific events or changes in the aggregate signal [1]. These techniques take all the samples of the aggregate signal and use them to estimate the energy consumption of individual appliances. Eventless NILM techniques primarily use low-frequency aggregate signals [1]. Eventless NILM algorithms can be used when event-based NILM techniques might not work, such as when appliances have a very low power consumption or when a few high-power appliances make up most of the signal as a whole. They can also be used when the appliance usage patterns are not known in advance or are difficult to model. The work of Gopinath et al. (2020) [6] proposes that both event-based and non-event-based approaches have comparable energy disaggregation performance.

2.3 Conclusion

In this section, we introduced all the needed concepts in order to understand the topics presented in the State-of-the-Art (chapter 3). We also presented some examples that help better contextualize the aforementioned concepts so that they be more easily internalized.

Chapter 3

State-of-the-Art

In recent years, there has been a surge of research and development in NILM, leading to significant advances in the accuracy and robustness of the algorithms. In this chapter, we provide a state-of-the-art review of NILM, covering the key techniques and challenges, and highlighting the most relevant and impactful contributions in the literature. This chapter will be heavily based on the works of Angelis et al. (2022) [4] and Sadeghianpourhamami et al. (2017) [7], as they are recent studies that compile multiple recent studies and their findings. This chapter will be divided into three sections, each going over an essential topic for finding the solution to this proposal. The first section will contain information on the most recently used features, how to appropriately select them, and how to process the raw measured data to avoid problems with the disaggregation algorithm. The following section goes over the disaggregation algorithms used in recent works, as well as their respective advantages and disadvantages. The final section presents multiple datasets to be used in the training stage of the ML algorithms, along with how to enhance the data present in said datasets.

3.1 Electrical features

As discussed before in chapter 2, electrical features are the intrinsic characteristics of appliances on a network, which allows the NILM device to properly identify said appliances.

3.1.1 Feature extraction

3.1.1.1 P-Q Plane

The P-Q plane, which represents real power (P) and reactive power (Q), is a common feature used in non-intrusive load monitoring to identify the operation of appliances [7]. These features can effectively detect Type I appliances as well as identify appliances with high power consumption. However, low-power appliances are difficult to distinguish using only P-Q features because they tend to overlap on the plane. To overcome this limitation, according to, NILM researchers have combined the P-Q plane with other features in order to expand the identifiable appliances.

A combination of low-frequency P-Q, voltage, and current features, such as power factor (PF), phase shift, max voltage (Vmax), max current (Imax), and effective current (Irms) has been found to be a good approach to identify the operational state of kitchen appliances using the Real Time Recognition and Profiling of Appliances (RECAP) method [7]. The RECAP method is a NILM approach that uses these features with a supervised ML algorithm, where the user first identifies the appliances in their household so that the device can then recognize them during real-time operation [8]. However, detecting the specific states of Type II appliances requires the use of features beyond steady-state features [7].

Combining the P-Q plane and macroscopic transient features allows the NILM system to identify appliances with a prolonged transient state and large spikes in power draw followed by gradual changes in said power consumption [7]. Examples include heat pumps and electric loads in industrial settings. These transient features are defined by their edges and slopes, as well as their power profiles. While these features are inexpensive to extract due to low-frequency requirements, they are not effective at identifying type II and type III appliances [7].

Another possible combination is the P-Q plane and harmonics, which are often used together in order to improve appliance identification. Harmonics are the frequency components of the current or power waveforms, which can be obtained by applying an FFT to high-frequency measurements [7].

By bringing together the two previous approaches, we can utilize the P-Q plane in combination with macroscopic transient and harmonic features. This set of features is particularly unique, as it combines the low-frequency and high-frequency features [7].

Finally, some studies employ only real power measurements as a feature, as reactive power is the more expensive of the pair in terms of sampling. Although this approach is cheaper, it still presents the problem of not being able to differentiate between appliances with similar behavior so it is often combined with external features, such as time and frequency of usage [7].

3.1.1.2 Spectral envelope and Wavelets

Spectral envelopes are a type of feature that are derived from the STFT of the electrical signal [7]. Unlike FFT, which maps the entire signal to the frequency domain and loses timing information, STFT uses a fixed window to transform a small section of the signal at a time, preserving the timing information by "mapping the signal to a two-dimensional function of time and frequency" [7]. As a result, spectral envelopes provide an accurate representation of the signal and are well-suited for identifying Type III appliances. However, the use of a fixed window size for all frequencies can restrict the flexibility of STFT and limit its ability to accurately capture certain types of signals.

To mitigate this accuracy inconsistency, we can employ the *wavelet transforms*, which are a flexible tool for representing variable signals [7]. These transforms decompose the signal into time and scale using wavelets with adjustable size properties, allowing for the use of longer or shorter windows depending on the desired frequency range. Researchers have explored various types of wavelet transforms, such as continuous wavelet transforms (CWT) and Discrete Wavelet Transforms (DWT). Wavelet coefficients are effective at representing nonlinear waveforms, but

they can require a high-frequency sampling rate and complex signal processing. The work of Sadeghianpourhamami et al. (2017) [7] also presents a study in which wavelets are combined with P-Q plane features, leading to a more accurate and faster training of the used disaggregation algorithm.

3.1.1.3 Waveforms and Trajectories

One approach to non-intrusive load monitoring is to use the raw, unprocessed electrical current waveforms for load discrimination, eliminating the need for signal processing in the feature extraction phase [7]. While the current waveforms of different nonlinear appliances may differ, research has shown that feature-based approaches are more robust than using raw waveforms alone. This is likely due to the fact that feature-based approaches can capture more information about the electrical signal and are better able to distinguish between different appliances [7].

An alternative way to use the waveforms as a feature is to use shape-based features derived from the two-dimensional VI trajectories of the electrical signal. The works presented in Sadeghianpourhamami et al. (2017) [7] demonstrated further improvements in load categorization using additional features extracted from VI trajectories. These shape-based features are able to take into account the shapes of the waveforms in a more robust way, enabling more accurate discrimination between different appliances.

3.1.2 Feature Selection

A key part of developing an ML algorithm for NILM is feature selection, which is the process of choosing a subset of relevant features or variables to use in building a model. There are three main categories of feature selection methods [7]:

- Filter methods, which select features based on intrinsic data properties, often by computing a feature relevance score and removing features with low scores. These methods are computationally fast but may be less accurate because feature selection is independent of the model [7].
- Wrapper methods involve evaluating feature subsets by the accuracy of a specific model using them. Because it is impractical to create a model for every possible subset of features, a heuristic search procedure is used [7], in which an algorithm finds a good enough solution in a reasonable amount of time rather than an optimal solution.
- Embedded methods determine feature importance by training a model and extracting the knowledge automatically; they are coupled to the model being used and require less computational effort than wrapper methods [7].

Sadeghianpourhamami et al. (2017) [7] investigate two feature selection methods in order to identify the possible best solutions for this problem, both of which use the *Random Forest* algorithm, which will be discussed in greater detail in another section of this chapter, as the classification model. One of the feature selection methods employed was recursive feature elimination

(RFE), which builds a model using all features and then ranks them according to their importance. Then, the least important parts are removed from the model, and the process is repeated until a stopping point is met.

Ultimately, Sadeghianpourhamami et al. (2017) [7] found that the results of this method were not satisfactory due to the high correlation among the features. As a result, another approach was studied, which involved iteratively training the Random Forest algorithm using the retained features from the first method and computing the permutation importance of each feature. This is done by removing the relationship between each feature and the target in the validation phase and measuring the average increase in the relative error of the model after randomly permuting the values of the features multiple times. This importance measurement is often more effective in cases where the included features are highly correlated. Essentially, features that have little influence on the model's accuracy in discriminating loads will only cause a minimal change in the model's performance.

3.1.3 Data Preprocessing

Data measured by a NILM device may require additional processing in order to be usable by the desired ML algorithms [4]. To prepare the data for training and evaluation, it may be necessary to undergo a process called data preprocessing. This process transforms raw data into a more suitable format. The work of Angelis et al. (2022) [4] presents multiple preprocessing methods that recent NILM studies employ.

In recent years, neural networks and other deep learning methods are effective for NILM tasks because they can learn complex features from raw data with minimal manual feature extraction [4]. As presented in the sub-subsection 3.1.1.2, one approach for converting signal data into a form that can be processed by a neural network is the sliding window method, which divides historical data into fixed-length sequences that overlap with each other. Long Short-Term Memory Networks can retain important data during training through the use of gates, but it is important to determine the optimal length of the input sequences. The optimal receptive field length is also crucial for time series regression tasks like NILM, but there is no universal method for determining the optimal input size. Angelis et al. (2022) [4] presents some studies that have used the differential of the raw signal as input, which can improve disaggregation performance by allowing the ML algorithm to focus on one appliance at a time.

It can be challenging for deep neural network (DNN) models to handle raw data, particularly when working with time-series data [4]. Scaling techniques can help improve the performance of the model and speed up the learning process. In NILM research, two common techniques are z-score normalization and min-max normalization. Z-score normalization involves calculating the mean and standard deviation of the main and sub-metered (as in, directly measured from the appliance) signals in the training set and using these values to transform the data. Min-max normalization scales the data to a range of [0, 1] or [-1, 1] by using the minimum and maximum values of the training set. Min-max normalization is often used with time series data, but less often with NILM because it is sensitive to outliers [4].

3.2 Machine Learning algorithms

Machine learning has been an effective tool for addressing NILM problems. There has been a lot of progress in developing machine learning algorithms and techniques for NILM tasks. These algorithms and techniques have shown that they can accurately separate the individual power of each appliance, even when the data has complex patterns and changes. In this chapter, we'll go over the current state of machine learning in NILM, according to the work of Angelis et al. (2022) [4].

3.2.1 Shallow Learning approaches

Shallow learning refers to traditional classification and regression algorithms [4]. These algorithms rely on manual feature extraction, which requires the developer to be an expert in the field. Shallow learning algorithms are generally easier to implement and require less computing power compared to deep learning algorithms. However, they may not always perform as well as deep learning algorithms on complex tasks.

3.2.1.1 Naive Bayes Classification

The *Naive Bayes classifier* is a machine learning algorithm that can be used for classification tasks where the primary goal is to determine the best method for linking a new piece of data to a set of predefined categories within a specific context or field of study [9]. In NILM, the goal is to map new data to a set of classifications, in this case, individual appliances, based on certain handpicked features [10]. The Naive Bayes classifier is based on the Bayes theorem and assumes independence between the states of each appliance. One advantage of using the Naive Bayes algorithm for NILM is that it only requires a small amount of training data to make predictions about the appliance class [4]. There have been several studies that have used the Naive Bayes classifier for NILM tasks and obtained good results [4]. In order to evaluate the performance of the classifier, a training dataset is used to train the model, and the prediction accuracy is measured using an unseen test set [10].

3.2.1.2 K nearest neighbours

The *K nearest neighbor* (KNN) algorithm is a machine learning method used to classify new objects based on the characteristics of their "nearest neighbors" [11]. KNN is considered a type of supervised learning, as well as a lazy learning, instance-based, or memory-based algorithm. It is relatively simple to implement and involves preparing data for training and testing, selecting a value for K, calculating the distance between the test data and the training data, sorting the distances from smallest to largest, and finally classifying the test data based on the K closest neighbors. In the context of NILM, the KNN algorithm can be used to classify appliances based on their power consumption characteristics [11].

3.2.1.3 Random Forest

Random Forest is an ML model that is made up of a collection of decision trees. It is designed to improve the stability and accuracy of decision trees by training multiple trees and combining their predictions [2]. Each tree in a Random Forest is trained on a different subset of the data and considers a randomly selected subset of the input variables at each split. This helps to ensure that the trees in the forest are diverse and that the model is more accurate. However, if the trees are too similar, the classification accuracy of the model may suffer. Random Forest addresses the instability of single decision trees, which can be affected by small changes in the data, by training multiple diverse trees and combining their predictions [2]. The work of Angelis et al. (2022) [4] presents Random forests as one of the best shallow learning algorithms, outperforming KNN, Naive Bayes, and deep learning algorithms.

3.2.2 Deep learning algorithms

Deep learning algorithms have proven effective in a range of applications, including speech recognition, machine vision, and asset condition monitoring [6]. Researchers have also started exploring their use in NILM to extract features from the aggregated signal without relying on manual feature extraction [4] and improve appliance classification and energy disaggregation. These algorithms have the ability to learn from and adapt to the characteristics of the data, which can improve the performance of NILM systems [6].

3.2.2.1 Feed Forward Network

Feed-forward neural networks (FFNNs) are a type of deep learning model that is organized into layers of interconnected nodes [4]. These nodes communicate with one another through connections called neurons, which have corresponding weights that determine their importance in the network. Raw data is input into the network through the input nodes in the first layer and is then transformed into feature vectors by the hidden layers. Finally, the output layer produces the desired output based on the processed data. FFNNs are the simplest and most fundamental form of deep learning models [4].

3.2.2.2 Convolutional neural networks

Convolutional neural networks (CNNs) are a type of deep learning model that has proven effective in a variety of applications [4]. They are made up of multiple layers, including convolutional layers, which perform the convolution process, and other types of layers such as pooling layers, normalization layers, activation function layers, flattening layers, and fully connected layers. These layers can be combined with other DNN layers, such as Recurrent neural networks, which will be discussed further in the following sub-section, to create effective models. In the field of energy disaggregation, CNNs have been used for both regression and classification tasks and have demonstrated strong results [4].

3.2.2.3 Autoencoders

Autoencoders are a type of neural network that can be used to extract features from data. They consist of three main components: an encoder, a decoder, and a latent space representation [4]. The encoder takes the input data and compresses it into a smaller representation, which is called the latent space. The decoder then takes this representation of the latent space and tries to figure out what the original data was. In the context of NILM, the encoder is used to create a latent space representation of the aggregate signal, and the decoder is responsible for attempting to recreate the sub-metered signal from this representation. Autoencoders are often used as a preprocessing step in machine learning pipelines, as they can help extract useful features from the raw data that can then be fed into other algorithms [4].

3.2.2.4 Recurrent neural networks and Autoencoders

Recurrent Neural Networks (RNNs) are a deep learning model that excels at processing sequential input [4]. They are able to incorporate previous state information as input to the current state but can be affected by issues such as gradient disappearing and explosion. As a solution, attention mechanisms have been introduced to improve the efficiency of RNNs by allowing the decoder to use a collection of features from the encoder's past states to build a final representation. While attention mechanisms have shown promising results in various tasks, outperforming RNNs in both accuracy and training time, they can be difficult to apply. In the field of NILM, many approaches based on RNNs have been proposed in recent years with the goal of improving system performance. RNNs have shown great potential in a variety of applications and can be a valuable tool in the field of NILM [4].

3.2.2.5 Deep Generative Models

Deep Generative Models are a type of deep neural network that are trained on a large amount of data in order to synthesize high-dimensional distributions. Two commonly used approaches are Variational Autoencoders and Generative Adversarial Networks [4]. Variational Autoencoders have been used to extract appliance-specific signals from the main signal in a process known as energy disaggregation. Generative Adversarial Networks have also been used for energy disaggregation, with one network synthesizing appliance-specific signals from a latent representation and another network attempting to determine whether these signals are real or fake. Some variations on this approach have also been proposed, such as the use of Gated Recurrent Units in the discriminator network and the concatenation of the main signal with the output of the generator network to improve training stability [4].

3.2.2.6 Transfer Learning

Transfer learning is a technique that allows a neural network to use its knowledge from one domain to another by utilizing a pre-trained model [4]. There are several ways in which transfer

learning can be applied, such as modifying the network's head, fine-tuning the full network, or using the pre-trained model as is. This technique has been widely used in various fields, but it has not yet been fully utilized in the domain of energy disaggregation due to the lack of well-established datasets and the diversity of appliances, datasets, and energy consumption behaviors. In order to address these issues, it is necessary to implement both appliance-specific and cross-dataset transfer learning in the field of energy disaggregation [4].

3.2.2.7 Federated Learning

Federated Learning (FL) is an approach that involves storing data locally on devices and training a shared model without the need to centralize the data [4]. One benefit of FL is that it reduces privacy and security risks by limiting possible attacks on individual devices. In the NILM domain, training data may contain sensitive information about clients, and the exposure of this data could lead to various privacy and security issues. FL can help address these issues and contribute to the development of more generalized models [4].

3.3 Datasets and Machine Learning

Datasets are crucial for training machine learning models, as they provide the necessary input data for the model to learn from. When it comes to NILM, datasets are especially important because they give the model the information it needs to learn how to accurately break down energy use by appliances. Without high-quality and diverse datasets, it is difficult to develop NILM models that accurately and reliably perform their tasks. In this section, we will present a variety of datasets, as presented by Angelis et al. (2022) [4], as well as techniques to overcome the limitations of their data.

3.3.1 Datasets

The datasets presented in Angelis et al. (2022) [4] are as follows:

- Almanac of Minutely Power Dataset, (Version 2) (AMPds/2) has 2 years' worth of consumption information for a single Canadian household, sampled every minute, and 21 power meters, 2 water meters, and 2 natural gas meters.
- Building-level fully labeled dataset for electricity disaggregation (BLUED) contains voltage and current data sampled at 12 kHz from a US domestic site for a week, with 50 appliances at 60 Hz and labeled state transitions for each appliance.
- Dataport contains aggregate power, real power, and sub-meter appliance level readings from 722 households and commercial buildings in the US from 2011 to 2015, sampled at 1-minute intervals.

- The Dutch Residential Energy Dataset (DRED) contains 6 months' worth of aggregate and sub-metered power readings from a single household in the Netherlands, with readings coming from 12 devices and sampled at a rate of 1 Hz.
- Electricity Consumption and Occupancy (ECO) contains active power, voltage, and current readings sampled at 1 Hz from 6 residential buildings in Switzerland for over 8 months, each with different appliances, data granularity, and deployment times.
- ENERTALK contains aggregate and individual active and reactive power readings, collected at 15 Hz from 22 households in Korea for 29 to 122 days; some readings are faulty due to "logs and meter" issues.
- Georges Hebrail UCI consists of 4 years' worth of aggregated and sub-metered power readings from a single household, sampled at a rate of 1 minute. The data includes readings of active and reactive power, voltage, and current, as well as readings from three sub-metering points in the house that correspond to different rooms.
- The Indian Dataset for Ambient Water and Energy (iAWE) contains voltage and current data sampled at 12kHz from a single Indian residential building for 73 days, with 33 sensors and 63 appliances.
- Plug-Level Appliance Identification Dataset (PLAID) has 1094 observations at 30 kHz and 11 different power consumption levels from 11 appliances in 56 domestic US sites, but no active or reactive power readings.
- Reference Energy Disaggregation Dataset (REDD) contains high-and low-frequency recordings, 119 days' worth of data from 6 residential structures in the US, and 92 measurements of home appliances.
- REFIT is a dataset from 20 households in the UK, always recorded for 2 years with a sampling period of 8s for active power, but 3 of the houses had solar panels, so their data is unusable for NILM.
- Synthetic Dataset (SynD) is a collection of simulated energy consumption data created specifically for energy disaggregation research in residential buildings. It uses data from 21 appliances that were sampled every 5 Hz and is based on how they were used and when they were used at two residential sites in Australia.
- UK-DALE: A dataset comprising power consumption data collected from five UK homes, contains over 10 types of appliances, with aggregate consumption frequencies ranging from 1 Hz to 16 kHz, depending on the household. All appliances are sub-metered at 1/6 Hz.

3.3.2 Data Harmonization and Augmentation

When developing a NILM system, it is important to consider that datasets may have problems or defects within their data, including handling missing values [4]. To solve this problem, we can

use data harmonization, such as Interpolation techniques, such as linear and spline interpolation, which can be used to calculate missing values in time-series datasets, leading to better, less biased datasets to be used for training [4].

Insufficient training data and imbalanced classes can be a problem for deep learning models, and data augmentation can be used to generate synthetic data by applying transformations to the original set [4]. However, care must be taken when applying these transformations to time-series data as it can easily be distorted or lose valuable information. There are several methods for augmenting time-series data, including manipulating the data in the time domain, mapping it to the frequency domain, and using generative models such as generative adversarial networks [4].

3.4 Conclusion

In summary, in this chapter, we discussed how the use of machine learning in the field of non-intrusive load monitoring has shown promising results in recent years. A variety of approaches, including deep and shallow learning methods, have been implemented and have achieved high levels of accuracy in energy disaggregation tasks. We also discussed data acquisition, such as feature types and their extraction, and the already available data in the form of datasets available for training algorithms. With the information on the current state of the needed technologies for this proposal, we will present the preliminary proposed solution for the system to be developed in the next phase of this dissertation.

Chapter 4

Proposal

In this chapter, we will go over the general proposition on how to solve the problem presented in this work. We were able to find the most recent works done on the NILM setting with the use of ML as the disaggregation method and how we might be able to apply these concepts on a cost-effective device thanks to the work done in the previous chapter. This section will also present how the implemented algorithm will be evaluated during the development of this work.

4.1 Solution Overview

As observed in chapter 3, many machine learning algorithms used in non-intrusive load monitoring require a significant amount of computational power to process and analyze large amounts of data in real time. These algorithms often involve complex mathematical calculations and operations on large datasets, which can be computationally intensive. For this proposal, it is imperative to find an algorithm that is efficient in terms of its computational power demands when implemented on a NILM system in resource-constrained environments such as microcontrollers.

The work of Angelis et al. (2022) [4] presents an algorithm that combines both accuracy in appliance identification and low power requirements, which are needed for the proposed solution of this work, in the form of the Random Forests algorithm. This approach has already been implemented in a NILM setting, in works such as Xiao et al. (2021) [12] and Wu et al. (2019) [2], with both finding compelling results in the disaggregation task.

Since the proposed algorithm is a shallow learning method, a process called "feature selection," which was presented in the previous chapter, will be needed to figure out which features are most important for identifying appliances in a typical household. These features will need to be measurable at a low frequency, since high-frequency devices are pricier, therefore defeating the purpose of a cost-effective device capable of proper energy disaggregation.

Furthermore, it will also be necessary to identify which datasets are most relevant in this setting in order to guarantee that the developed algorithm is highly accurate.

The developed NILM architecture will have the following requirements:

- **Computationally Fast**: the developed algorithm must be executable by systems with cost-effective hardware;
- **Memory efficiency**: The algorithm must use the least amount of memory as possible in order to ensure that it is lightweight;
- **Non-complex Features**: The selected features must be samplable and processed by cost-effective hardware;
- **Accurate disaggregation**: Despite the previous restrictions, the algorithm must still be reasonably accurate in its disaggregation;
- **Robust disaggregation**: The algorithm must be capable of disaggregating multiple types of loads.

By combining these ideas, we hope to achieve a NILM architecture capable of accurate energy disaggregation and appliance identification.

4.2 Implementation

In this section, we will go over the concepts of how this work will be modeled, including the software that will be designed, what hardware it will be tested on.

4.2.1 Hardware

Although it isn't the main focus of this thesis, it will be important to test the implemented ML algorithm on a real microprocessor to make sure it works as desired. As such, we won't be using any measurement equipment, such as smart meters, etc. This decision was made since there are readily available devices that sample the necessary features at a low cost, and due to there not being an available environment where the capturing and labeling of a household's electrical data for real-world tests.

When choosing the necessary hardware, we will need a cost-effective computational device, and for this project, the chosen device will be a Raspberry Pi 3 Model B. This device was chosen due to its low cost and its market availability. Market availability was an integral part of choosing this device since the current market instability meant that other devices might not be available during the time scope of this project.

4.2.2 Machine learning algorithm

As indicated above, the proposed disaggregation algorithm is the Random Forest algorithm. Random Forest algorithms are popular due to their ability to achieve high accuracy while also having relatively low computational power requirements. A random forest is an ensemble learning

method that trains a group of decision tree models on different subsets of the data and combines their predictions to make a final prediction [12].

A decision tree is a model for making decisions based on the features of the input data. Each internal node represents a feature, and each branch represents a decision based on that feature. The tree is made by recursively splitting the input data into groups based on the most distinguishing feature at each step. The goal is to reduce the impurities in the nodes as much as possible. The final prediction is made by traversing the tree from the root node to a leaf node, with the value at the leaf node being the prediction made by the tree.

The individual decision trees in the random forest are relatively simple and can be trained and evaluated quickly, making the overall random forest algorithm efficient in terms of computational power [2]. At the same time, combining several decision trees can lead to very accurate predictions because the random forest can find complex patterns in the data that a single process decision tree might not be able to find [2].

The process of training a random forest begins by selecting a random subset of an input dataset, which is used to train the first decision tree in the forest [12]. This process is then repeated for each of the remaining trees in the forest, with each tree being trained on a different subset of the data [12], this process is expressed in figure 4.1. The randomness in the random forest algorithm comes from the fact that each tree is trained on a different subset of the data, which helps to reduce the risk of overfitting and improve the generalization performance of the model [12].

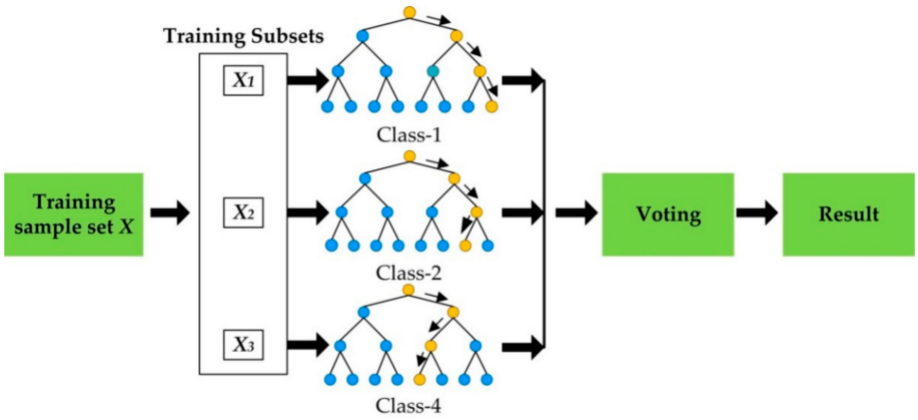


Figure 4.1: Random Forest training process [2]

For this project, we will be using a Random Forest regression algorithm [13]. Contrary to classification tasks, which are trained to have discrete labels for their inputs, regressions can have continuous values for their labels, which allows us to train the random forest to predict the continuous power usage of specific appliances.

4.2.3 Datasets

The selection of the dataset is an important factor to consider when developing the NILM disaggregation algorithm. The dataset used for training and evaluating the algorithm has a significant

impact on its performance, and it is crucial to choose a dataset that is representative of the target application and has sufficient diversity and complexity to challenge the algorithm.

If the dataset is not representative of the target application, the NILM algorithm may not generalize well to real-world situations and may not perform accurately on unseen data. Similarly, if the dataset is not diverse or complex enough, the algorithm may not be able to learn the necessary patterns and relationships to accurately perform appliance disaggregation. It is also important to note that no dataset will be able to account for all possible households, and as such, the chosen dataset for initial training must be as wide-reaching as possible, so that we may be able to accurately access the performance of the implemented algorithm.

Therefore, it is essential to carefully select the dataset to ensure that it is suitable for the development of a robust NILM algorithm. This may involve collecting or curating a dataset specifically for the purpose of NILM, or it may involve using a publicly available dataset that has been carefully selected to be representative of the target application.

4.2.4 Proposed features

To determine the optimal number of features for a machine learning model we will use a filter feature selection algorithm, as presented in the previous chapter, to iterative reduce the number of features and evaluate the impact on model accuracy. The goal of the feature selection process is to identify the smallest subset of features that can still achieve good accuracy, as using a larger number of features can lead to overfitting and decreased generalization performance. However, as indicated in the hardware section of this chapter, not all features will be available for sampling, due to the associated hardware cost for their measuring. As such, **If** the chosen dataset does not contain a large number of features or the features it contains are all important to the disaggregation process, this step may be entirely skipped.

4.3 Performance metrics

We need to use performance metrics to test the model to see if the trained algorithm has reached a good enough state. The main metric to be used in this work will be the *Mean Square Error* (MSE). MSE is a common metric used in machine learning as a way to evaluate the performance of regression Tasks. The mathematical formula for MSE is as follows:

$$MSE = \frac{1}{N} \cdot \sum_{n=1}^{N} (y_n - \bar{y}_n)^2$$

In this formula, N represents the total number of points in the considered set of data, y_n is any actual value of a label, sampled from any of the N points in the dataset, and $\bar{y}_n)^2$ represents the corresponding predicted label value for that specific point. As we can incur by this formula, the value of this metric is more heavily impacted by larger error values than the low ones, since the value of said error is squared.

By using this as our main performance metric, we can tailor our training process of the Random Forest Regression in order to minimize these larger prediction errors, while disregarding the smaller ones, considering that, in the NILM application, small errors in the power consumption are not relevant to the average consumer.

However, it's important to remember that one single metric might not be enough to fully show how well the model works. As such, additional metrics will be used in order to evaluate the performance of the algorithm. These metrics will be acquired by a Box plot of the error. A Box plot is a graphical way to display a spread of values with statistical significance. An example of a Box plot is presented in figure 4.2. As we can see, the first quartile (Q1), the third quartile (Q3), the median, the whiskers, and the outliers are identified in this example. The median can be used as a performance metric since it indicates the most common error for a specific label. Q1 and Q3 show us the range of error for 50% of the most common data, allowing us the see the usual range of error. The other 50% are within the whiskers and outliers. The values within the whiskers are at a maximum distance of the nearest quartile of 1.5 times the interquartile range, which is defined as the distance between Q3 and Q1. On the other hand, the outliers are error values far removed from the norm, with absolute values higher than the previously presented threshold. The percentage of outliers will be used as a performance metric, as a low number of outliers indicate an accurate disaggregation process.

Additional to these statistical metrics, we will also keep track of the algorithm's memory usage, in order to ensure that it is executable in the chosen platform. As such, we end up with four considered metrics, these being: MSE, median error, percentage of outliers, and memory usage.

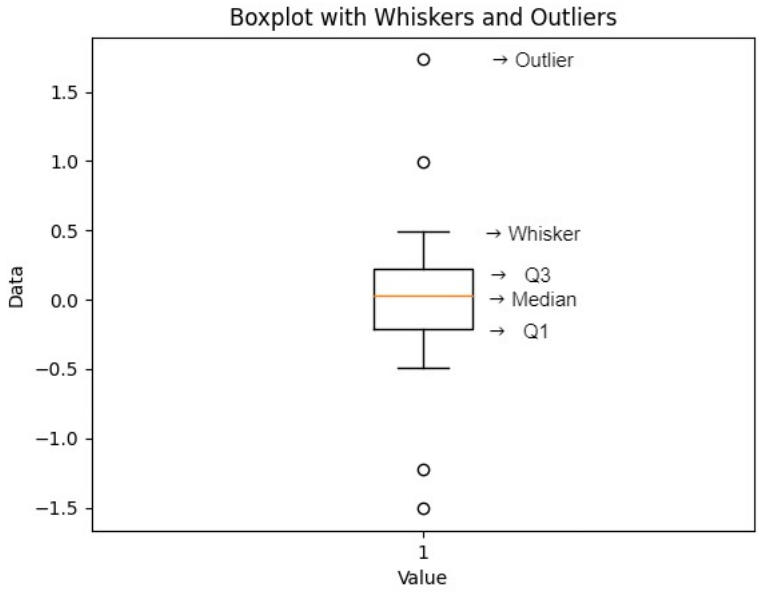


Figure 4.2: An example of a labeled Box Plot

4.4 Conclusion

In this chapter, we went over a general overview of how we would implement a NILM architecture capable of functioning on a cost-effective device. Besides specifying the microprocessor to be used in testing, we also selected the Machine learning algorithm to be used for the disaggregation process, chosen due to being a lightweight implementation with a performance comparable to other more computationally taxing algorithms. We went over the requirements for the chosen dataset used to model our algorithm, as well as how the considered electrical features would be selected. Finally, this chapter also explains how the performance of the algorithm will be evaluated, as well as the chosen performance metrics used in said evaluation.

Chapter 5

Development

In this chapter, we will go over the specific methods used to develop the proposal presented in the previous one. Research was done in order to find the best-suited technologies in order to properly develop the random forest regression and then deploy it to the chosen cost-effective device. This chapter will discuss the library used to implement the algorithm, the dataset, and how its data was preprocessed, as well as the specific procedures used to guarantee a good performance of the disaggregation algorithm, and its adaptations for the specified hardware.

5.1 Scikit-learn

Scikit-learn is a Python library that provides the implementation of various common machine learning algorithms for classifications and regressions, among others, while maintaining the low complexity and ease of use [14].

This library was chosen to implement the *Random Forest* regression algorithm because it provides multiple benefits to the development of this dissertation, such as its distribution being under BSD licensing, which allows for unrestricted use for academic and commercial purposes [14]. Its code is also of high quality, as all contributions require at least 90% testing coverage to be accepted [15]. The available documentation is also extensive, containing detailed descriptions and use cases of their classes and methods, as well as their parameters and attributes [14]. The library also only depends on two other common Python libraries, *Numpy* and *SciPy*, making its implementation on systems containing these two libraries simple [14].

5.2 ECO Dataset

The ECO dataset was chosen for the development of this dissertation due to its relevant characteristics. For starters, it contains data collected over an 8-month period, which is a longer time span than most other data sets presented in chapter 3, with only the AMPds/2, Dataport, and Georges Hebrail UCI covering a similar or larger period. Second, the aggregate electricity consumption data provided with the ECO data set contains multiple electrical features, such as measurements

of real power, voltage, and current for each of a household's three phases [3]. The dataset also contains labeled sub-metered data for the appliances in all households [16]. Finally, in the ECO dataset, plug-level data at 1 Hz frequency was collected for each household. This complies with the requirement for a cost-effective device since high-frequency sampling would require higher processing power.

5.2.1 Data Description

As presented before, this dataset contains the data for the electricity consumption in 8 different households. All of these household's captured electrical features are identical, with [16] listing them as such, with **X** representing phases 1, 2, and 3, reaching a total of 16 electrical features:

- powerallphases: Sum of real power over all phases
- powerlx: Real power phase **X**
- currentneutral: Neutral current
- currentlx: Current phase **X**
- voltagel1: Voltage phase **X**
- phaseanglevoltagel2l1: Phase shift between the voltage on phase 2 and 1
- phaseanglevoltagel3l1: Phase shift between the voltage on phase 3 and 1
- phaseanglecurrentvoltagelx: Phase shift between current/voltage on phase **X**

For the purposes of this dissertation, **household 2** was selected as the training model for the Random forest algorithm. This specific household was selected due to the fact that it contains the biggest number of sub-metered appliances, along with the lowest amount of unmeasured power consumption on an appliance level, as can be seen in figure 5.1. The lower amount of unaccounted appliances in the household reduces the data complexity and betters the disaggregation process.

As can also be seen in figure 5.1, **household 2** possesses 10 sub-metered appliances, however, the ECO dataset [16] presents 12 labeled individual power consumption. This is due to a combination of aspects, such as the power consumption of the "TV" and "Stereo" being measured by a single power meter, listed in the dataset as "Entertainment", and then manually disaggregated in order to obtain the individual consumption. The dataset also lists the "air exhaust" separated from the "stove" since it was not possible to sub-meter the stove directly. The power consumption for the stove was acquired manually by using the air exhaust data to detect when the stove was turned on and by knowing that the stove is the only appliance to consume power in two phases.

Considering all this, we then get the labels as presented in table 5.1, alongside the number of days that the data was metered for and their data coverage. Data Coverage is the percentage of sampled data that didn't return a missing value.

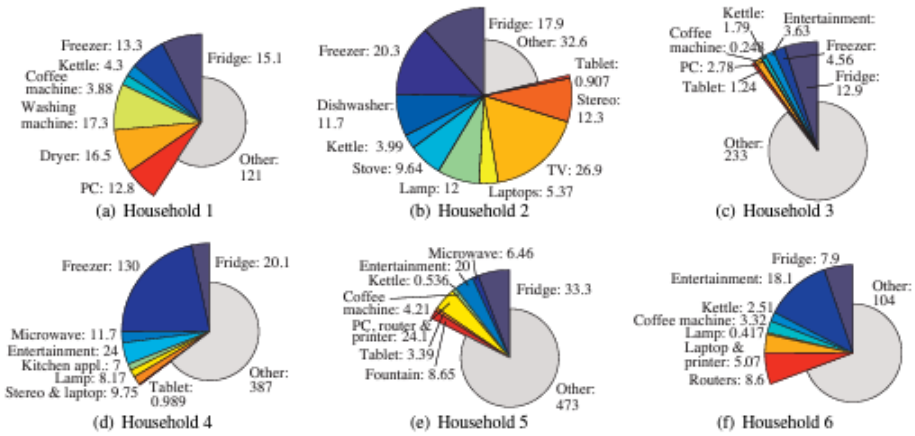


Figure 5.1: KW/h per appliance for all households [3]

	Tablet	Dishwasher	Air exhaust	Fridge	Entertainment	Freezer
N Days	240	240	240	240	240	240
Coverage	97.43%	97.09%	96.18%	98%	96.18%	96.39%

	Kettle	Lamp	Laptops	Stove	TV	Stereo
N Days	240	240	240	28	240	240
Coverage	88.5%	90.21%	83.36%	100%	100%	95.95%

Table 5.1: Number of sampled days and percentual coverage per label

5.2.2 Data Preprocessing

Data preprocessing is a crucial step in working with the ECO dataset and the *Scikit-learn* Python library, the files present in the dataset [16] aren't immediately usable to train the *Random Forest Regression*. The ECO dataset not only has the features and labels in separate *Comma-Separated Values* (CSV) files but each individual label is also separated into individual CSV files. These files are organized by date, with each file containing 86400 measurements, which is equivalent to the indicated 1Hz sampling rate. In order to use the ECO dataset, four Python scripts were designed in order to preprocess the data so that it becomes usable.

Scripts A.1 and A.2 are responsible for reorganizing the data by renaming the individual label data files and grouping them into individual files containing the data for all labels for a single day, respectively. Script A.3 is responsible for finding missing values in the grouped label files and removing them. Initially, the cleaning script was meant to interpolate the probable consumption in a missing value, however, considering that the Random Forest algorithm does not use a window for its features, considering only the instantaneous measurements, and the fact that some of the missing values spanned several hours, such as the ninth label, the laptops, on the 7th of July of 2012, where 81484 consecutive missing measurements were captured [16]. These facts create an environment where data interpolation is not only not necessary, due to a large amount of non-faulty available data, but also not desirable, due to the error introduced by any interpolation in these large intervals. Despite also containing missing values, the feature files are not examined for

missing values, this is due to the fact that the Random Forest algorithm will, by its design, choose the features that best split the data, therefore being tolerant of missing values in its features.

The final script, A.4, has the purpose of joining the clean data for every single day into two files for features and labels, allowing them to be used in the training process. Both final files contain more than $1.52 * 10^8$ individual measurements, which is more than enough to train the desired algorithm.

5.3 Implementation

The following code was implemented in order to implement the Random forest Regression:

```
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

agg_data_dir = "drive/MyDrive/master_data.csv" /*Swap for your directory containing
     the aggregated measurements data/*
label_dir = "drive/MyDrive/master_label.csv" /*Swap for your directory containing
    the label data/*

agg_data_matrix = np.loadtxt(agg_data_dir , delimiter=",")
label_matrix = np.loadtxt(Label_dir + j, delimiter=",")


agg_data_matrix_train, agg_matrix_test, label_matrix_train, label_matrix_test =
    train_test_split(agg_matrix, label_matrix, test_size=0.20)

reg = rf = RandomForestRegressor(n_estimators=16, max_depth=20, random_state=42)
reg.fit(agg_matrix_train, label_matrix_train)


label_matrix_pred = reg.predict(agg_matrix_test)
```

Listing 5.1: Scikit's Random forest Regressor implementation

As we can see in the code, in order to implement the *Scikit-learn Random Forest Regression* algorithm, we must first load the measured aggregated data and the labeled load-specific power data into *Numpy* arrays. In this implementation, this is achieved by loading this data from local CSV files using the *Numpy* "*loadtxt*" method.

Upon loading this data we then proceed into separating it into two categories, Train and Test data. Train data will be used by the *Random Forest* algorithm to create its random decision trees, and Test data will be used to evaluate the quality level of the predictions made by the regressor, by predicting data not included in the training batch. This separation is made using *Scikit-learn's* "*train_test_split*" method, randomly separating the original data into proportions as indicated by the "*test_size*" parameter.

With the separated data, we can finally begin the training process with the "*fit*" method. In this step, the algorithm first starts by creating random sets of data via Bootstrap Sampling. As presented in image 4.1, Bootstrap Sampling is a process by which the original data is sampled at random, with a seed defined by the "*random_state*" parameter, in order to generate individual data sets. In the *Scikit-learn* Random Forest implementation, The generated bootstrap samples have the same length as the original data [17], This allows for unbloated memory usage by the generated Random Forest.

As per the information in the previous chapter, the following step in *Scikit-learn* consists of the division of the branches of the generated decision trees in a way that minimizes the MSE until we reach a specified max depth of 20, as specified by the "*max_depth*" parameter. This depth was reached via an iterative process, by which the "*maxdepth*" was increased until a bottleneck was reached, in which the mean square error did not significantly decrease in comparison to the increase in memory used, as can be seen in table 5.2. It is also notable that these MSE values were measured on the same random subset of data and are not indicative of the actual performance of the algorithm in the real NILM application.

Max Depth	Mean Square Error	Memory usage
15	313.53	35.55 MB
16	293.92	55.26 MB
17	277.11	84.49 MB
18	264.81	126.49 MB
19	251.91	183.69 MB
20	242.26	256.98 MB
21	235.09	358.28 MB

Table 5.2: Impact of Max Depth on MSE and Memory usage

As presented in Chapter 4, another important metric of Random Forests is the *number of decision trees*, and in *Scikit-learn's* Random forest, that number is specified by the "*n_estimators*". As with the "*max_depth*", the "*n_estimators*" was also reached in an iterative process, where the number of decision trees was decreased to a point where the memory usage was reduced, but the MSE wasn't significantly increased until 16 was reached as the optimal number of decision trees, as can be observed in table 5.3. As with the previous process, these tests were done on a random, persistent subset of data and don't necessarily reflect the performance of the algorithm in the actual NILM application.

N of Decision trees	Mean Square Error	Memory usage
64	282.71	693.07 MB
32	293.92	346.02 MB
16	291.74	173.66 MB

Table 5.3: Impact of the Number (N) of decision trees on MSE and Memory Usage

5.3.1 Hardware Implementation

The hardware constraints of a Raspberry Pi 3 Model B must be carefully taken into account while implementing the random forest regression, considering that it is a single-board computer with a quad-core Broadcom BCM2837 processor running at 1.2GHz and 1GB of RAM.

Multiple decision trees must be trained for random forest regression, which can be computationally demanding. In order to make sure that the developed algorithm is executable on this cost-effective device, it was necessary to take two additional measures, as well as the two discussed before.

The first measure was to limit the size of the training batch of data so that we can reduce the amount of memory the generated *Scikit-learn's* Random Forest Regressor object isn't bloated. By applying the previously obtained 16 "$n\_estimators$" and 20 "max_depth", the amount of data was similarly reduced, iteratively, until reaching a state that the lowest possible MSE, whilst maintaining memory usage below 512 MB, so that it isn't larger than half of the Raspberry Pi's RAM. This process generated the following results presented in table 5.4. As we can see in the table, we do not reach the maximum allotted memory even while using 80% of the established training data acquired in the ECO dataset, this is mainly due to the already low max depth, which proves to be the biggest bottleneck in terms of memory usage.

Percentage of Data	Mean Square Error	Memory usage
10%	287.65	176.62 MB
20%	243.96	225.62 MB
30%	224.22	256.98 MB
40%	209.83	280.50 MB
50%	199.02	299.88 MB
60%	193.86	313.42 MB
70%	186.60	331.69 MB
80%	179.23	339.70 MB

Table 5.4: Impact of the amount of data used in the training process, expressed as a percent of the total data, on the MSE and Memory usage

The second measure is to export the already trained algorithm directly into the Raspberry Pi. This step is necessary since while the board can load the algorithm into its RAM, it cannot load the necessary training data, which occupies 6GB and 4GB for the feature and label CSV files, respectively. This is solved by using the *Joblib* library. *Joblib* is a Python library that simplifies the operation of computationally heavy calculations by providing efficient serialization of Python objects. With *Joblib*, serialized objects can be saved to the disc and restored back into memory, allowing for seamless object sharing across processes and machines. This allows us to use machine learning algorithms that require powerful hardware in their training process in machines that usually wouldn't be able to handle them.

The Code Developed for this export and import process is as follows:

```
#------------------Dump the Previously Trained Regression----------------------#

from joblib import dump
dump(clf, 'drive/MyDrive/random_forest_regression_16_20_80.joblib')

#------------------Load the Regression into a new machine----------------------#
from joblib import load
from sklearn.ensemble import RandomForestRegressor
import numpy as np
import time


reg = load('drive/MyDrive/random_forest_regression_16_20_80.joblib')

agg_matrix = np.loadtxt("Directory_containing_Test_feature_data", delimiter=",")

for i in agg_matrix:
  label_matrix_pred = reg.predict(agg_matrix_test)
  print(label_matrix_pred)
  time.sleep(1)
```

Listing 5.2: Scikit's Random forest Regressor implementation

As we can see, the code presented above is divided into two distinct sections. The first section simply dumps the already trained regression from the RAM to the disk, saving it as a "*joblib*" file. This file was then transferred to a Raspberry Pi 3 Model B, running the 32-bit Raspberry Pi OS version 6.1, based on Debian 11 (bullseye). With the file in the platform, the second part of the script was executed in order to simulate the hardware sampling the data at 1Hz, and then disaggregating the measured features in real time. The electrical feature data used to simulate the real environment was sampled from the data in the ECO dataset.

The algorithm executed flawlessly in the intended platform, with the maximum occupied RAM never passing the **700MB** even with the underlying OS tasks running in parallel with the algorithm, thus, this proves the Viability of Random Forest Algorithms in cost-effective devices.

5.4 Conclusion

In this chapter, we specified the Python libraries and dataset used to develop a lightweight *Random Forest Regressor*. We explained in detail the contents of the ECO dataset, along with how said contents were handled in order to be usable in order to train the disaggregation algorithm. We also went over the iterative process that allowed for the developed algorithm to not be bloated nor over-fitted, as well as the specific measures taken in order to allow the algorithm to run the Raspberry Pi. We concluded that this specific implementation of a *Random Forest Regressor* can run in the microprocessor and can now test its performance in depth.

Chapter 6

Experimental Results

In this chapter, a series of tests will be presented and evaluated in order to ascertain the quality of the disaggregation performance. We will start by testing the data on simulated data, in order to find what aspects of the data result in a performance loss in the disaggregation process. After the tests on simulated data, we will then use the real data, obtained in the ECO dataset [16], in order to validate the relevancy of the data acquired in the simulation environment.

6.1 Simulation environment

In this section, we will be presenting the software used to simulate electrical loads, as well as the types of loads that were simulated and how they were connected in order to simulate a household's electrical network.

6.1.1 PSIM

PSIM, also known as PowerSIM [18], is a powerful simulation software typically used for power electronics simulation and development, which allows engineers to simulate electronic circuits. This software allows us to design power circuits by selecting and connecting preset power electronic components in order to simulate their behavior.

This software was chosen due to its robustness, allowing the users to simulate complex circuits with multiple loads at the same time as well as what features are going to be probed at a time, and its versatility, which allows us to graphically change which loads are connected to the simulated network and the amount of equivalent real-world time that is going to be simulated. The license for this software was also already available to FEUP, which allowed the development of the tests without any cost overhead.

6.1.2 Simulated Loads

In order to simulate the loads needed for the tests, we must first acknowledge the load types mostly present in people's homes. In Chapter 3, we present the notion that loads can be divided into four

categories with different power behaviors and different features needed in order to identify said behaviors.

However, most Portuguese households only contain loads of type I, II, and IV, with type III being reserved for niche electronics such as dimmable lights. Although type IV loads are common, with most homes containing appliances such as internet modems and TV receivers, these aren't very interesting from a disaggregation point of view, as the *Random Forest Regression* would always predict their power consumption as constant, due to the training data pointing that way. If these loads were simulated as turning off for some period of time, then they would be identical to type I loads. As such, types III and IV were disregarded in the simulation process, mainly focusing on appliances of type I and II.

It is also important to consider that in most Portuguese households, most appliances have either resistive or inductive behavior. As such, we will be simulating 4 different load sets, in which each load set will be either purely resistive or inductive. These loads will be connected to a single phase, operating at 230Vrms and with a frequency of 50Hz.

Appendix B contains the developed schematic, along with a brief explanation of the loads there present.

6.1.3 Simulated Features

For these simulations, we are going to be emulation the same conditions in which the data for the ECO dataset was captured. As indicated in previous chapters, the features in the said dataset were sampled at 1Hz. When it comes to electrical features, the ECO dataset obtained them in a household with three electrical phases, but in order to expedite the simulation process and more closely emulate the typical Portuguese household, the simulations were made in an environment with a single phase, reducing the 16 electrical features down to five, these being Vrms, Irms, P, S, and power factor.

6.2 Simulation Categories

Having established what loads we are going to simulate, it is essential to establish the tests we are going to perform on them. In order to ascertain disaggregation quality for all the simulated load types, we must create different combinations of the loads for simulation. As we can see in Appendix B, there are five for each of the Type I Loads and three for the Type II ones. Additionally, each of the sets of loads contains one high-power load in order to assess the impact of said high consumption on the quality of the *Random Forest Regression*. We will be performing the following tests for the ON/OFF loads:

- 1 Low power load: to create a performance baseline in a simple environment;
- 3 Low power loads: to assess the drop in quality when processing similar loads;

- 1 High power load: to evaluate the impact of high power consumption in the disaggregation process;
- 1 High power and 1 Low power load: to determine the changes in behavior when disaggregating loads with different power consumption;
- 1 High power and 4 Low power load: as a final more complex test to better approximate a real-world scenario.

As for the FSM loads, only three different tests were performed, this was mainly due to the time scope of this dissertation, considering each test took nearly three hours to simulate for each test. The tests are as follows:

- 1 Low power load: to compare to the behavior of the equivalent ON/OFF loads;
- 1 High power load: to evaluate if the quality of disaggregation is similarly affected by the high-power;
- 1 High power and 2 Low power load: as another complex test to put more strain on the algorithm.

All these tests also be repeated with a noise signal coupled to the input voltage, in order to add data complexity and better approximate the simulations to a real-world scenario and ascertain the impact of input noise in the disaggregation process. The noise signal is composed of the sum of four sine waves with arbitrary frequencies from 24.3Hz to 767 Hz and amplitude that would cause fluctuation of the input signal in $\pm 14\%$. this amplitude error was chosen so that the effects of the noise would be obvious to the naked eye.

There will also be two final tests made in order to access the quality of the *Random Forest* disaggregation algorithm. The first one will consist of simulating all previous loads at the same time, both with and without noise, in order to observe how the algorithm performs with the leap in data complexity. The second one will use the ECO dataset, which will allow us to see how accurately our conclusions on simulated data relate to real-world data.

6.3 Performance Analysis

In this section, we will be analyzing the performance of the *Random Forest Regression* in the various envisioned tests. As indicated in chapter 4, this dissertation considers four individual performance metrics, these being MSE, median, percentage of outliers, and memory usage, however, for this part of the dissertation memory usage no longer has any relevance, as the simulated data is so reduced in comparison to the dataset that the trained regression object would never pass the threshold defined by hardware. For tests containing multiple loads, there will be a global MSE, as well as an MSE for each individual load in the test.

This section will also be divided into individual subsections per set of tests, as indicated previously in this chapter, containing the loads used for that specific set. Each subsection will also be further divided per individual test, with each test containing the performance metrics for that test, a box plot for the error, and a behavior plot comparing the predicted signal (in red) to the real simulated signal (in blue).

6.3.1 Resistive Type I Loads

In this subsection, we will be analyzing the following set of loads in table 6.1. As mentioned before, these loads were designed so that we could have both low-power and high-power loads. As can be seen in Annex B, these loads were turned ON and OFF by a square wave with arbitrary frequency ranging from 10 mHz to 0.1Hz, and arbitrary duty-cycle ranging from 0.3 to 0.85.

	R1	R2	R3	R4	R5
Resistance	2000 Ω	3000 Ω	2500 Ω	25 Ω	250 Ω
Power consumption	13.7 W	9.1 W	10.9 W	1,09 kW	110 W

Table 6.1: Resistance and power consumption of all 5 loads

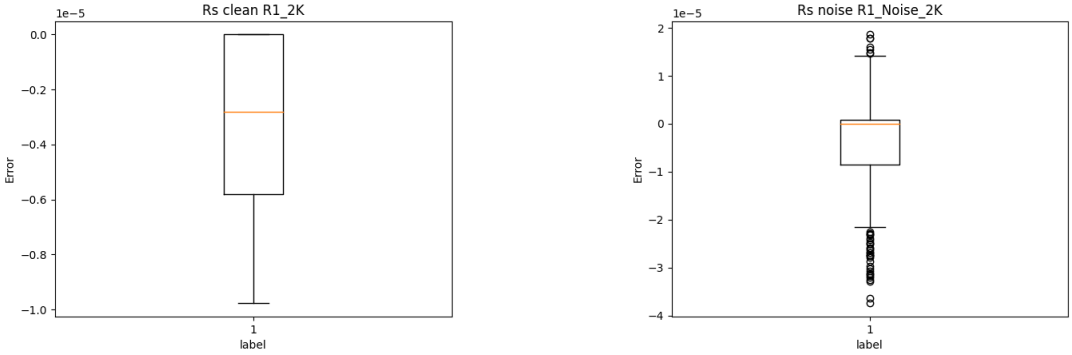
6.3.1.1 Test RI: 1 Low power load

For this test, the only considered load is **R1**, obtaining the following results:

	Clean Mean Square Error		
	2.14e-11		
Clean	MSE	Median	% of Outliers
R1	2.14e-11	-2.80e-6	0.0

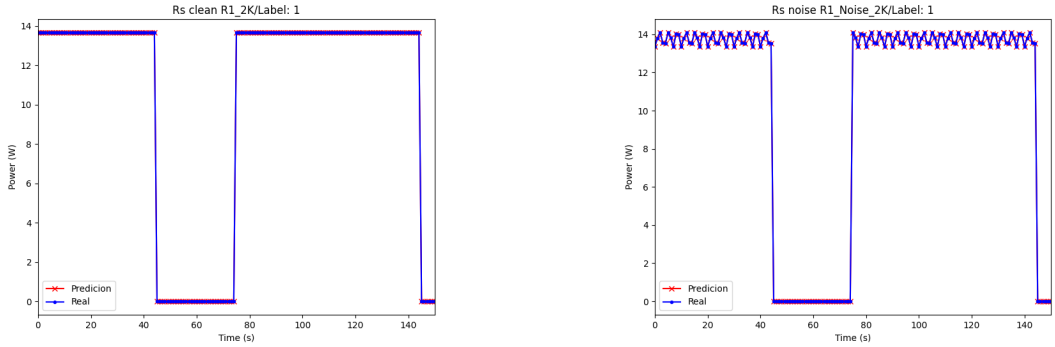
	Noise Mean Square Error		
	1.16e-10		
Noise	MSE	Median	% of Outliers
R1	1.16e-10	-9.33e-9	9.25

Table 6.2: Performance metrics for Test RI



(a) Clean Data. (b) Noise data.

Figure 6.1: Box Plot for the R1 load in Test RI



(a) Clean Data. (b) Noise data.

Figure 6.2: Behaviour plot for the R1 load in Test RI

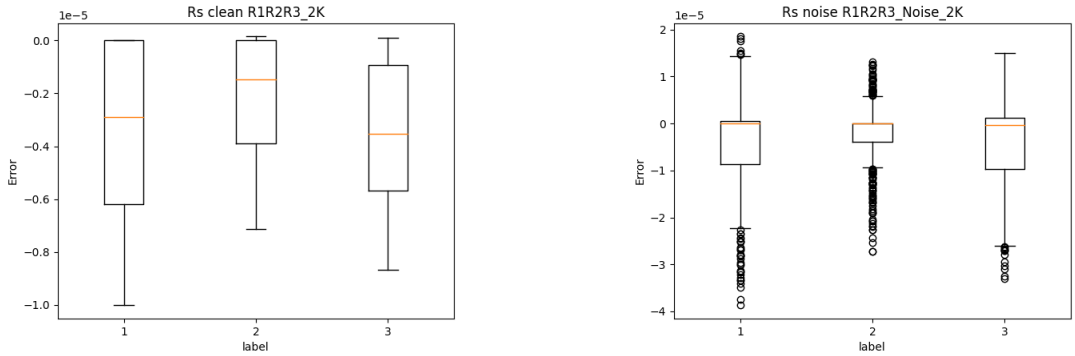
6.3.1.2 Test RII: 3 Low power load

In this test, we will consider loads **R1**, **R2**, and **R3**

	Clean Mean Square Error		
	1.67e-11		
Clean	MSE	Median	% of Outliers
R1	2.26e-11	-2.88e-6	0.0
R2	9.26e-12	-1.48e-6	0.0
R3	1.83e-11	-3.53e-6	0.0

	Noise Mean Square Error		
	9.67e-11		
Noise	MSE	Median	% of Outliers
R1	1.26e-10	-2.12e-22	9.5
R2	5.37e-11	-4.24e-22	25.25
R3	1.10e-10	-3.74e-7	3.0

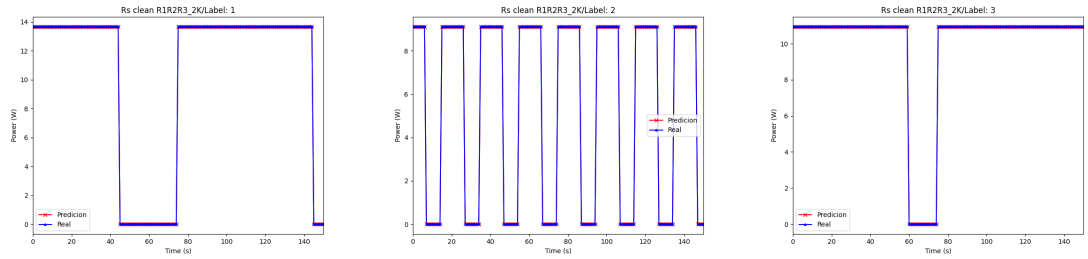
Table 6.3: Performance metrics for Test RII



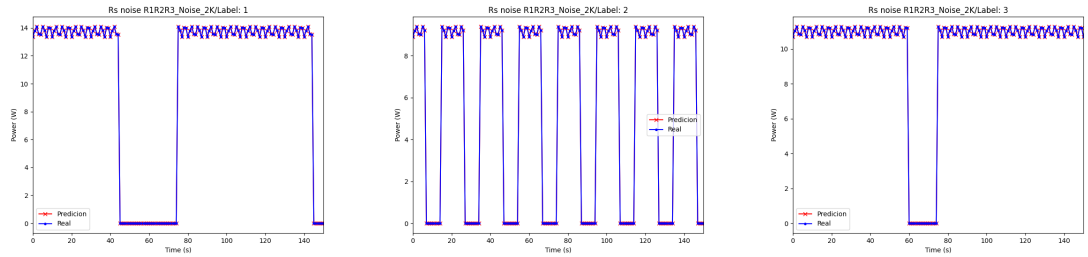
(a) Clean Data.

(b) Noise data.

Figure 6.3: Box Plot for the R1, R2, and R3 loads in Test RII



(a) Clean Data.



(b) Noise data.

Figure 6.4: Behaviour plot for the R1, R2, R3 load in Test RII

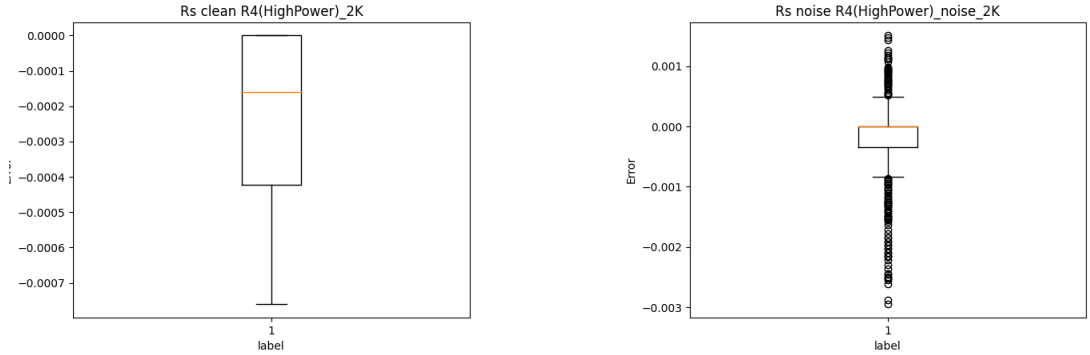
6.3.1.3 Test RIII: 1 High power load

In this test, only the high-power **R4** is considered:

	Clean Mean Square Error		
	1.16e-7		
Clean	MSE	Median	% of Outliers
R4	1.16e-7	-1.60e-4	0.0

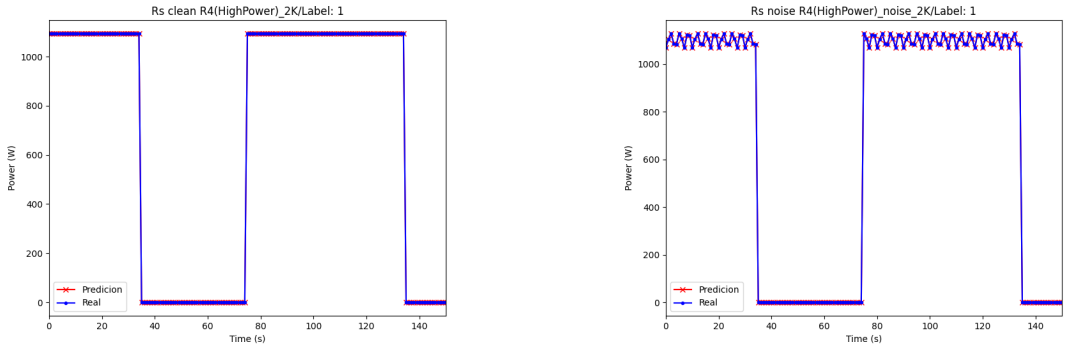
	Noise Mean Square Error		
	6.38e-7		
Noise	MSE	Median	% of Outliers
R4	6.38e-7	-3.98e-10	27.75

Table 6.4: Performance metrics for Test RIII



(a) Clean Data. (b) Noise data.

Figure 6.5: Box Plot for the R4 load in Test RIII



(a) Clean Data. (b) Noise data.

Figure 6.6: Behaviour plot for the R4 load in Test RIII

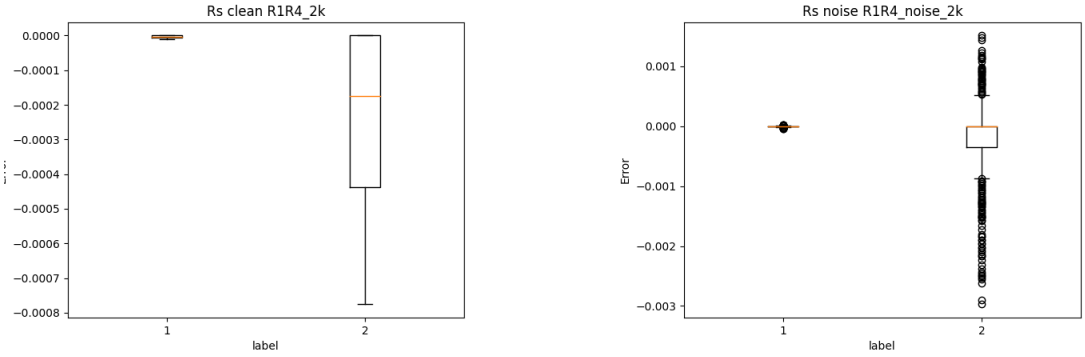
6.3.1.4 Test RIV: 1 High-power and 1 Low-power load

This test considers loads **R1** and **R4**, obtaining the following results

	Clean Mean Square Error		
	6.12e-08		
Clean	MSE	Median	% of Outliers
R1	2.35e-11	-2.93e-6	0.0
R4	1.22e-7	-1.74e-4	0.0

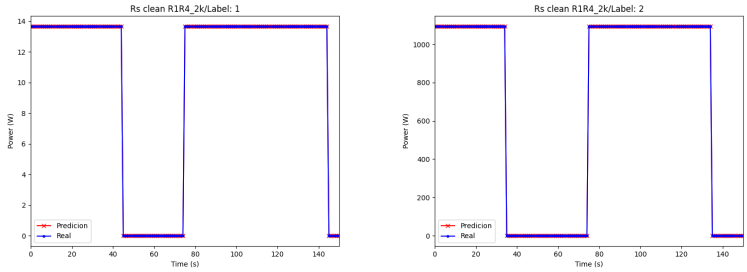
	Noise Mean Square Error		
	3.20e-7		
Noise	MSE	Median	% of Outliers
R1	1.30e-10	-1.51e-8	10
R4	6.40e-07	-8.28e-12	27.0

Table 6.5: Performance metrics for Test RIV

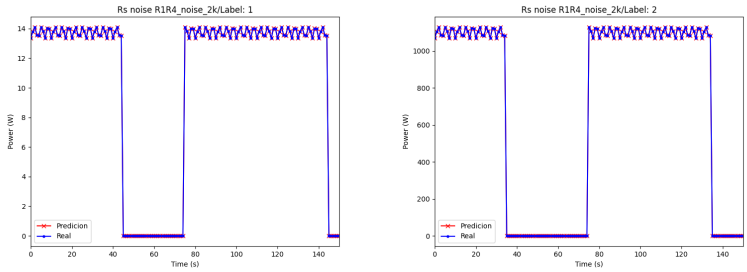


(a) Clean Data. (b) Noise data.

Figure 6.7: Box Plot for the R1, and R4 loads in Test RIV



(a) Clean Data.



(b) Noise data.

Figure 6.8: Behaviour plot for the R1, and R4 load in Test RIV

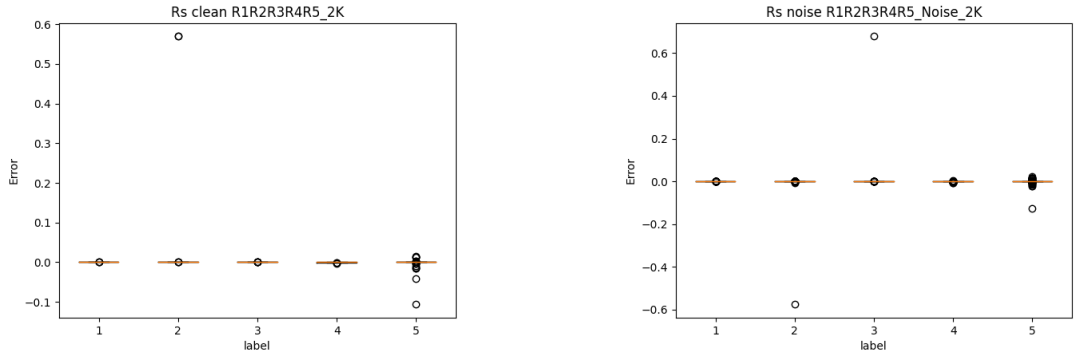
6.3.1.5 Test RV: 1 High-power and 4 Low-power load

This test considers all loads presented in table 6.1:

	Clean Mean Square Error		
	3.31e-04		
Clean	MSE	Median	% of Outliers
R1	2.83e-11	-2.84e-6	0.75
R2	1.62e-3	-1.63e-6	1.25
R3	2.65e-11	-3.72e-6	1.25
R4	1.56e-7	-1.72e-4	0.75
R5	3.66e-5	-2.38e-6	4.25

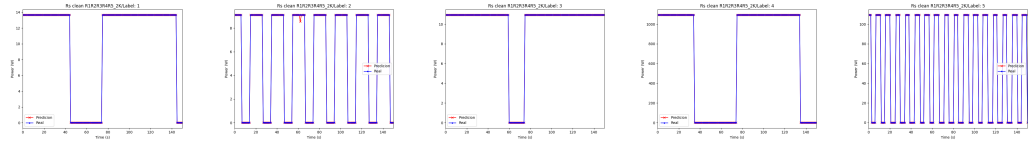
	Noise Mean Square Error		
	4.06e-4		
Noise	MSE	Median	% of Outliers
R1	3.00e-10	-2.12e-22	11.25
R2	8.30e-04	-4.24e-22	27.5
R3	1.15e-03	-2.09e-7	5.75
R4	1.25e-06	-1.65e-24	25.25
R5	5.14e-05	0	35.5

Table 6.6: Performance metrics for Test RV

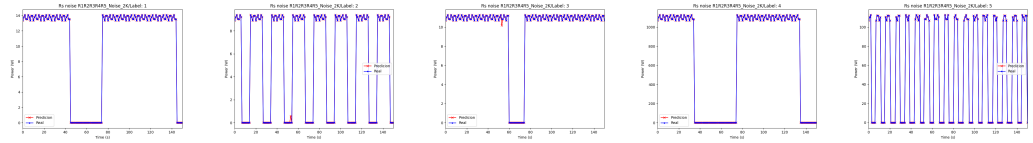


(a) Clean Data. (b) Noise data.

Figure 6.9: Box Plot for the R1, R2, R3, R4, and R5 loads in Test RV



(a) Clean Data.



(b) Noise data.

Figure 6.10: Behaviour plot for the R1, R2, R3, R4, and R5 loads in Test RV

6.3.1.6 Conclusions

Test RI shows a low MSE and Median error for both clean and noise data, showing that the disaggregation procedure performs well. The clean data's lack of outliers reinforces the accuracy of the disaggregation process, however, in the presence of noise, 9.25% of outliers are found, indicating some difficulties in managing noisy inputs. The global MSE score for clean and noise data in Test RII matches Test RI, maintaining the same order of magnitude. Clean data load-specific metrics match the prior test, maintaining strong disaggregation quality. The noise data indicates a sharp contrast, with a significantly smaller median error for loads **R1** and **R2** and a large rise in outliers for load **R2**. This shows that noise increases feature data for the regression, lowering the median error, but it also increases margins of error, thus increasing the percentage of outliers.

Tests RIII and RIV show a decline in the disaggregation quality for the high-power load **R4**. While this disaggregation is still accurate, the global MSE for both clean and noise data increases by 4 to 5 orders of magnitude. However, both tests also show that high-power loads are more resistant to the impact of noise in the disaggregation process, with both clean and noise that for load **R4** having the same order of magnitude. These tests also prove consistent with the assertion that the introduction of noise has the effect of lowering the median error, while increasing the margins of error, as can be observed in figures 6.5 and 6.9.

The results of test RV show a decline in the quality of disaggregation for all the considered loads. While the median error is constant with the four previous tests, the MSE values for either global or individual loads are increased across the board by a few orders of magnitude. This is also the first test to present with a non-zero outlier percentage for any of the clean load data, further demonstrating the degradation of the disaggregation quality with the increase in data complexity. Another discernable factor is the significantly worse disaggregation of loads **R2** and **R5** in the clean batch of data. We can also observe in figure 6.10a that these two loads have a higher frequency of activation, which could impact disaggregation quality.

6.3.2 Inductive Type I Loads

In this subsection, we will be analyzing the following set of loads:

	RL95	RL90	RL60	RL80	RL85
Resistance	100 Ω	500 Ω	252 Ω	27 Ω	750 Ω
Inductance	104.62 mH	770 mH	803 mH	64.45 mH	1.48 H
Power Consumption	250.2 W	46.3 W	68.2 W	736.7 W	31.9 W

Table 6.7: Resistance and Inductance and power consumption of all 5 loads

As in the previous subsection, these loads were designed so that we could test the impact of high-power and low-power loads on the disaggregation quality. These loads were also designed to have different power factors, as indicated by their designation (RL95 has a power factor of 0.95), in order to infer if said power factor also impacts the performance of the *Random Forest*

Regression. These loads were also turned ON and OFF at an arbitrary frequency and duty-cycle in the same range as the resistive loads.

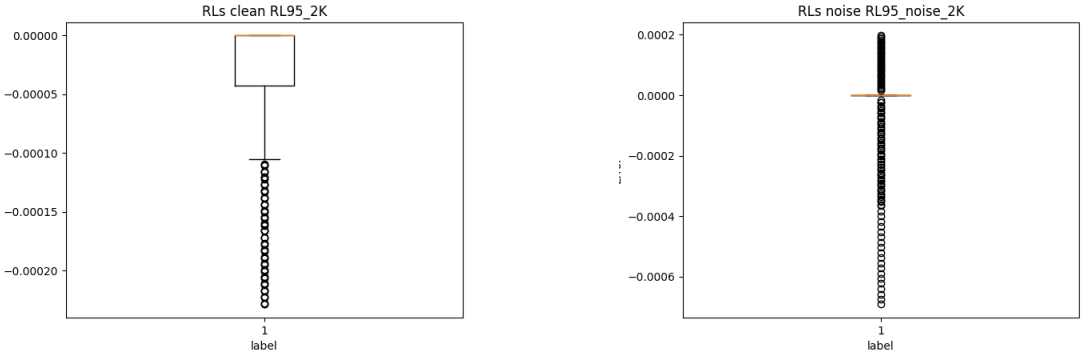
6.3.2.1 Test RLI: 1 Low power load

For this test, the only considered load is **RL95**, obtaining the following results:

	Clean Mean Square Error		
	5.48e-9		
Clean	MSE	Median	% of Outliers
RL95	5.48e-9	-1.31e-9	16.5

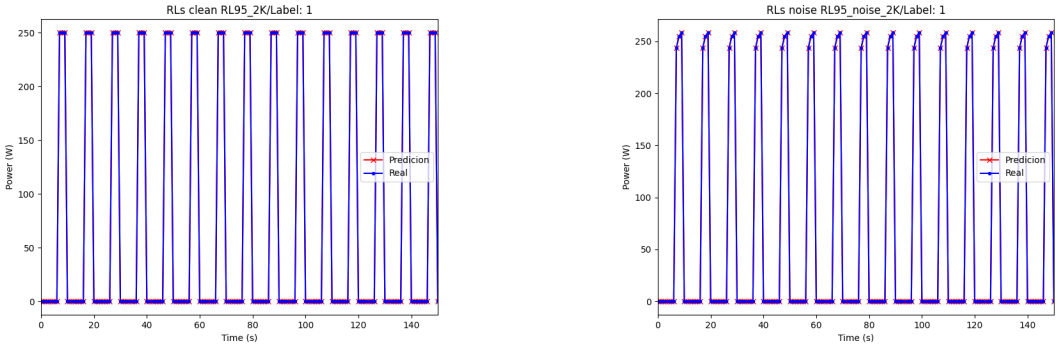
	Noise Mean Square Error		
	2.27e-8		
Noise	MSE	Median	% of Outliers
RL95	2.27e-8	-1.48e-11	30

Table 6.8: Performance metrics for Test RLI



(a) Clean Data. (b) Noise data.

Figure 6.11: Box Plot for the RL95 load in Test RLI



(a) Clean Data. (b) Noise data.

Figure 6.12: Behaviour plot for the RL95 load in Test RLI

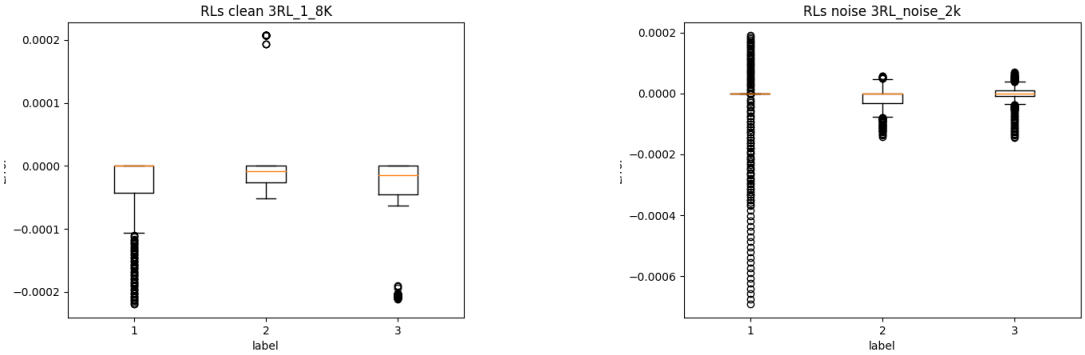
6.3.2.2 Test RLII: 3 Low power load

This test considers loads **RL95**, **RL90**, and **RL60**, obtaining the following results:

	Clean Mean Square Error		
	4.89e-9		
Clean	MSE	Median	% of Outliers
RL95	5.08e-9	-2.12e-22	15.79
RL90	4.66e-9	-8.65e-6	10
RL60	4.92e-9	-1.45e-5	10

	Noise Mean Square Error		
	8.57e-9		
Noise	MSE	Median	% of Outliers
RL95	2.27e-8	-5.29e-23	40
RL90	2.27e-8	-1.27e-21	12
RL60	2.27e-8	-8.47e-22	21

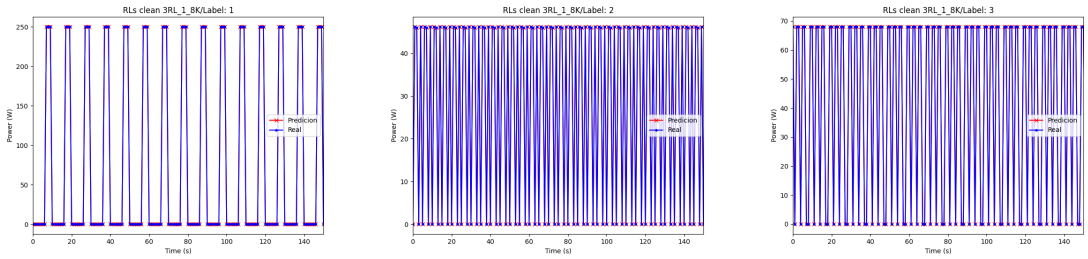
Table 6.9: Performance metrics for Test RLII



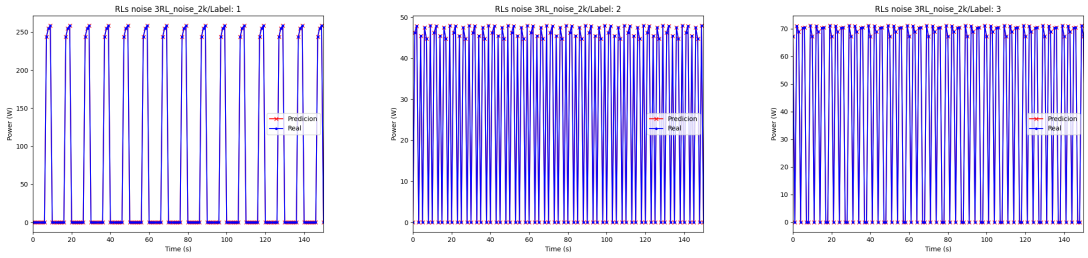
(a) Clean Data.

(b) Noise data.

Figure 6.13: Box Plot for the RL95, RL90, and RL60 loads in Test RLII



(a) Clean Data.



(b) Noise data.

Figure 6.14: Behaviour plot for the RL95, RL90, and RL60 load in Test RLII

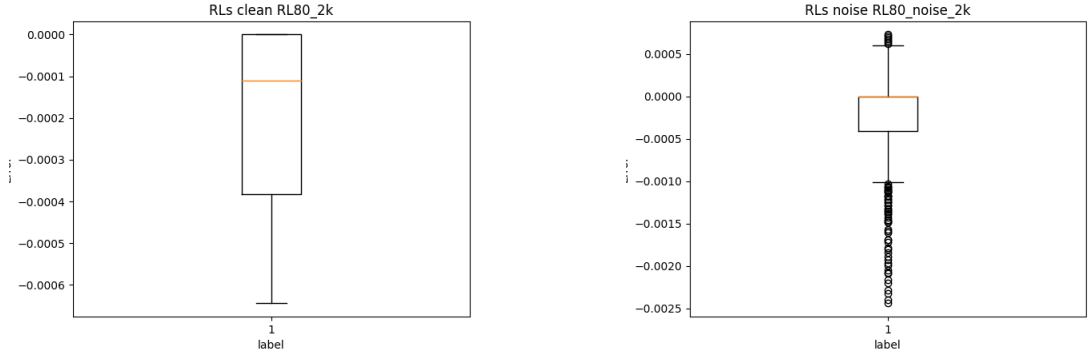
6.3.2.3 Test RLIII: 1 High power load

This test considers the only high power load among the inductive type one batch, **RL80**, obtaining the following results:

	Clean Mean Square Error		
	8.38e-8		
Clean	MSE	Median	% of Outliers
RL80	8.38e-8	-1.10e-4	0

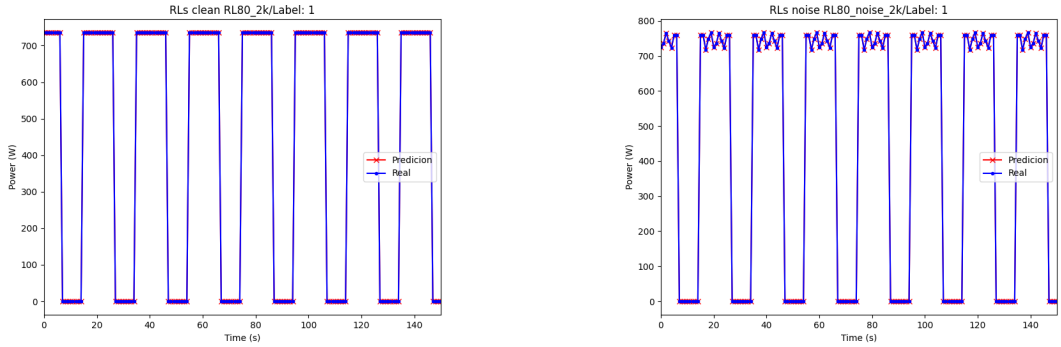
	Noise Mean Square Error		
	4.31e-7		
Noise	MSE	Median	% of Outliers
RL80	4.31e-7	-1.38e-8	14.5

Table 6.10: Performance metrics for Test RLIII



(a) Clean Data. (b) Noise data.

Figure 6.15: Box Plot for the R4 load in Test RLIII



(a) Clean Data. (b) Noise data.

Figure 6.16: Behaviour plot for the RL80 load in Test RLIII

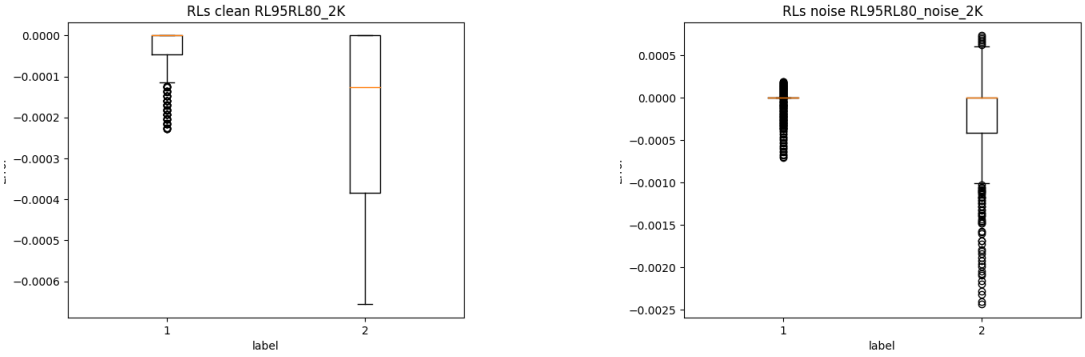
6.3.2.4 Test RLIV: 1 High-power and 1 Low-power load

In this test, we consider loads **RL95** and **RL80** obtaining the following results:

	Clean Mean Square Error		
	4.69e-8		
Clean	MSE	Median	% of Outliers
RL95	5.55e-9	-1.18e-9	15
RL80	8.84e-8	-1.26e-4	0

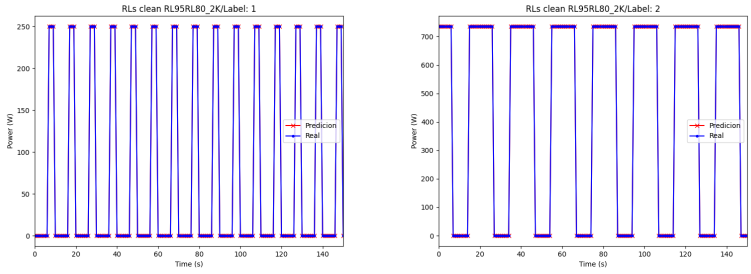
	Noise Mean Square Error		
	2.28e-7		
Noise	MSE	Median	% of Outliers
RL95	2.45e-8	0	40
RL80	4.31e-7	-1.17e-10	14.5

Table 6.11: Performance metrics for Test RLIV

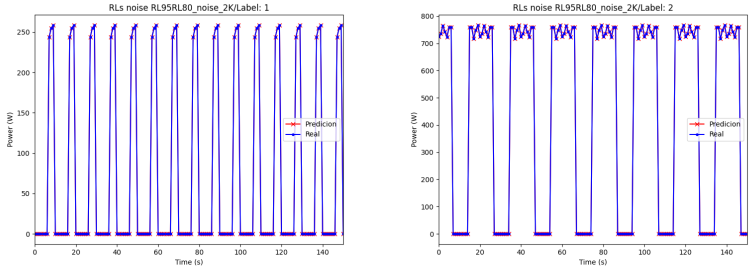


(a) Clean Data. (b) Noise data.

Figure 6.17: Box Plot for the RL95, and RL80 loads in Test RLIV



(a) Clean Data.



(b) Noise data.

Figure 6.18: Behaviour plot for the RL95, and RL80 load in Test RLIV

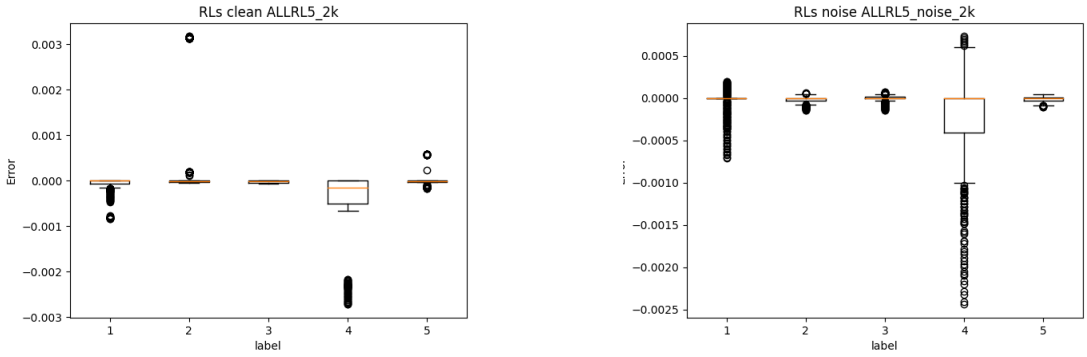
6.3.2.5 Test RLV: 1 High-power and 4 Low-power load

This test considers all loads presented in Table B.2, obtaining the following results:

	Clean Mean Square Error		
	4.58e-7		
Clean	MSE	Median	% of Outliers
RL95	4.17e-8	-1.06e-22	16.5
RL90	1.37e-6	-3.30e-6	23.75
RL60	1.04e-9	-1.46e-5	0
RL80	8.60e-7	-1.47e-4	13.75
RL85	1.99e-8	-1.16e-5	24

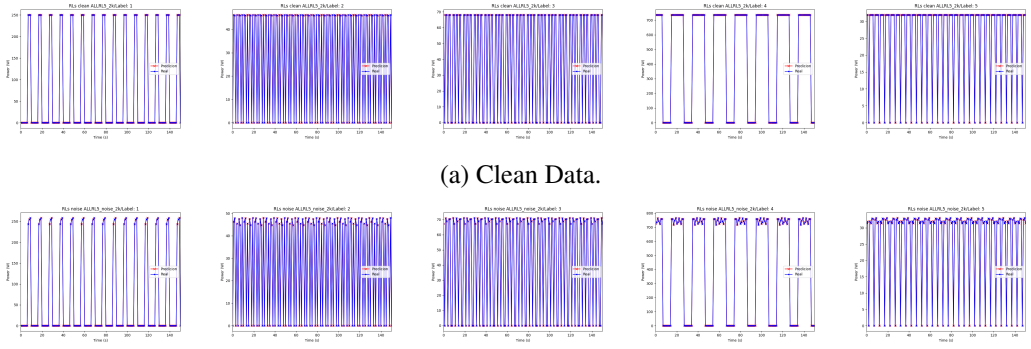
	Noise Mean Square Error		
	9.20e-8		
Noise	MSE	Median	% of Outliers
RL95	2.46e-8	0	30
RL90	1.78e-9	-4.24e-22	11.25
RL60	1.22e-9	0	19.75
RL80	4.31e-7	-5.29e-23	14.5
RL85	1.18e-9	-9.43e-8	2.25

Table 6.12: Performance metrics for Test RLV



(a) Clean Data. (b) Noise data.

Figure 6.19: Box Plot for the RL95, RL90, RL60, RL80, and RL85 loads in Test RLV



(a) Clean Data.

(b) Noise data.

Figure 6.20: Behaviour plot for the RL95, RL90, RL60, RL80, and RL85 loads in Test RLV

6.3.2.6 Conclusions

Despite the worse disaggregation quality when compared to test RI, test RLI also displays a very low MSE and median error for both clean and noise data. However, this test also displays a moderate to high percentage of outliers in the clean and noise data respectively. However, this is likely due to the high frequency and low duty-cycle used when powering load **RL95**, as can be observed in Figure 6.12.

Test RLII displays results consistent with test RLI and RII, with the addition of 2 other low-power loads not significantly affecting the quality of disaggregation, with the Median error for both clean and noise data decreasing drastically. The percentage of outliers for the noise data **RL95** load is also significantly increased, however, upon analyzing figure Fig, we can observe that the range of error values does not significantly increase. Tests RLIII and RLIV shows peculiar behavior. While their results are consistent with tests RIII and RIV, load **RL80** is the first inductive type I load to present with 0% outliers on clean data. This could be due to its lower frequency of activation and higher duty-cycle, as can be observed in Figure 6.16. Despite this, Test RLIV also displays that the capability of the regression to disaggregate low-power loads is not significantly affected by the introduction of high-power loads.

Test RLV presents the most unexpected results, in which while most of its performance metrics are consistent with test RV, with a slight degradation of the disaggregation quality in both clean and noise samples due to data complexity, load RL60 shows a much better performance when compared to test RLIII. This is likely due to its power factor being the most disparate when compared to the others, making its impact the most distinguishable in the feature data among all loads.

Overall, these tests show that despite there being a reduction in disaggregation quality, with a systematic increase in MSE values and percentage of outliers, inductive type I loads are still highly accurately disaggregated with *Scikit-learn's Random Forest Regression*.

6.3.3 Resistive Type II Loads

In this subsection, we will be analyzing the following set of loads:

	VR1	VR2	VR3
Resistance	2000 Ω	1000 Ω	30 Ω
Max Power Consumption	13.7 W Ω	27.3 W Ω	939.5 W Ω

Table 6.13: Resistance and power consumption of the 3 loads loads

These resistive values represent the maximum consumed power by each respective load. As can be seen in Annex B, 5 square waves with the same amplitude but different arbitrary frequencies and duty-cycles, create different possible states for the loads by increasing the resistance of the load and lowering the power consumption. These loads also have an ON/OFF switch, adding an additional state. As with previous tests, these loads are also designed in order to be high and low-power loads, however, as previously explained, due to time constraints there are only three

loads. These three loads are also turned on and off by a square wave with arbitrary frequency and duty-cycle in the same range as the previous loads.

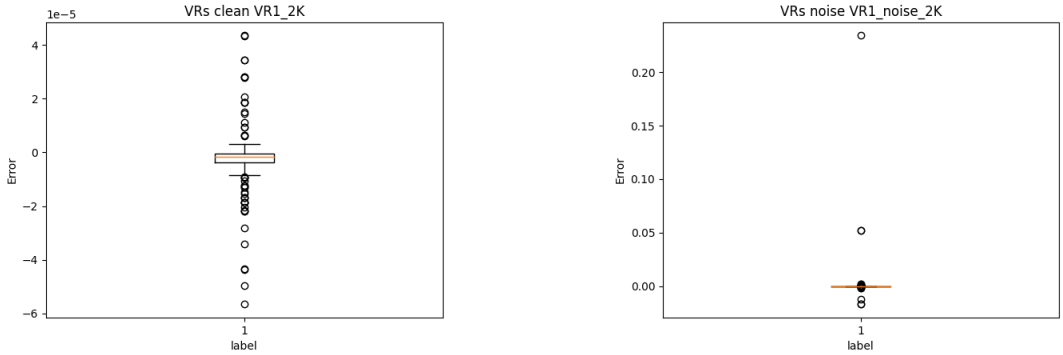
6.3.3.1 Test VRI: 1 Low power load

For this test, the only considered load is **VR1**, obtaining the following results:

	Clean Mean Square Error		
	7.72e-11		
Clean	MSE	Median	% of Outliers
VR1	7.72e-11	-1.80e-6	11.75

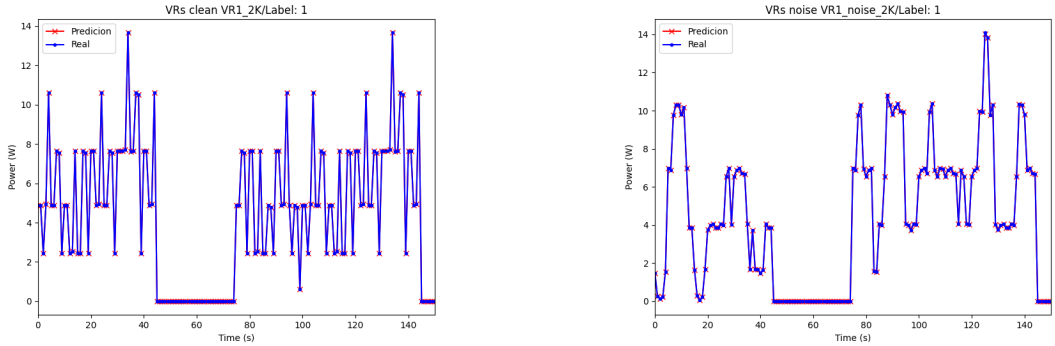
	Noise Mean Square Error		
	1.53e-4		
Noise	MSE	Median	% of Outliers
VR1	1.53e-4	6.63e-9	17.25

Table 6.14: Performance metrics for Test VRI



(a) Clean Data. (b) Noise data.

Figure 6.21: Box Plot for the VR1 load in Test VRI



(a) Clean Data. (b) Noise data.

Figure 6.22: Behaviour plot for the VR1 load in Test VRI

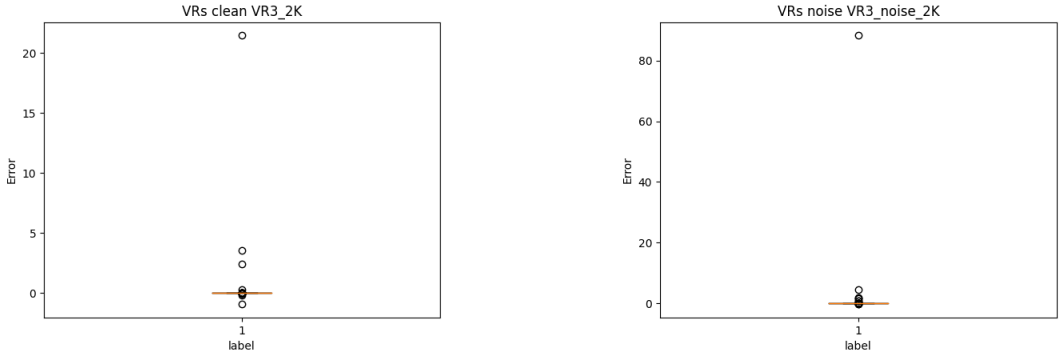
6.3.3.2 Test VRII: 1 High power load

For this test, the only considered load is **VR3**, obtaining the following results:

	Clean Mean Square Error		
	1.20		
Clean	MSE	Median	% of Outliers
VR3	1.20	-1.31e-4	6.5

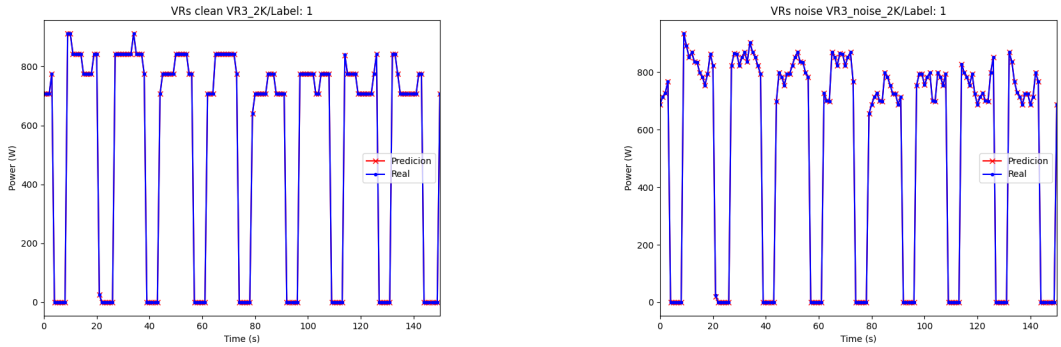
	Noise Mean Square Error		
	19.57		
Noise	MSE	Median	% of Outliers
VR3	19.57	-2.10e-6	12.75

Table 6.15: Performance metrics for Test VRII



(a) Clean Data. (b) Noise data.

Figure 6.23: Box Plot for the VR3 load in Test VRII



(a) Clean Data. (b) Noise data.

Figure 6.24: Behaviour plot for the VR3 load in Test VRII

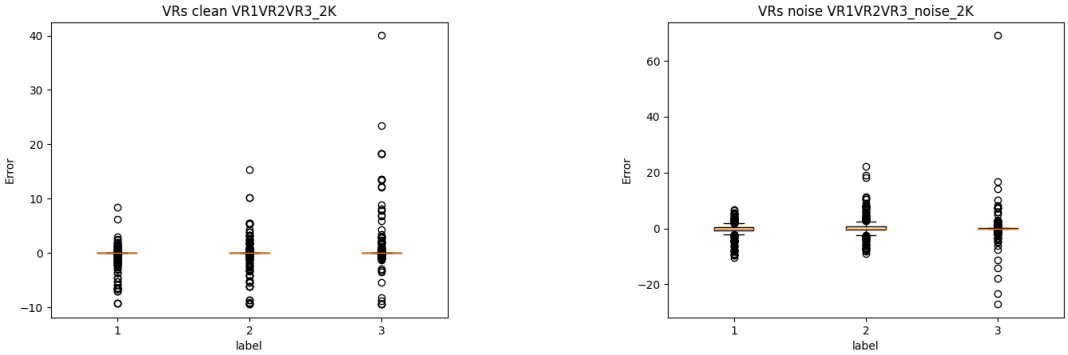
6.3.3.3 Test VRIII: 1 High-power and 2 Low-power load

This test considers all loads as presented in table B.3, obtaining the following results:

	Clean Mean Square Error		
	7.09		
Clean	MSE	Median	% of Outliers
VR1	2.91	-3.98e-6	43.25
VR2	4.28	-6.48e-6	39.75
VR3	14.08	-1.34e-4	35

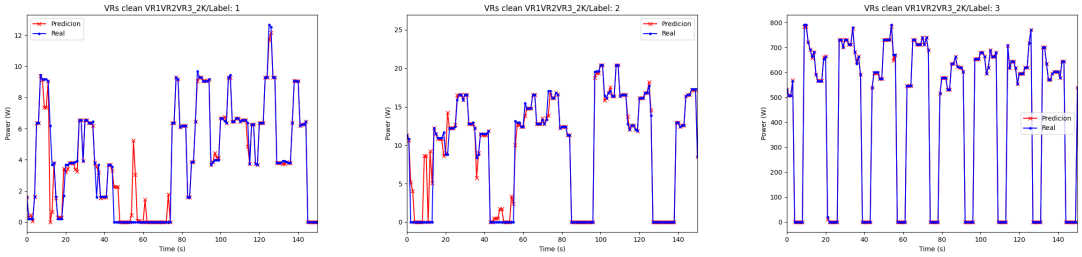
	Noise Mean Square Error		
	12.14		
Noise	MSE	Median	% of Outliers
VR1	5.47	-4.94e-5	21
VR2	10.65	2.64e-6	20.25
VR3	20.30	-3.70e-5	44

Table 6.16: Performance metrics for Test VRIII

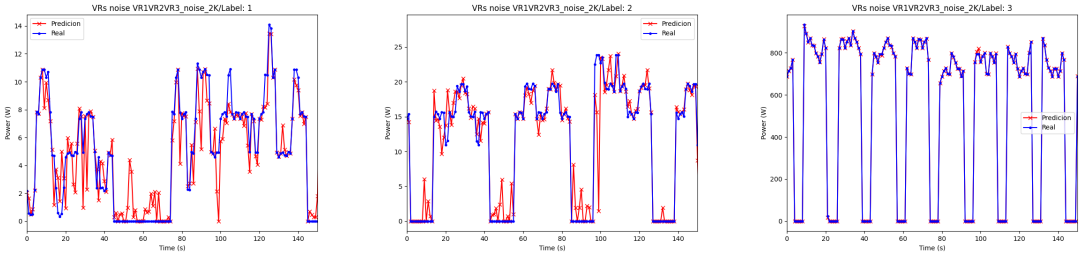


(a) Clean Data. (b) Noise data.

Figure 6.25: Box Plot for the VR1, VR2, and VR3 loads in Test VRIII



(a) Clean Data.



(b) Noise data.

Figure 6.26: Behaviour plot for the VR1, VR2, and VR3 loads in Test VRIII

6.3.3.4 Conclusions

Test VRI displays a baseline for the behavior of type II loads. This test presents a high disaggregation quality for the clean data, with **VR1** displaying very low MSE and median error, and despite having 11.75% outliers, we can observe in Figure 6.21a that most of these outliers are concentrated near the whiskers of the box plot. When looking at the noise data for this test, we can observe that while the disaggregation quality is still very high, and the median error and percentage of outliers are consistent with previous tests, there is an increase in 7 orders of magnitude in the MSE value. This is likely due to the muddling of the individual states due to noise. Figures 6.22a and 6.22b show the normally discrete states for **VR1** become less distinguishable with the introduction of noise, leading to an increase in error.

In Test VRII, we observe a very significant increase in MSE for both clean and noise data, reaching non-fractional values. While the median error and percentage of outliers remain relatively low, this sharp increase in MSE is consistent with the conclusions made about tests RIV, RLIV, and VRI, with the multiple states and high power consumption degrading the disaggregation quality. Despite all this, we can still observe a good disaggregation quality, since an MSE of 1.20 and 19.57 for a load with up 850 W (in clean data) consumed is still low.

Test VRIII is the first test with poor disaggregation quality. While the results for **VR3** are consistent, with MSE only significantly increased for the clean data, the disaggregation is still accurate, loads **VR1** and **VR2** show a sharp increase in MSE. While the median error for these loads is low in both clean and noisy data, the outliers are also significantly higher for the clean data. The MSE for these two loads is at a point where the disaggregation quality is poor, as can be observed in both figures 6.26a and 6.26b.

This batch of tests shows that the disaggregation quality highly degrades with data complexity that comes with loads with multiple operating states, and in order to get a more precise identification of power consumption, more feature data would be required.

6.3.4 Inductive Type II Loads

In this subsection, we will be analyzing the following set of loads:

	VRL84	VRL92	VRL88
Resistance	1500 Ω	1500 Ω	23 Ω
Inductance	3.084 H	2.034 H	39.5 mH
Max Power Consumption	18.3 W Ω	18.3 W Ω	1.24 kW Ω

Table 6.17: Resistance, inductance, and maximum power consumption of the 3 loads

As with the previous batch of tests, the presented values for resistance represent the maximum power consumption for each respective load. This also means that the power factor values presented on the label for each load in table B.4 are also only correct for the maximum consumption of power. The resistance values for these loads are also changed by 5 square waves with equal

amplitude but arbitrary frequency and duty-cycle in order to create different operational states, along with the additional square with the same characteristics that turn the load ON and OFF.

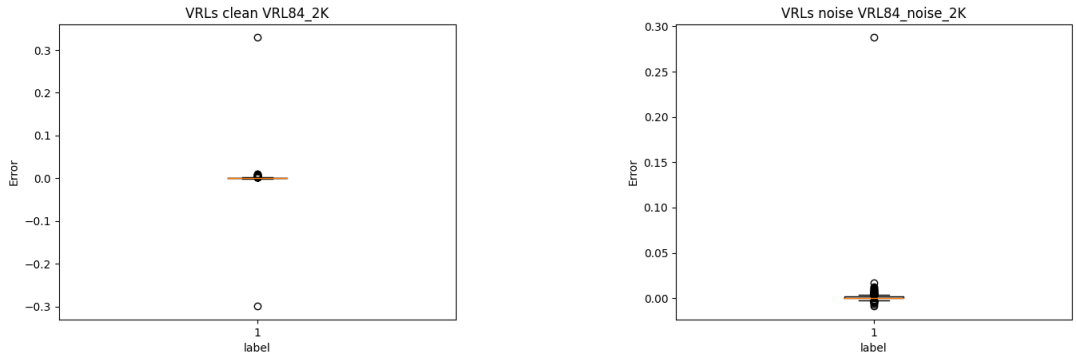
6.3.4.1 Test VRLI: 1 Low power load

For this test, the only considered load is **VRL84**, obtaining the following results:

	Clean Mean Square Error		
	4.97e-4		
Clean	MSE	Median	% of Outliers
VRL84	4.97e-4	2.50e-6	7.25

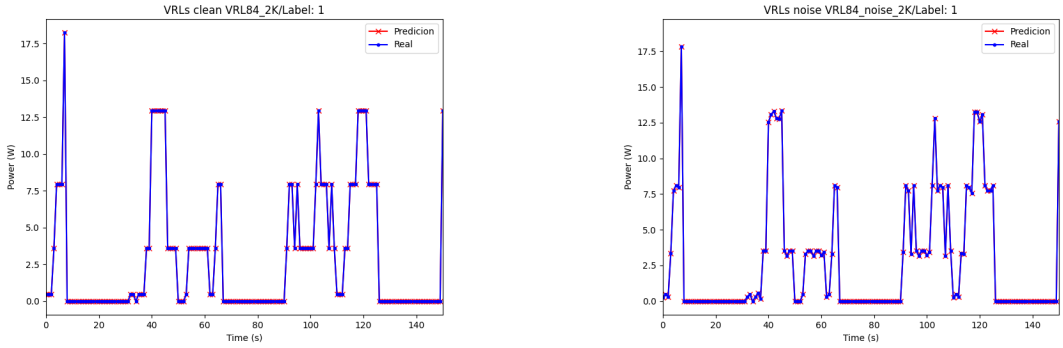
	Noise Mean Square Error		
	2.13e-4		
Noise	MSE	Median	% of Outliers
VRL84	2.13e-4	7.36e-6	13.5

Table 6.18: Performance metrics for Test VRLI



(a) Clean Data. (b) Noise data.

Figure 6.27: Box Plot for the VRL84 load in Test VRLI



(a) Clean Data. (b) Noise data.

Figure 6.28: Behaviour plot for the VRL84 load in Test VRLI

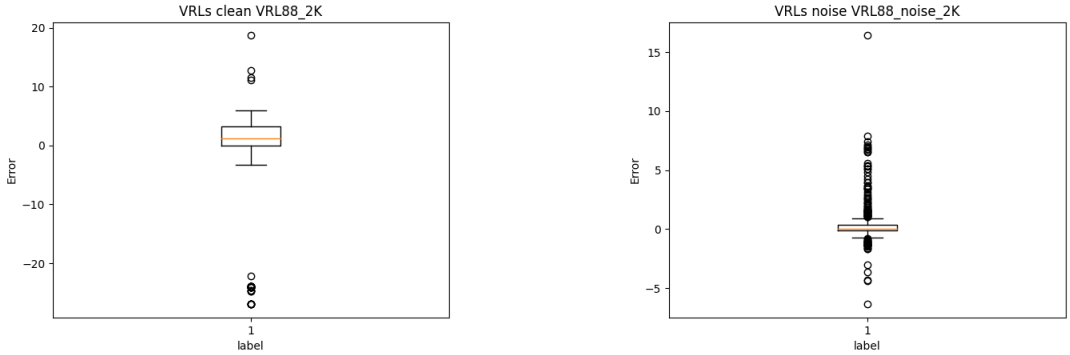
6.3.4.2 Test VRLII: 1 High power load

For this test, the only considered load is **VRL88**, obtaining the following results:

	Clean Mean Square Error		
	69.11		
Clean	MSE	Median	% of Outliers
VRL88	69.11	1.19	11.25

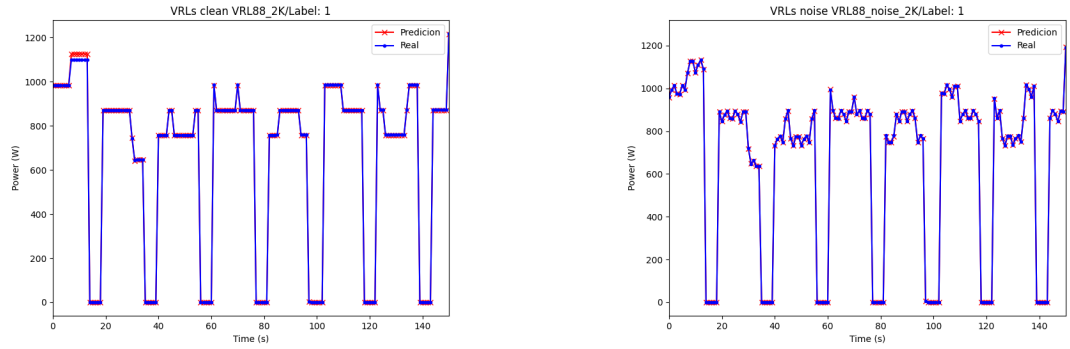
	Noise Mean Square Error		
	3.47		
Noise	MSE	Median	% of Outliers
VRL88	3.47	2.09e-5	24.75

Table 6.19: Performance metrics for Test VRLII



(a) Clean Data. (b) Noise data.

Figure 6.29: Box Plot for the VRL88 load in Test VRII



(a) Clean Data. (b) Noise data.

Figure 6.30: Behaviour plot for the VRL88 load in Test VRLII

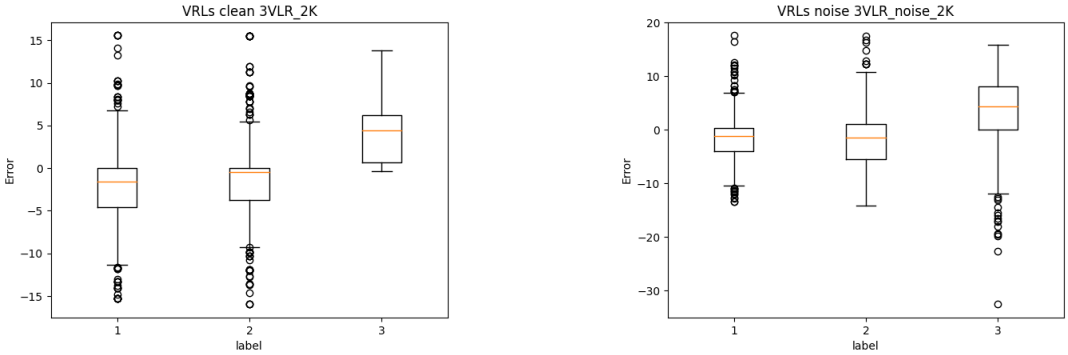
6.3.4.3 Test VRLIII: 1 High-power and 2 Low-power load

This test considers all loads in table B.4, obtaining the following results

	Clean Mean Square Error		
	31.01		
Clean	MSE	Median	% of Outliers
VRL84	33.24	-1.61	12.5
VRL92	31.81	-0.44	17.5
VRL88	27.99	4.43	0.0

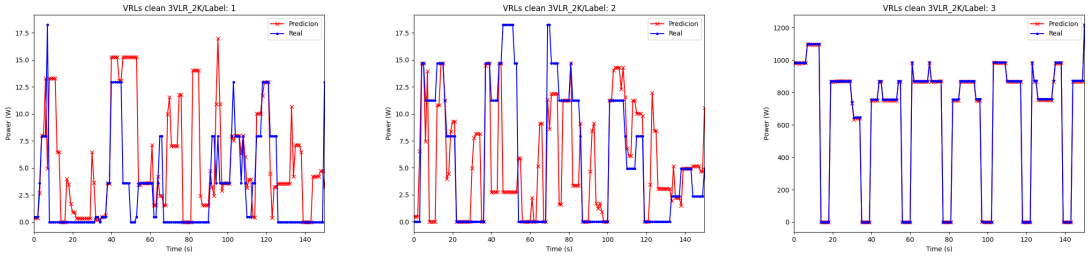
	Noise Mean Square Error		
	38.03		
Noise	MSE	Median	% of Outliers
VRL84	24.08	-1.10	8
VRL92	33.96	-1.38	2.25
VRL88	56.05	4.29	3.75

Table 6.20: Performance metrics for Test VRLIII

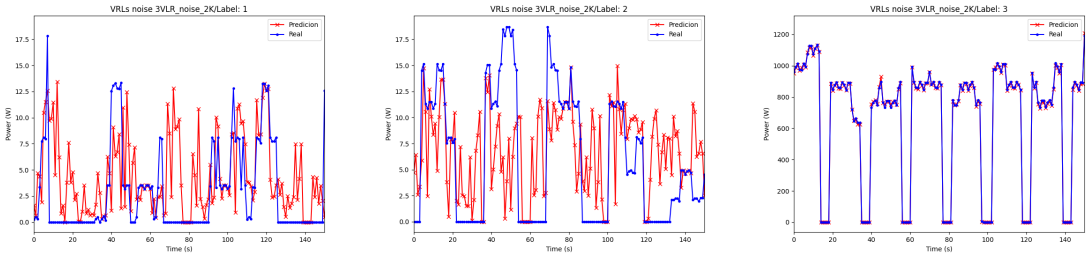


(a) Clean Data. (b) Noise data.

Figure 6.31: Box Plot for the VRL84, VRL92, and VRL88 loads in Test VRLIII



(a) Clean Data.



(b) Noise data.

Figure 6.32: Behaviour plot for the VRL84, VRL92, and VRL88 loads in Test VRLIII

6.3.4.4 Conclusions

Test VRLI presents uncharacteristic performance metrics for the baseline test for a batch of loads. While the disaggregation is still high quality, this is the first baseline test to present with a higher MSE for the clean data than the noise one. This can also be observed in figures 6.27a and 6.27b, where the outliers for the clean data spread across a wider range than the ones in the noise data. In accordance with the resistive type II loads, this baseline test also displays outliers when disaggregating a single load, reinforcing the idea that data complexity is the bottleneck for this specific regression.

Test VRLII maintains the trend of this batch of having higher MSE for the clean data than the noise data, however, for this test the median error is also significantly higher, reaching a non-fractional value. This supports the idea that the introduction of noise creates additional feature data, via the fluctuations in voltage, current, etc., leading to a lower median error, but widening the range for outliers, thus creating the conditions observed in table 6.19. When observing figures 6.28b and 6.28a, we can observe that no error is discernible in the noise data, and while it is discernible in the clean data, it is not so egregious that the disaggregation quality could be considered poor.

Test VRLIII has significantly lower disaggregation quality. As seen in table 6.16,**VRL84** and **VRL92**, these low-power loads have significant MSE values. Figure The median error for these loads is likewise considerable, causing the behavior in figures 6.32a and 6.32b. Figure 6.31 also shows that these loads have a moderate percentage of outliers despite having a high interquartile range, where 50% of the most common data is located. The disaggregation quality for the noise data of the **VRL88** load decreases as data complexity increases, but the clean data quality increases compared to test VRLII, likely due to this load being the most discernible in the batch.

Overall, these three loads present a worse disaggregation quality than their purely resistive counterparts, this is likely due to the changes in power factor with the changes in resistance of the load, effectively muddling this feature and creating further data complexity.

6.3.5 All Loads

In this subsection, we will be analyzing a simulation in which all previously presented loads are connected to the power source. All previous statements made about these loads are still applicable in this test. For the purposes of this test, table 6.21 presents what labels correspond to each load.

Label	1	2	3	4	5	6	7	8
Load	R1	R2	R3	R4	R5	RL95	RL90	RL60

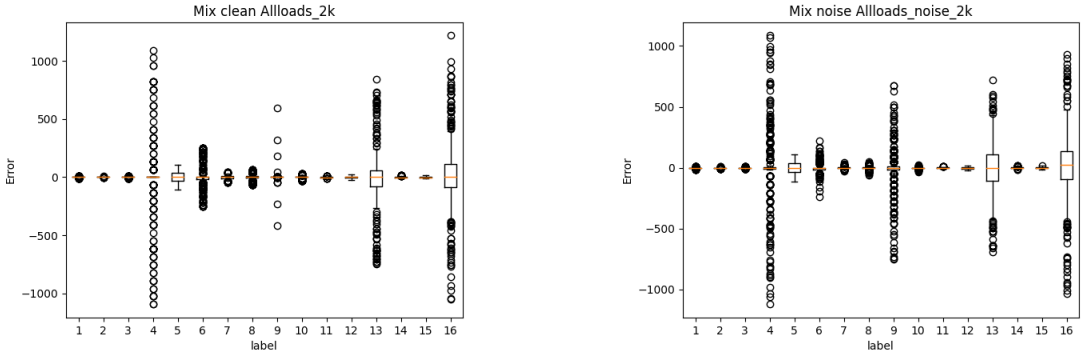
Label	9	10	11	12	13	14	15	16
Load	RL80	RL85	VR1	VR2	VR3	VRL84	VRL92	VRL88

Table 6.21: Behaviour plot for the VRL84, VRL92, and VRL88 loads in Test VRLIII

	Clean Mean Square Error		
	1.62e4		
clean	MSE	Median	% of Outliers
R1	19.4	-2.03e-5	42.25
R2	0.27	-1.01e-5	2.25
R3	15.0	-1.67e-5	40
R4	8.71e4	-1.38e-3	31.5
R5	2.90e3	1.58e-4	0.0
RL95	8.62e3	-3.75e-9	36.25
RL90	2.90e2	1.39e-3	7.75
RL60	6.31e2	2.28e-8	22.5
RL80	1.82e3	-1.94e-9	24
RL85	1.52e2	3.01e-3	24.75
VR1	10.4	-0.304	4.5
VR2	88.2	0.385	0.0
VR3	6.06e4	-1.66e-4	17
VRL84	27.0	-1.17	3.5
VRL92	41.0	-0.496	0.0
VRL88	9.70e4	4.48	18.25

	Noise Mean Square Error		
	1.65e4		
noise	MSE	Median	% of Outliers
R1	19.6	-1.37e-8	38.25
R2	5.16	4.23e-22	48
R3	13.1	2.05e-5	34.75
R4	8.60e4	-3.43e-11	42.5
R5	2.81e3	5.80e-6	0.0
RL95	1.56e3	-4.22e-9	19.5
RL90	52.2	3.04e-8	10
RL60	1.70e2	3.84e-8	15
RL80	3.39e4	3.12e-10	46
RL85	20.8	1.05e-7	31.75
VR1	10.5	-0.208	2
VR2	81.9	0.731	0
VR3	5.00e4	6.29e-5	7.5
VRL84	26.6	-1.29	7
VRL92	41.4	-0.674	0.25
VRL88	8.96e4	22.5	11.5

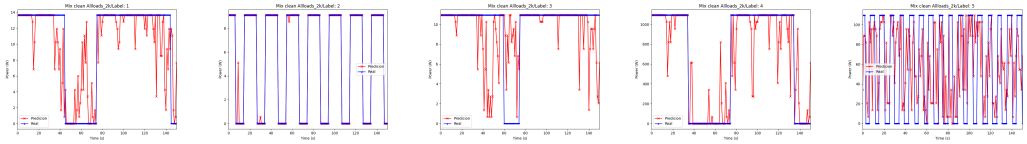
Table 6.22: Performance metrics for All loads Test



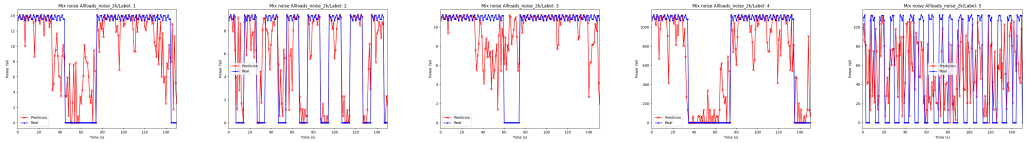
(a) Clean Data.

(b) Noise data.

Figure 6.33: Box Plot for the All loads Test

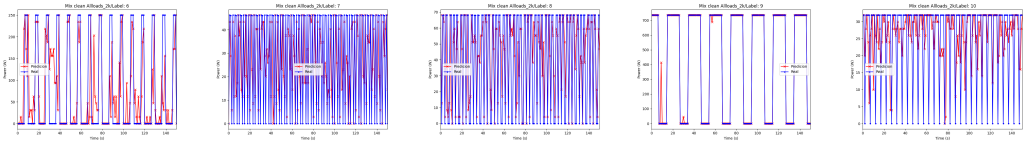


(a) Clean Data.

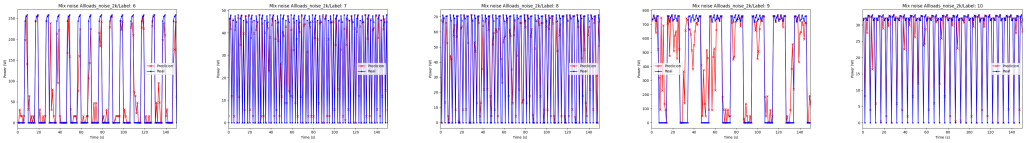


(b) Noise data.

Figure 6.34: Behaviour plot for the R1, R2, R3, R4, and R5 loads in All loads Test

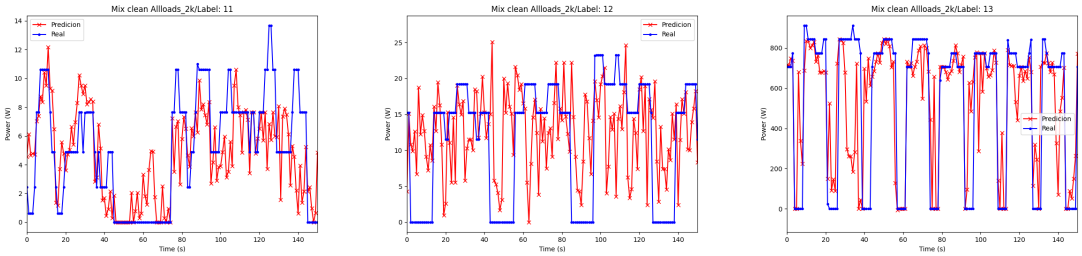


(a) Clean Data.

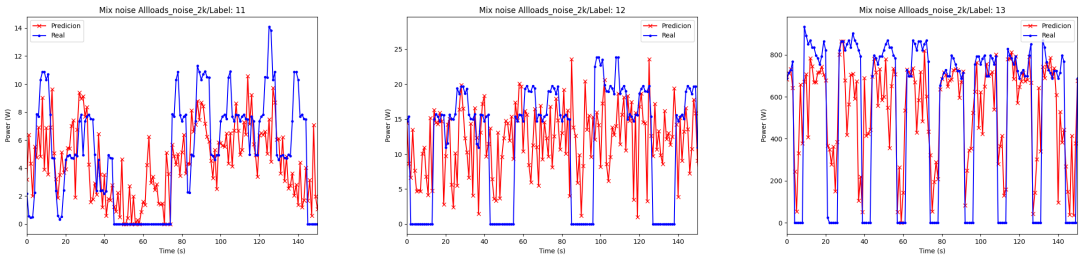


(b) Noise data.

Figure 6.35: Behaviour plot for the RL95, RL90, RL60, RL80, and RL85 loads in All loads Test

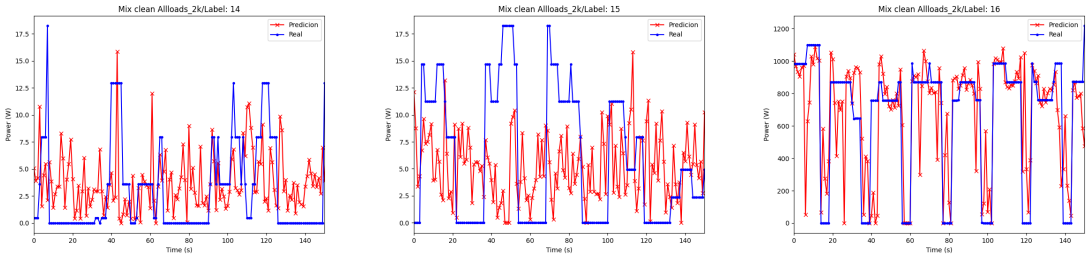


(a) Clean Data.

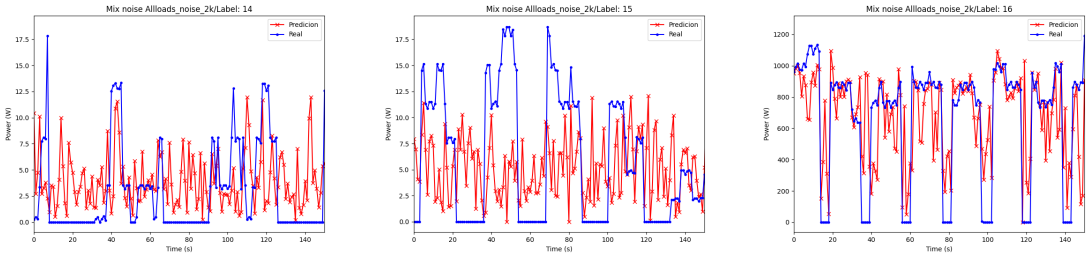


(b) Noise data.

Figure 6.36: Behaviour plot for the VR1, VR2, and VR3 loads in All loads Test



(a) Clean Data.



(b) Noise data.

Figure 6.37: Behaviour plot for the VRL84, VRL92, and VRL88 loads in All loads Test

6.3.5.1 Conclusions

There is a wide range of behavior observed among the loads **R1**, **R2**, **R3**, **R4**, and **R5** with regard to the clean and noise MSE and the percentage of outliers. R1 and R3 both have MSE values that are relatively high, which is consistent with the notion that data complexity worsens disaggregation. However, **R2** shows an MSE that is noticeably lower. When comparing these loads' behavior plots with the data from test RV, we observe that **R2** has a comparable behavior, with only the error values increasing, as can be predicted by the MSE. R4 has an extraordinarily high clean and noise MSE, which indicates that there may be difficulties in effectively recording the pattern of power consumption in high power loads. R5 is particularly notable for having a high MSE despite also being a low-power load. when looking at Figure 6.33, we can see that the interquartile range is also significantly higher than in other low-power resistive loads, thus justifying the percentage of outliers for this load. As stated before, this worse behavior appears to stem from the frequency of activation of this load.

Loads **RL95**, **RL90**, **RL60**, **RL80**, and **RL85** exhibit worse behavior than their resistive counterparts, particularly with regard to their MSE and outlier percentages. All of the low-power loads from this set display an egregious MSE for the clean data, and a more reasonable, but still high MSE for the noise one. This discrepancy between sets of loads supports the idea that a high frequency and low duty-cycle negatively impacts the disaggregation performance. Load **RL85** further supports this idea, since, as can be observed in Figure 6.35, it is the low-power load with the highest duty-cycle and also the lowest MSE and percentage of outliers for all RL loads. The

results observed in the noise data for these five loads in table 6.22 also support the idea that the introduction of noise to the power signal can improve the disaggregation process due to additional feature data, lowering on average the median error value, but it also on average increases the percentage of outliers and the range of error for a specific load.

The loads **VR1**, **VR2**, and **VR3** each display poor disaggregation quality, which can be observed in figures 6.36a and 6.36b. Both **VR1** and **VR2** have clean MSE values that are quite low when compared to other loads in this test, however, considering the low power consumption of these loads this MSE is still relatively high. **VR3** also presents with an MSE 3 to 4 orders of magnitude higher than the other resistive type II loads, further cementing the difficulty of disaggregating high power loads when in the presence of high data complexity. When looking at the noise data for this test present in table 6.22, when can observe a rather uncharacteristic behavior, considering all performance metrics, excluding the median error for load **VR2**, have improved from their clean counterparts, however, when looking at the box plot in figure 6.22, we can observe that the range of error values is similar to the one for the clean data, with only the interquartile range increasing, meaning that the 50% of the most likely error values are more spread out, resulting in a lower percentage of outliers.

Loads **VRL84**, **VRL92**, and **VRL88** display similar while slightly worse disaggregation performance, as can be observed in their behavior plot in figures 6.37a and 6.37b. Similarly to loads **VR1** and **VR2**, the low power loads **VRL84** and **VRL92** present with a low MSE when compared to the low power type I inductive loads, but considering their low power consumption, these loads, along with **VRL88**, present a high MSE across the board. **VRL84** and **VRL88** are also the only two loads to present non-fractional median error for this test, which is consistent with the findings in test VRLIII, where the data complexity and feature muddling due to overlapping power and power factor values between loads highly degrades the quality of the disaggregation.

6.3.6 ECO Dataset Test

During this last test, we will evaluate the quality of the disaggregation process in a real-world setting by using *Scikit-learn's Random Forest Regression* object that was trained before. We will then compare this evaluation to the conclusions that were gathered using the simulated data. It was important to use consecutive data and to display such data in a scatter plot in order for the behavior graphs to be interpretable. Additionally, it was necessary to use consecutive data. The following is what the research found:

Mean Square Error
66.38

	MSE	Median	% of Outliers
Tablet	1.32	-0.164	0.15
Dishwasher	38.0	-1.46e-2	11.07
Air Exhaurst	6.32	-6.35e-3	14.53
Fridge	2.78e2	-0.388	12.17
Entertainment	19.7	-0.417	4.01
Freezer	2.22e2	-6.44e-2	27.43

	MSE	Median	% of Outliers
Kettle	46.9	-1.47e-5	7.03
Lamp	80.1	3.32e-2	28.16
Laptops	34.3	1.36e-2	32.24
Stove	54.7	0.0	10.54
TV	10.5	0.0	25.56
Stereo	3.63	-0.417	6.33

Table 6.23: Preformace Metrics for the ECO dataset Test

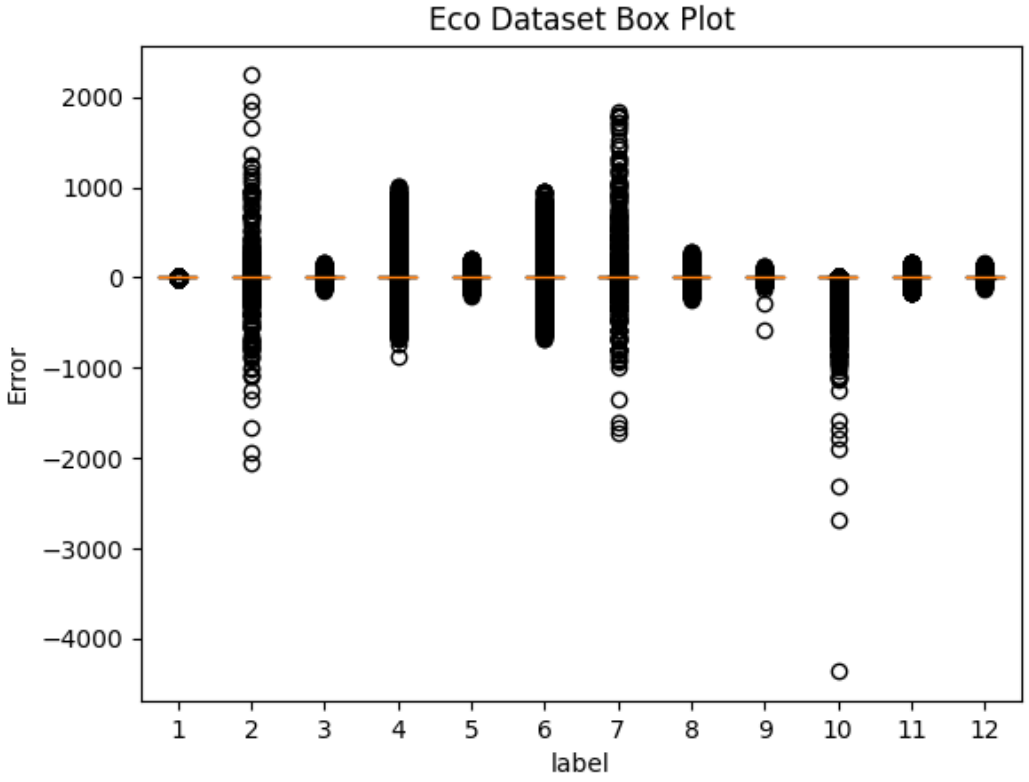
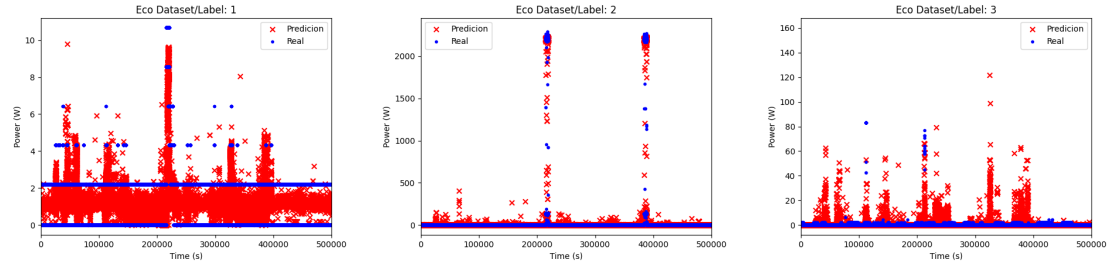
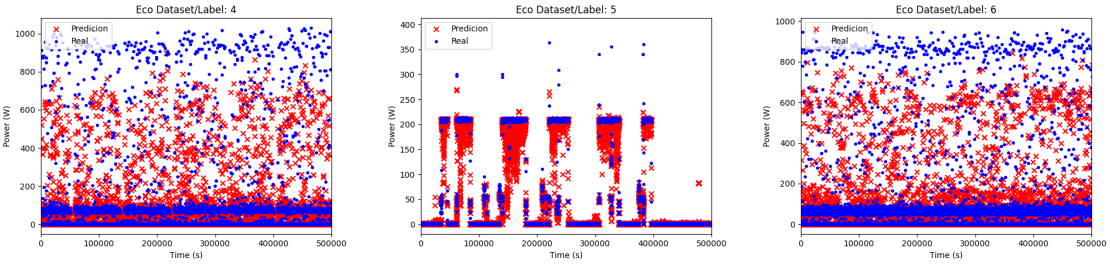


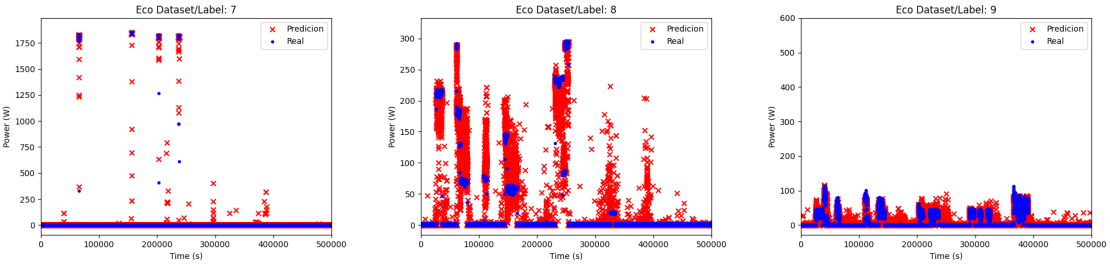
Figure 6.38: Box plot for the ECO dataset Test



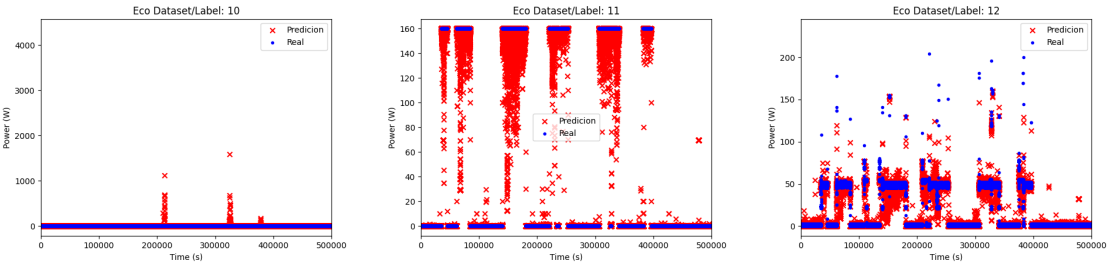
(a) Labels 1, 2, and 3: Tablet, Dishwasher, and Air Exhaust



(b) Labels 4, 5, and 6: Fridge, Entertainment, and Freezer



(c) Labels 7, 8, and 9: Kettle, Lamp, and Laptops



(d) Labels 10, 11, and 12: Stove, TV, and Stereo

Figure 6.39: Behaviour plot for All ECO dataset loads

6.3.6.1 Conclusions

When compared to the test containing all simulated loads, the ECO dataset presents with a much lower global MSE, indicating an improvement in performance disaggregation.

The tablet presents as a type II load with a relatively low MSE, median error, and percentage of outliers, however, looking at its behavior scatter plot in Figure 6.39a, we can observe that the error in disaggregation is enough to make the disaggregation quality worse than desirable.

By looking at the behavior plot of the dishwasher in Figure 6.39a, we can presume that it is a high-power type I load with low frequency and low duty-cycle. As such, this load presents a moderate percentage of outliers and a large error distribution, from -2000 to 2000. However, the MSE and median error for this load shows that most predictions are accurate, leading to a moderately accurate disaggregation process.

Similarly to the dishwasher, the air exhaust also behaves as a type I load with an even lower duty-cycle, as such its disaggregation quality is less than desirable, as can be observable in Figure 6.39a and by relatively moderate MSE with a relatively high percentage of outliers.

The fridge and freezer have similarly abnormal power consumption behavior and as such cannot be compared to any of the simulated loads, however, as can be observed in Figure 6.39b and in the loads' MSE and percentage of outliers, the disaggregation of these loads are rather poor, as can be expected by their complex behavior.

As stated in the previous chapter, the entertainment label contains two type I loads, the TV and the Stereo. When looking at these two loads, despite the unimpressive-looking behavior plots, presented in Figure 6.39d, it is important to remember that these cover a much wider timespan than the simulated tests. The performance metrics in for these two loads in table 6.23 support the idea of a quality disaggregation for these loads, with the relatively low MSE and median error. the TV has a high outlier percentage, but by analyzing the box plot in Figure 6.38, we can observe that even these less likely error values are still condensed near the median error. Looking at the entertainment label, we can see that it behaves similarly, if not slightly worse than the two loads that compose it. This is consistent with the idea that increased data complexity degrades the quality of the prediction.

The kettle presents an unexpected power behavior, as can be observed in figure 6.39c, since despite being comparable to a type I load with a low duty-cycle, its performance metrics and behavior plot indicate a high-quality disaggregation, with a relatively low MSE, low median error, and moderate percentage of outliers. This indicates that his load has a significant impact on the feature data, making it more easily discernable.

When looking at the behavior plot for the lamp in Figure 6.39c, we can observe that it is a type two load with noise and, as such, a poor disaggregation quality. The performance metrics in Table 6.23 further support this idea, with the high MSE when compared to the relatively low power consumption of this load and a high percentage of outliers. This load's median error is also low, but by looking at its boxplot in Figure 6.39c, we can see that the error values are spread across a long-range, thus cementing the poor disaggregation quality.

The laptops also behave in a way that is disparate from any simulated load, as can be seen in figure 6.39c, This load shows a slightly poor disaggregation quality, with a relatively high MSE for the loads' power consumption and a high percentage of outliers.

The behavior plot for the stove, present in Figure 6.39d, does not present any state where its power consumption is non-zero. this is due to the small number of days that data was sampled for this load, as can be seen in Table 6.23 As such, no conclusions can be made on its data.

In summary, the ECO dataset shows similar behavior, but better disaggregation quality when compared to the simulation containing all hypothetical loads. This is likely due to the higher number of data points and the higher number of electrical features, creating a more robust setting that is less susceptible to the effects of data complexity and creating a better disaggregation environment.

6.4 Conclusion

In this chapter, we presented PSIM as the simulation environment for the performance analysis of this dissertation, which proved to be well-suited for the simulation of loads for the purposes of load disaggregation. We also presented the simulation categories for this dissertation, which helped get an understanding on what are the main sources of impurity for the NILM algorithm. The tests made for these categories helped identify that data complexity is the biggest source of said impurities, the error in prediction increasing exponentially with the number of considered loads. Analyzing the performance of *Scikit-learn's Random Forest Regression* with the ECO dataset also helped us demonstrate that an increase in features would lead to a more accurate disaggregation process. Overall, in order to properly deploy this algorithm in a household with multiple appliances considered for disaggregation, we would require the sampling of more features so that the predicted power consumption of the more complex loads could be properly disaggregated.

Chapter 7

Conclusion and Further Work

In this dissertation, a study was conducted into the application of machine learning techniques in the field of non-intrusive load monitoring in order to find a disaggregation algorithm that is lightweight and efficient. When it comes to accurately disaggregating appliance-level power consumption, the application of machine learning, which can include both shallow learning and deep learning methods, has shown the capability to accurately disaggregate the individual consumption of different types of appliances.

The Random Forest Regressor, a shallow learning ML ensemble algorithm that takes the predictions of multiple decision trees trained on different sets of data into account to make its final prediction, was selected due to its computational efficiency and accuracy of predictions when compared to other resource-intensive algorithms. In this dissertation, the parameters of this regression, such as the number of trees in the forest and the maximum growth depth of said trees, were iteratively tweaked in order to reach a balance between the accuracy of the algorithm and its memory usage, allowing it to be tested and executed on a Raspberry Pi 3 Model B

The selection of a dataset and electrical features was also an important phase for this dissertation. While there are a lot of listed datasets in the state-of-the-art chapter, due to the lightweight nature of the desired NILM architecture, we would need to choose a dataset containing features sampled at a low frequency and with enough data complexity to thoroughly test the accuracy of the regression.

For these tests, the dissertation presented the PSIM simulation environment for performance evaluation, which turned out to be suitable for simulating hypothetical loads. This was one of the main contributions of the dissertation. It was established, through the use of a number of different simulation categories, that the complexity of the data was the largest barrier to accurate disaggregation. In addition, trials conducted with the ECO dataset indicated that an increase in features led to improved disaggregation accuracy. These findings underscore the significance of sampling additional features when dealing with complex load behavior in a household environment.

In conclusion, this dissertation offers insightful knowledge concerning the utilization of machine learning methods in the construction of a NILM architecture that is accurate and memory efficient. The proposed system provides a promising strategy for accurately disaggregating the

amount of energy that is consumed in homes that have more than one appliance. The studied NILM algorithm could benefit from an additional investigation in the electrical features field in order to broaden its range of accurately disaggregated loads and achieve higher levels of performance quality. Another area for further work in this area would be the development of a NILM device based on a microprocessor capable of sampling the electrical features from a network, executing the developed algorithm, and then communicating its predictions to a user.

Appendix A

Data Organization and Cleaning Code

The code present in this appendix was used in order to clean and reorganize the ECO dataset [16].

A.1 Rename_Files.py

This script was designed to look for all the "*.csv*" label files in the directory containing the Eco dataset, and renaming them by adding a prefix to the files according to their specific label, from "01_" to "12_", so they they are more easily readable.

```
import os

for i in range(1, 13):
    print(i)
    directory = "C:/Users/Professor/Desktop/tese/02/" + str(i).rjust(2, '0') #
    change this directory to your own
    prefix = str(i).rjust(2, '0') + "_"
    print(prefix)
    print(directory)

    for filename in os.listdir(directory):
        if not filename.startswith(prefix):
            new_filename = prefix + filename
            os.rename(os.path.join(directory, filename), os.path.join(directory,
    new_filename))
```

Listing A.1: Python script designed to rename the ECO dateset files

A.2 Group_Labels.py

The purpose of this script is to group all label files into a single file, containing all the measurements for each day, by loading all the "*.csv*" label files into a Numpy array, joining them in a single matrix, with each column representing each label, and then saving it in a single master file. If there is no file for a specific day, it is substituted in the master file by a column of zeros.

```
import os

import numpy as np

agg_data_dir = "C:/Users/freix/Desktop/tese/files/02/measured_data"

for agg_data_date in os.listdir(agg_data_dir):
    label_matrix = np.zeros((86400,1))
    for i in range(1,13):
        directory = "C:/Users/Professor/Desktop/tese/02/" + str(i).rjust(2,'0')+ "/
    " + str(i).rjust(2,'0') + "_" + agg_data_date
        print(directory)
        try:
            array = np.array([np.loadtxt(directory, delimiter=',')]).transpose()
        except:
            array = np.zeros((86400,1))
        # print(array)
        label_matrix = np.hstack((label_matrix,array))
        # print(label_matrix)
    label_matrix = np.delete(label_matrix,0,1)
    # print(label_matrix)
    directory = "C:/Users/Professor/Desktop/tese/02/labels/labels_" + agg_data_date
    np.savetxt(directory, label_matrix, delimiter=',')
```

Listing A.2: Python script designed to Group the Eco Dateset Label files

A.3 Clean_Missing_Data.py

This script goes through all the master label files looking for missing values. Upon finding a missing value, it stores its index into an array. This array is then inverted in order to delete lines from both the master label matrix and the electrical feature matrix from the end to the beginning, avoiding the miss-matching of indexes due to their repositioning upon deleting a line. When all the lines containing missing values are deleted, the clean label and measurement matrices are again saved.

```
import os

import numpy as np

agg_data_dir = "C:/Users/freix/Desktop/tese/files/02/measured_data"

for agg_data_date in os.listdir(agg_data_dir):
    if os.path.isfile("C:/Users/freix/Desktop/tese/files/02/clean_labels/
    clean_labels_" + agg_data_date):
        print("File exists")
        continue
    print("working on" + agg_data_date)
```

```
    label_directory = "C:/Users/freix/Desktop/tese/files/02/labels/labels_" +
    agg_data_date
    agg_directory = "C:/Users/freix/Desktop/tese/files/02/measured_data/" +
    agg_data_date
    agg_matrix = np.loadtxt(agg_directory, delimiter=',')
    label_matrix = np.loadtxt(label_directory, delimiter=',')
    line_index = []
    for i in range(len(label_matrix)):
        if -1 in label_matrix[i]:
            line_index.append(i)
    line_index.reverse()
    for i in line_index:
        label_matrix = np.delete(label_matrix,i,0)
        agg_matrix = np.delete(agg_matrix,i,0)
    label_directory = "C:/Users/freix/Desktop/tese/files/02/clean_labels/
    clean_labels_" + agg_data_date
    agg_directory = "C:/Users/freix/Desktop/tese/files/02/clean_measured_data/
    clean_measured_data_" + agg_data_date
    np.savetxt(label_directory, label_matrix, delimiter=',')
    np.savetxt(agg_directory, agg_matrix, delimiter=',')
    del label_matrix
    del agg_matrix
    del line_index
    print(" done with " + agg_data_date)
```

Listing A.3: Python script designed to clean the missing labels from the Eco Dateset

A.4 Join_Data.py

This final Script takes all the clean data from the label and electrical feature measurements and joins them into two files containing all the data, in order to be used in the Random Forest training process.

```
import os

import numpy as np

agg_data_dir = "C:/Users/freix/Desktop/tese/files/02/measured_data"
master_data_matrix = np.zeros((1, 16))
master_label_matrix = np.zeros((1, 12))
for agg_data_date in os.listdir(agg_data_dir):
    label_directory = "C:/Users/freix/Desktop/tese/files/02/clean_labels/
    clean_labels_" + agg_data_date
    agg_directory = "C:/Users/freix/Desktop/tese/files/02/clean_measured_data/
    clean_measured_data_" + agg_data_date
    agg_matrix = np.loadtxt(agg_directory, delimiter=',')
    label_matrix = np.loadtxt(label_directory, delimiter=',')
    master_data_matrix = np.vstack((master_data_matrix, agg_matrix))
```

```
14      master_label_matrix = np.vstack((master_label_matrix, label_matrix))
15      del label_matrix
16      del agg_matrix
17      print("done with " + agg_data_date)
18  master_label_directory = "C:/Users/freix/Desktop/tese/files/02/Master_data/
        master_label.csv"
19  np.savetxt(master_label_directory, master_label_matrix, delimiter=',')
20  master_data_directory = "C:/Users/freix/Desktop/tese/files/02/Master_data/
        master_data.csv"
21  np.savetxt(master_data_directory, master_data_matrix, delimiter=',')
```

Listing A.4: Python script designed to join all label and feature files from the Eco Dateset

Appendix B

PSIM Schematic and Simulated Loads Desciption

	R1	R2	R3	R4	R5
Resistance	2000 Ω	3000 Ω	2500 Ω	25 Ω	250 Ω
Power consumption	13.7 W	9.1 W	10.9 W	1,09 kW	110 W
Frequency	10 mHz	50 mHz	10 mHz	10 mHz	108,7 mHz
Duty Cycle	0.7	0.6	0.85	0.6	0.5

Table B.1: Five simulated Type I resistive loads

	RL95	RL90	RL60	RL80	RL85
Resistance	100 Ω	500 Ω	252 Ω	27 Ω	750 Ω
Inductance	104.62 mH	770 mH	803 mH	64.45 mH	1.48 H
Power Consumption	250.2 W	46.3 W	68.2 W	736.7 W	31.9 W
Frequency	0.1 Hz	0.6 Hz	0.3 Hz	50 mHz	0.2 Hz
Duty Cycle	0.3	0.5	0.6	0.6	0.7

Table B.2: Five simulated Type I inductive loads

	VR1	VR2	VR3
Resistance	2000 Ω	1000 Ω	30 Ω
Max Power Consumption	13.7 W	27.3 W	939.5 W
Frequency	10 mHz	24 mHz	57 mHz
Duty Cycle	0.7	0.7	0.7

Table B.3: Three simulated type II resistive loads

	VRL84	VRL92	VRL88
Resistance	1500 Ω	1500 Ω	23 Ω
Inductance	3.084 H	2.034 H	39.5 mH
Max Power Consumption	18.3 W	18.3 W	1.24 kW
Frequency	17 mHz	31 mHz	48 mHz
Duty Cycle	0.6	0.55	0.75

Table B.4: Three simulated type II inductive loads

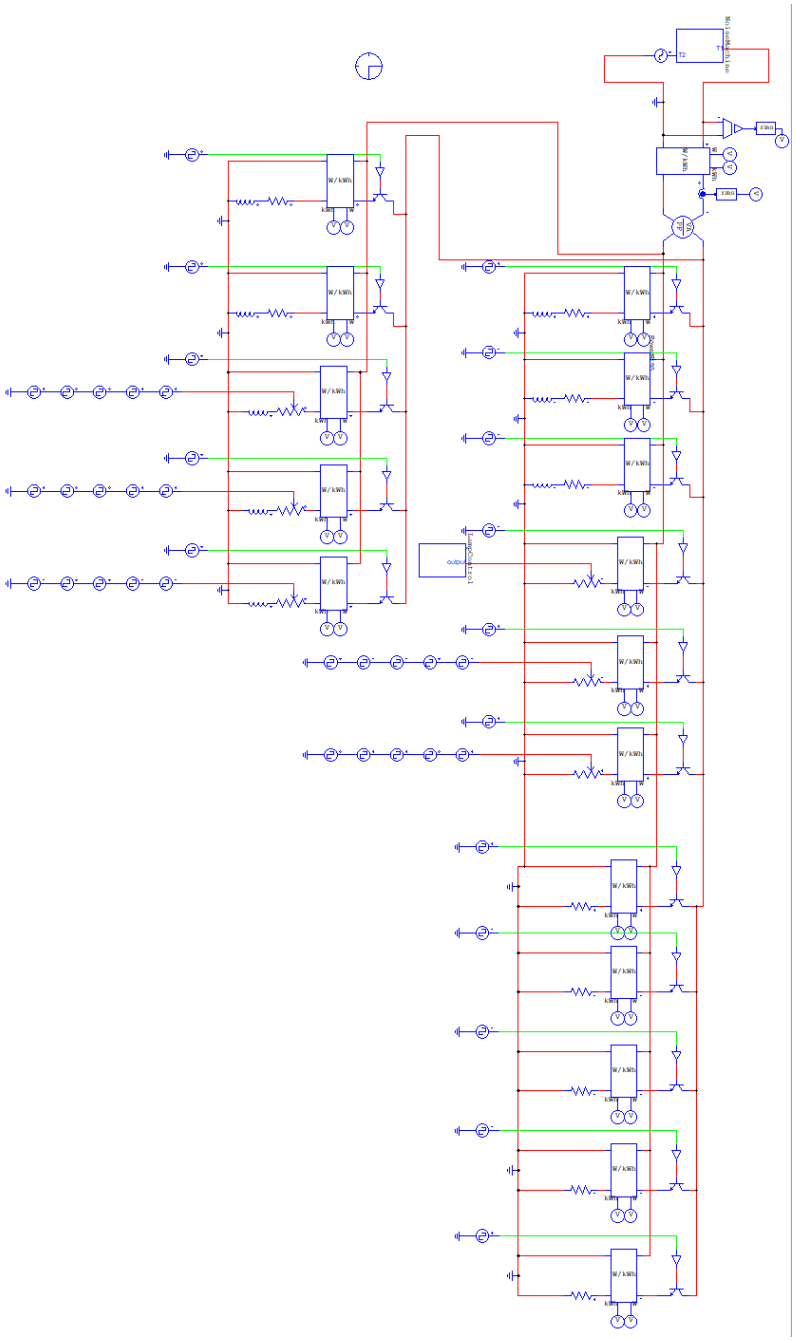


Figure B.1: Schematic developed in PSIM for load simulation

References

[1] Suryalok Dash and N.C. Sahoo. Electric energy disaggregation via non-intrusive load monitoring: A state-of-the-art systematic review. *Electric Power Systems Research*, 213:108673, December 2022. URL: https://linkinghub.elsevier.com/retrieve/pii/S0378779622007398, doi:10.1016/j.epsr.2022.108673.

[2] Xin Wu, Yuchen Gao, and Dian Jiao. Multi-Label Classification Based on Random Forest Algorithm for Non-Intrusive Load Monitoring System. *Processes*, 7(6):337, June 2019. URL: https://www.mdpi.com/2227-9717/7/6/337, doi:10.3390/pr7060337.

[3] Christian Beckel, Wilhelm Kleiminger, Romano Cicchetti, Thorsten Staake, and Silvia Santini. The eco data set and the performance of non-intrusive load monitoring algorithms. In *Proceedings of the 1st ACM International Conference on Embedded Systems for Energy-Efficient Buildings (BuildSys 2014). Memphis, TN, USA*, pages 80–89. ACM, November 2014.

[4] Georgios-Fotios Angelis, Christos Timplalexis, Stelios Krinidis, Dimosthenis Ioannidis, and Dimitrios Tzovaras. NILM applications: Literature review of learning approaches, recent developments and challenges. *Energy and Buildings*, 261:111951, April 2022. URL: https://linkinghub.elsevier.com/retrieve/pii/S0378778822001220, doi:10.1016/j.enbuild.2022.111951.

[5] Pascal A. Schirmer and Iosif Mporas. Non-intrusive load monitoring: A review. *IEEE Transactions on Smart Grid*, pages 1–1, 2022. doi:10.1109/TSG.2022.3189598.

[6] R. Gopinath, Mukesh Kumar, C. Prakash Chandra Joshua, and Kota Srinivas. Energy management using non-intrusive load monitoring techniques – State-of-the-art and future research directions. *Sustainable Cities and Society*, 62:102411, November 2020. URL: https://linkinghub.elsevier.com/retrieve/pii/S2210670720306326, doi:10.1016/j.scs.2020.102411.

[7] N. Sadeghianpourhamami, J. Ruyssinck, D. Deschrijver, T. Dhaene, and C. Develder. Comprehensive feature selection for appliance classification in NILM. *Energy and Buildings*, 151:98–106, September 2017. URL: https://linkinghub.elsevier.com/retrieve/pii/S0378778817314366, doi:10.1016/j.enbuild.2017.06.042.

[8] A. G. Ruzzelli, C. Nicolas, A. Schoofs, and G. M. P. O'Hare. Real-Time Recognition and Profiling of Appliances through a Single Electricity Sensor. In *2010 7th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pages 1–9, Boston, MA, USA, June 2010. IEEE. URL: http://ieeexplore.ieee.org/document/5508244/, doi:10.1109/SECON.2010.5508244.

[9] Feng-Jen Yang. An implementation of naive bayes classifier. In *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 301–306, 2018. doi:10.1109/CSCI46756.2018.00065.

[10] Chuan Choong Yang, Chit Siang Soh, and Vooi Voon Yap. A non-intrusive appliance load monitoring for efficient energy consumption based on Naive Bayes classifier. *Sustainable Computing: Informatics and Systems*, 14:34–42, June 2017. URL: https://linkinghub.elsevier.com/retrieve/pii/S2210537916301469, doi:10.1016/j.suscom.2017.03.001.

[11] Fitra Hidiyanto and Abdul Halim. KNN Methods with Varied K, Distance and Training Data to Disaggregate NILM with Similar Load Characteristic. In *Proceedings of the 3rd Asia Pacific Conference on Research in Industrial and Systems Engineering 2020*, pages 93–99, Depok Indonesia, June 2020. ACM. URL: https://dl.acm.org/doi/10.1145/3400934.3400953, doi:10.1145/3400934.3400953.

[12] Ziwei Xiao, Wenjie Gang, Jiaqi Yuan, Ying Zhang, and Cheng Fan. Cooling load disaggregation using a NILM method based on random forest for smart buildings. *Sustainable Cities and Society*, 74:103202, November 2021. URL: https://linkinghub.elsevier.com/retrieve/pii/S2210670721004807, doi:10.1016/j.scs.2021.103202.

[13] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Random Forests. In *The Elements of Statistical Learning*, pages 587–604. Springer New York, New York, NY, 2009. Series Title: Springer Series in Statistics. URL: http://link.springer.com/10.1007/978-0-387-84858-7_15, doi:10.1007/978-0-387-84858-7_15.

[14] 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.

[15] Scikit-learn: Contributing. URL: https://scikit-learn/stable/developers/contributing.html.

[16] Wilhelm; Beckel Kleiminger. ECO data set (Electricity Consumption & Occupancy), 2016. URL: http://data-archive.ethz.ch/delivery/DeliveryManagerServlet?dps_pid=IE594964, doi:10.5905/ETHZ-1007-35.

[17] sklearn.ensemble.RandomForestRegressor. URL: https://scikit-learn/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html.

[18] PSIM: Unbeatable Power Electronics Software. URL: https://powersimtech.com/products/psim/capabilities-applications/.