UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

UNIVERSITÀ DI MODENA E REGGIO EMILIA

GIOVANNI BRAGLIA

UM SISTEMA EMBARCADO PARA MONITORAMENTO DE CARGA NÃO INTRUSIVA USANDO APRENDIZADO DE MÁQUINA

THESIS

CURITIBA / MODENA

2021

GIOVANNI BRAGLIA

UM SISTEMA EMBARCADO PARA MONITORAMENTO DE CARGA NÃO INTRUSIVA USANDO APRENDIZADO DE MÁQUINA

An Embedded System for Non-Intrusive Load Monitoring using Machine Learning

Trabalho de conclusão de curso de Tese apresentada como requisito para obtenção do título de Mestre em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná (UTFPR).

Supervisor: Prof. Dr. André Eugenio Lazzaretti

Thesis presented as a requirement to obtain the title of Doctor in Electronics Engineering for the University of Modena and Reggio Emilia (Unimore).

Supervisor: Prof. Dr. Laura Giarrè
Co-Supervisor: Prof. Dr. Giovanni Franceschini

CURITIBA / MODENA

2021

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Curitiba

GIOVANNI BRAGLIA

UM SISTEMA EMBARCADO PARA MONITORAMENTO DE CARGA NÃO INTRUSIVA USANDO APRENDIZADO DE MÁQUINA

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestre Em Ciências da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Engenharia De Computação.

Data de aprovação: 20 de Outubro de 2021

Prof Andre Eugenio Lazzaretti, Doutorado - Universidade Tecnológica Federal do Paraná

Prof Fabio Kurt Schneider, Doutorado - Universidade Tecnológica Federal do Paraná

Prof.a Laura Giarre, Doutorado - Università Degli Studi Di Modena e Reggio Emilia

Prof Luiz Felipe Ribeiro Barrozo Toledo, Doutorado - Instituto de Tecnologia para Desenvolvimento, Departamento de Eletricidade, Divisão de Sistemas Elétricos - Lactec

Prof Paolo Pavan, Doutorado - Università Degli Studi Di Modena e Reggio Emilia

Documento gerado pelo Sistema Acadêmico da UTFPR a partir dos dados da Ata de Defesa em 22/10/2021.

ACKNOWLEDGEMENTS

Quando penso al fatto che 5 anni fa ero esattamente agli inizi del mio percorso universitario, penso a quanto questo intervallo di tempo abbia inciso su di me, sulle mie idee, sulle mie ambizioni. Eppure se guardo indietro sembra che sia durato tutto 5 minuti, momenti che non potrò rivivere ma che posso ricordare, grazie a quali non potrei essere la stessa persona che scrive queste righe.

Vorrei dunque sfruttare questo momento per ricordare e ringraziare tutte quelle persone che hanno partecipato, in piccola o grande parte, alla realizzazione di quello che per me è un grande traguardo.

Innanzitutto non sarei riuscito a laurearmi se non fosse stato per i professori che ho avuto, quelli che ti motivano, quelli sempre disponibili, quelli disordinati, quelli noiosi, quelli che ti fanno studiare tanto e quelli che ti costringono a risolvere i problemi da solo. Sono certo che con ognuno di loro ha avuto qualcosa da apprendere, i loro insegnamenti saranno sempre le basi su cui proietterò la mia vita lavorativa di ingegnere. Ringrazio in particolare i professori Giarré e Franceschini, per la loro disponibilità a seguirmi in questo progetto di tesi e a scambiare opinioni su un campo che non era sempre il loro.

Se dovessi ricordare in particolare uno di questi 5 anni di università non posso che finire a parlare del 2020, anno della pandemia per tanti, ma anno del Brasile per me. Ringrazio dunque Francesca e Paolo, per il progetto di scambio meraviglioso che hanno creato tra Unimore e UTFPR, per la loro constante presenza, per gli aiuti che hanno saputo darmi prima, durante e dopo il periodo di mobilità, per l'incredibile opportunità che mi hanno dato.

Volevo dunque ringraziare tutti quelli che hanno fatto parte della mia esperienza in Brasile. Ringrazio UTFPR per la sua accoglienza, per il riguardo verso la mia situazione. I professori, per la loro disponibilità, per il loro interesse verso gli alunni. Ringrazio in particolare André, per la passione che ha saputo trasmettermi su tematiche a me nuove, per la sua premura nei miei confronti, per le chiamate con lui al venerdì e per avermi portato una chitarra durante il periodo di quarantena. E' sicuramente grazie lui se, nei primi mesi del 2020, ho deciso di rimanere in Brasile nonostante il dilagare dell'epidemia.

Ringrazio poi tutte le persone che mi sono state vicine, quelle che ci hanno tenuto a farmi conoscere le usanze brasiliane, ringrazio i miei coinquilini, i miei amici. Ringrazio Cacau, mentre scrivo queste righe la sto aspettando in aeroporto ed è notte inoltrata. La ringrazio perchè

da quando l'ho conosciuta non cè stato un secondo in cui non mi abbia dimostrato quanto mi volesse bene, perchè sà starmi vicina, per la calma e l'allegria che sà trasmettermi.

Ringrazio tutte le persone, amici, conoscenti e parenti che in un qualche modo hanno saputo contribuire con poco ad attimi di felicità. In particolare ringrazio i ragazzi del campo Rwanda 2019, perchè pur non vedendoci spesso sanno sempre dar un valore particolare alla nostra amicizia, mantenendo i contatti e soprattutto organizzando rimpatriate che sanno sempre rendere speciali.

Gli anni dell'università sono poi stati anche gli anni della band, vorrei dunque ringraziare i MangoStreet, per avermi aspettato, perchè suonare con voi alla fine è sempre una scusa per vederci, per avermi insegnato che si riesce sempre a trovare del tempo per dedicarsi alle proprie passioni.

Ringrazio poi Claudio, Luca, Marta, Simone, Stefano e Valeria, per essere sempre stati una sicurezza, per la loro presenza in tutti i momenti più importanti, per la loro amicizia che cè sempre.

Ringrazio infine la mia famiglia, perchè hanno sempre saputo appoggiarmi in qualsiasi scelta, darmi una soluzione o dirmi che tutto andrà bene. Grazie Valeria e Matteo, perchè quando siete a casa o siamo insieme tutto ha colore. Grazie mamma e papà, perchè mi ispirate giorno dopo giorno.

RESUMO

BRAGLIA, Giovanni. **Um Sistema Embarcado para Monitoramento de Carga nao Intrusiva usando Aprendizado de Màquina**. 2021. 116 f. Thesis (Double Degree em Engenharia Elétrica e Informática Industrial e Electronics Engineering) – Universidade Tecnológica Federal do Paraná e Università di Modena e Reggio Emilia. Curitiba / Modena, 2021.

O interesse em sistemas de gerenciamento de energia tem crescido nos últimos anos, como que a maioria das plantas industriais ou domésticas adotam técnicas para reduzir com eficiência a demanda de energia e os custos relacionados a ela. Uma solução atrativa são os sistemas de monitoramento de carga não intrusiva (NILM), cujo objetivo principal é encontrar uma forma mais adequada de acompanhar o consumo de energia causado por cada uma das cargas que estão conectadas à planta monitorada. Uma possível implementação na vida real de um sistema NILM é abordada neste trabalho, discutindo todos os blocos fundamentais em sua estrutura, incluindo detecção de eventos, extração de features e classificação de carga, usando dataset disponíveis publicamente. Além disso, oferecemos uma solução para um sistema embarcado, capaz de analisar formas de onda agregadas e reconhecer a contribuição de cada aparelho nas mesmas. O projeto é então completado por medições de cargas realizadas em laboratório, com o intuito de validar ainda mais o algoritmo proposto e sua viabilidade, envolvendo a criação de um novo dataset de dados de carga e características. A metodologia adotada, suas características, desvantagens e implementação são explicados, mostrando os desafios atuais e futuros para a aplicação final.

Palavras chave: Sistemas Embarcados. Aprendizado de Maquina. NILM. Gerenciamento de Energia. Processamento de Sinais.

ABSTRACT

BRAGLIA, Giovanni. **An Embedded System for Non-Intrusive Load Monitoring using Machine Learning**. 2021. 116 p. Thesis (Double Degree in Engenharia Elétrica e Informática Industrial and Electronics Engineering) – Universidade Tecnológica Federal do Paraná and Università di Modena e Reggio Emilia. Curitiba / Modena, 2021.

The interest in power managing systems has been growing in recent years since every industrial or domestic plant moves towards techniques to efficiently reduce energy demand and costs related to it. An attractive solution is represented by Non-Intrusive Load Monitoring (NILM) systems, whose primary purpose is to find a more appropriate way of keeping track of the power consumption caused by each of the loads that are connected to the monitored plant. A possible real-life implementation of a NILM system is addressed in this work, discussing all the fundamental blocks in its structure, including detecting events, feature extraction, and load classification, using publicly available datasets. Additionally, we provide a solution for an embedded system, able to analyze aggregated waveforms and to recognize each appliance's contribution in it. The project is then completed by real loads' measurements performed in lab, with the intention of further validate the proposed algorithm and its feasibility, involving the creation of a new load and features dataset. The adopted methodology, its features, drawbacks, and implementation are thus explained, showing current and future challenges for the final application.

Keywords: Embedded System. Machine Learning. NILM. Power Management. Signal Processing.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

ABBREVIATIONS

art.	Article
cap.	Chapter
fig.	Figure
sec.	Section
tab.	Table

INITIALISM

AC	Alternate Current
angp	Angle between maximum and minimum point
APP	Appliance
ar	Area
asy	Asymmetry
AUC	Area Under Curve
CNN	Convolutional Neural Networks
COOLL	Controlled On/Off Loads Library
CPU	Central Processing Unit
DC	Direct Current
dpb	Distance between maximum and minimum point
DSP	Digital Signal Processor
DWT	Discret Wavelet Transform
FFT	Fast Fourier Transform
FLAC	Free Lossless Audio Codec
FP/FN	False Positive/Negative
GPU	Graphical Processing Unit
HAND	High Accuracy NILM Detector
HCapP	Half-Cycle Apparent Power
IDE	Integrated Development Environment
IG	Information Gain
itc	Current Span
KNN	K-Neighbors Classifier
L	Length
LIT	Laboratory for Innovation and Technology in Embedded Systems
lpa	Area with loop direction

LR	Logistic Regression
M	Mean Line
md	Maximum distance
MElting Lab	Industrial Automation and Power Electronics Laboratory
NaN	Not a Number
NILM	Non-Intrusive Load Monitoring
OVR	One-Versus-Rest
P	Active Power
PCA	Principal Component Analysis
Q	Reactive Power
r	Curvature of mean line
RBF	Radial Basis Function
RF	Random Forest
RMS	Root Mean Square
ROC	Receiving Operating Characteristic
S	Apparent Power
SBS	Sequential Backward Selection
sc	Number of self intersections
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TED	Total Even Harmonics Distortion
TOD	Total Odd Harmonics Distortion
TP/TN	True Positive/Negative

SUMMARY

1 INTRODUCTION

1.1 MOTIVATION

Energy saving always had an important role in all fields. From companies, houses or devices (being small or even huge), one of the parameters that can really make the difference in terms of costs is power consuming. Moreover, we are globally facing a huge environmental crisis, mainly due to carbon dioxide emissions.

Fossil fuels finite quantities are forcing us to move to renewable energy sources, since the huge growth of population will increase energy demand. Therefore, a way to reduce energy consumption is needed, which can be achieved with new sources, but also finding a way to wisely manage those ones.

NILM (Non-Intrusive Load Monitoring) is part of a research field whose goal is propose energy savings solutions. Here the keyword is efficiency, since NILM aims at producing systems for power consumption monitoring, basically developing a device able to recognize loads connected to an electrical system and display the amount of energy that is being used. In this way, people will always be aware in real-time about the devices that are on, knowing exactly how much energy they are using (HART, 1992).

A first way to do that could be install a sensor for every electric socket of the system that has to be monitored, to directly get information from the latter. However, for large systems, being monitored means needing a high amount of sensors which of course will be translated into higher costs (CHANG, 2012).

NILM provides in this case another solution, which is performing a non-intrusive analysis of loads by monitoring the whole system power consumption in one point, thus disaggregating, from the whole bunch of devices connected in the same bar, individual appliances ready to be processed and classified.

According to Picon *et al.* (2016) and Figueiredo *et al.* (2012), usually the approach used for NILM considers the following steps:

- Data Acquisition;
- Features Extraction;

- Load Identification.

Even though the first step might be obvious, the way data are acquired should be coherent with the next steps of the approach. For example, one of the parameters we must care about is the sampling frequency, as it determines the type of information that can be extracted from the electrical signals.

Low-frequency systems, for example, allows to capture common power features as active, reactive or apparent power and are more suitable for commerce purposes since they are less costly. However, because of their reduced bandwidth, those systems may lose some important information in signals' transients as well as high-order harmonics, thus loosing important features for the classification process.

The process of feature extraction follows the one of data acquisition. Here the signal referring to a single appliance, after being detected and isolated, will be analyzed in order to extract the features necessary for the appliance classification.

During this last passage, the load will be identified by a classifier embedded in the NILM system. The use of a classifier implies that it was already evaluated and selected. In fact, among the steps listed above, the model should be trained over the available data for improving its classification performance.

Different types of classifiers can be used based on their accuracy over tests. In particular, training methods can be split into two categories: on-line and off-line (MULINARI *et al.*, 2019). In case of on-line training, time window based methods for real-time detection and learning of appliances' features are used. However, upon detection of load events, manual labeling of the appliances is challenging and complex as only the aggregated load values are observed instead of individual appliance measurements (ZOHA *et al.*, 2012). In this work, since two datasets were used, an off-line method was employed. Off-line simply means that the classifier has been trained over a dataset built upon signals that were collected and then correctly labeled, in a way to have specific references to the appliances belonging to each signal.

Off-line training methods are usually the most employed ones (ZOHA *et al.*, 2012), although we are always limited by the amount of information provided by datasets, where it's often difficult to find large variety of devices that could be useful to generalize over new data.

1.2 OBJECTIVES

1.2.1 General Objective

In this work, the main purpose is to develop the algorithm for an embedded system able to perform NILM with high-sampled voltage and current signals.

1.2.2 Specific Objectives

The whole general objective can then be divided in smaller tasks that are:

- Build a features dataset from the existing signals;
- Evaluate the dataset by maximizing classifiers' accuracy;
- Test if the whole system is working on the two datasets: COOLL (PICON *et al.*, 2016) and LIT Dataset (RENAUX *et al.*, 2018b).
- Develop an algorithm for signal detection and load disaggregation;
- Integrate the algorithms into the selected system.

Moreover, this work includes a final experiment, were the whole algorithm was further validated over lab measurements of different appliances.

1.3 STRUCTURE OF THIS DOCUMENT

This document is organized in six chapters. In the first one, an introduction was provided touching the main topics that will be discussed throughout the document, then:

- Chapter 2 introduces state of the art techniques and related works, to have a more clear view of the actual scenario;
- Chapter 3 presents theoretical aspects worth to mention for the purposes of the project;
- Chapter 4 will be the core of the document, analyzing the algorithms and sources used and going through practical needs and implementations of the project;
- Chapter 5 shows the results obtained;
- Chapter 6 concludes this work.

2 RELATED WORKS

In this chapter, the main topics that have been used for the development of this work are discussed. In particular, a first general view of the methods adopted is provided, giving the big picture of NILM scenario, and going through state-of-the-art techniques and approaches.

2.1 NON-INTRUSIVE LOAD MONITORING

NILM may well become a widespread diagnostic and energy management solution, in the context of Electrical Efficiency, available to every end-user. As a diagnostic tool, it identifies energy waste and improper use for energy management and may, at the end, be employed by residential and commercial/industrial users (Pöttker *et al.*, 2018).

This non-intrusive technique aims at providing a solution to monitor energy consumption in a centralized fashion, without having the need of using multiple sensors for the measures of individual appliances. In practice, the system uses only aggregated power consumption data from one instrument installed at main power distribution bus, then the disaggregated power consumption data of the selected appliances is saved, analyzed and classified, displaying those that are turned-on and off, and information about their power consumption. This will, therefore, bring significant savings in energy consumption by improving energy management. Usually, the NILM approach includes three steps:

1. Load detection and disaggregation;
2. Features extraction;
3. Classification.

It is useful to detect loads from the aggregated signal and isolate them properly. The correct isolation of the load is fundamental for the next steps and will be discussed in the next section. In the sequence, features extraction process is performed. This passage aims at extracting all those characteristics of the signal that have to be evaluated from the classifiers, in order to make their own prediction. The extracted appliance features are then further analyzed by the load identification algorithms, in order identify appliance-specific states from the aggregated measurement.

Most of the research works in NILM are focused on supervised machine learning techniques that require labeled data for training the classifiers. Nowadays, most of the supervised

learning methods for loads classification are either optimization based or pattern recognition approaches.

The optimization approach tries to match the observed power measurements $p(t)$ to a possible combination of appliance power signals (present already in the database). Optimization tries to find the minimum residue while comparing the unknown loads with a set of candidates extracted from the known database (DU *et al.*, 2010). However, one major drawback is that the presence of unknown loads in $p(t)$ complicates the optimization problem as the method attempts to provide a solution based on the combination of known appliances.

Therefore the pattern recognition approach has been a preferred method by researchers for the task of load identification (WANG *et al.*, 2018b; FIGUEIREDO *et al.*, 2012). Similar to pattern matching, extracted features are matched with a pool of load signatures already available in the appliance feature database in order to identify an event associated with a operation of an appliance.

Another approach could be based on unsupervised learning. Recently, researchers have shown an increased interest in unsupervised methods for the load disaggregation so that the need for data annotation can be eliminated (RUANO *et al.*, 2019). Unlike most of the supervised load disaggregation approaches that rely on detection of events for classification, the unsupervised methods are non-event-based. These methods make use of unsupervised learning techniques and attempts to disaggregate the aggregated load measurements directly without performing any sort of event detection, detailed as follows.

2.2 EVENT DETECTION

Event detection is one of the most important steps when in load monitoring. One of the main problems is that, as long as a detector is not working properly, result analysis will be very laborious. Different technical problems are related to detection and will be discussed in Chapter 5.

Nowadays (RENAUX *et al.*, 2018b), typically an ensemble of detectors is used for augmenting the quality and the probability of an event being detected. Here, some of the most common detectors will be presented.

In Nait Meziane *et al.* (2017), an algorithm called HAND (High Accuracy NILM Detector) is proposed. This algorithm tracks the envelope of a signal and computes its variance, in a way that transients can be associated to the parts of the variance with higher amplitudes.

This method turns out to be very accurate, but actually works well just on those transients whose shape is sharp, since smooth transients does not have a higher variance oscillations.

Another proposed detection algorithm is the HCApP (Renaux *et al.*, 2018). The HCApP (Half-Cycle Apparent Power) is a technique that analyzes the apparent power signal transforming it into a binary signal having two values: stable and transient. This method combines the analysis of power triangle components for detect power changes in the signal, thus determining the transient window for both the turn-on and turn-off events.

Similar to HCApP, Discrete Wavelet Transform (DWT) (UKIL; ŽIVANOVIć, 2008) is another detection method that performs the segmentation of the signal by using a wavelet decomposition. Here, what matters is not just that events detection accuracy is improved, but even its automatization, since this method leads to an universal threshold, enabling the whole process to work without defining an empirical threshold.

Even a Kalman Filter could be used for spotting the turn-on and turn-off intervals in a signal (BOLLEN; GU, 2006). The Kalman Filter is able to estimate parameters relative to the signal harmonics, thus reconstructing the signal. The reconstructed signal will have some errors due to the estimation operated by the filter. As expected, the errors will be higher in the region of the signal where more changes occur, which are the turn-on and turn-off events we were looking for. Therefore, from an analysis of the errors between the real and the estimated signal, event detection can be performed.

The last detection approach that is presented uses vectorization (DANTAS *et al.*, 2019; RENAUX *et al.*, 2018a). This method characterized the samples of a signal into valid and non-valid states by considering two parameters to set the constraints. The case of samples considered non-valid is the one relative to events such as abrupt transitions, high derivatives values or high frequency oscillation that are typical of transient states and can thus be used for event detection.

Even though the above mentioned detection approaches could be even grouped together to increase the accuracy of events detection, thus joining their results in an optimal fashion, still this may not be a robust solution since it is difficult to generalize over all the appliances signals. The problems related to detection will be better analyzed in Chapter 5.

2.3 FEATURE EXTRACTION

In NILM, the step after signal acquiring is its processing, so that we can extract the features relevant for our classification purposes. Even this passage could have different

approaches, mainly due to loads characteristics or even to the tools available for building the monitoring system.

Steady-state methods, for example, identify devices based on variations in their steady-state signatures, such as Active, Reactive and Apparent power as well as, for example, current harmonics. Several steady-state features are already present in literature and can be extracted, thus facilitating load identification at even low-cost hardware because of low-sampling rate requirements.

Despite those benefits, these methods show their limitation when loads have more or less the same steady-state shape (ANCELMO *et al.*, 2019). In cases where two or more loads have similar demand levels, algorithms use decision analysis techniques to distinguish among them, based on the assumptions regarding to the usage of these appliances, such as the daily on-time or the length of usage. Another limitation to identify the load using, for example, real and reactive power is that it is based on steady-state power consumption. Therefore, it requires waiting until the transient behavior settles down so that steady-state values can be measured.

To provide a further mean of discrimination, transient state methods for features extraction were introduced (DU *et al.*, 2010). The transient behavior of a load is typically related to the physical task that the load performs. Therefore, most loads observed in the field have repeatable transient profiles, which provide the possibility for identification of variable loads. Transient harmonics power can provide extra information for variable loads, besides the transient power. It is also very useful to identify variable drive connected loads, since the drive startup is generally repeatable, controlled by a microprocessor.

However, in order to analyze the transient power, one of the major drawbacks is that high sampling rate is required, needing then, a more expensive hardware (LIANG *et al.*, 2010). One of the solutions might be joining both steady and transient features extraction, in a way to exploit more expensive hardware even for calculating steady-state features with higher accuracy, thus improving the quality of appliances clustering for classification. This was actually what was done in this work, where a NILM system was developed on an embedded system to analyze two high sampled datasets.

2.4 CLASSIFICATION METHODS

The last step in the sequence is classification. This process aims at analyzing the features we mentioned in the previous section, to effectively recognize the load that has been detected and,

therefore, giving as an output its power consumption. Even here, different classification methods are available, each one providing a different way of approaching the classification problem.

The most widely used approach is the supervised disaggregation methods. Supervised learning requires labeled datasets to train the classifier, so it would be able to recognize appliance operations from the aggregated load measurement (ZOHA *et al.*, 2012). In particular, supervised methods can be mainly divided into optimization or pattern recognition based algorithms.

Optimization methods deal with the task of load disaggregation as an optimization problem. In the case of single load recognition, it compares the extracted feature vector of an unknown load with the ones of known loads present in the pool of the appliance database, then it tries to minimize the error between them to find the closest possible match (MULINARI *et al.*, 2019).

On the other hand, pattern recognition methods exploit databases features to cluster appliances based on their characteristics. In this way, loads are classified by being included in the cluster whose features are more similar to them (NAIT-MEZIANE *et al.*, 2019).

Even though supervised learning is the most robust classification methods because excellent accuracy performances can be obtained, its strong dependency on the datasets used for training the model avoid having good generalization over new appliances. This implies as well an huge effort to build datasets with new appliances, which requires years of data collection.

This is the reason why recently researchers have shown an increased interest in unsupervised learning techniques. These methods make attempt to disaggregate the aggregated load measurements directly without performing any sort of event detection and without evaluating previous information – such as the ones provided by datasets in the case of supervised learning.

For the purposes of building a plug-and-play device to monitor power consumption in different environments, unsupervised approach provides the best solution for the NILM systems to be installed in a target environment with a minimal setup cost, not as supervised load identification whose training requirements for load identification algorithms are expensive and laborious.

Good results (accuracies $> 90\%$) has been reached by the research, suggesting that unsupervised learning might be a good solution in NILM (NAIT-MEZIANE *et al.*, 2017). However, still some very non-trivial problems need to be solved. For example, in the case of multi-state appliance source reconstruction becomes even more challenging as they form several clusters due to multiple states, which results in mixing of the events. For this reason, other

classification techniques, such as deep learning, are considered.

Deep learning was actually conceived in the early 1940 with the first studies on artificial neural networks, but it was never exploited as nowadays because of high algorithms' computational cost. However, deep learning has been recently becoming popular due to the growth of GPUs and libraries such as TensorFlow (ABADI *et al.*, 2015), able to boost performances highly parallelizing codes and thus, reducing computational effort.

One way of using deep learning can be, for example, training a number of neural networks in cascade, which are then used as pattern classifiers to identify the various loads. Each network classifies the family in a specific level. This is the case, for example, of Convolutional Neural Networks (CNN) that extract features directly from signals and fed them into some models made of multiple layers, where filters act over data in order to perform classification.

Steady-state appliance signatures, such as fundamental frequency quantities, current, power and impedance contours, and harmonic frequency current information and distortion power are considered as the inputs of neural networks (DU *et al.*, 2010).

2.5 CONTRIBUTIONS OF THIS WORK

The proposal of this work is to elaborate a load monitoring module able to perform appliances detection and classification. The embedded system that will host all the algorithm has been chosen in a way to process multiple signals in parallel, thus increasing the capacity of the system.

The idea behind the choice of the embedded system was using a powerful board, provided with a GPU, to exploit parallel programming and calculus accuracy and see how far performances can be increased without cost constrains related to the hardware. It is worth mentioning that studies in this context are still underexplored in the literature, which reinforces the need for such developments.

This is actually just a small part of the project presented in (MULINARI *et al.*, 2019; RENAUX *et al.*, 2018b; RENAUX *et al.*, 2018c; Linhares *et al.*, 2018), where the intentions are to use several load monitoring module units controlled by a Center of Operation, whose task is managing each units and analyze power consuming from the data collected.

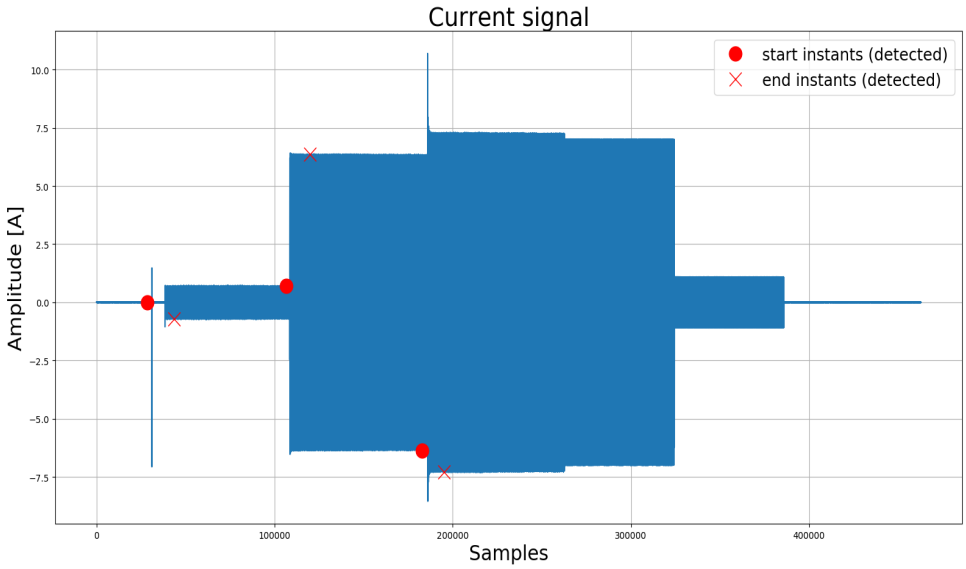
3 THEORETICAL ASPECTS

In the next sections the main theoretical aspects used in this work will be detailed, also providing pseudo-code implementations and references to the Appendix, where all the codes in Python are detailed.

3.1 DETECTION ALGORITHM

Even though event detection is not the core concept behind NILM, it turns out to be a very critical aspect that has to be carefully managed. By event detection it is meant to find a way to label the on and off instants of every appliance inside the aggregated signal (Figure 1). However, since features are calculated for both the transient and the steady-state, the problem reduces to analyze just a window of the signals by detecting start and end instants of the transient, thus taking an interval of the steady state of the same size.

Figure 1 – Events detection in an aggregate signal



Source: Own Autorship

The detector used here is called High Accuracy NILM Detector (HAND) (Nait Meziane *et al.*, 2017). HAND algorithm working principle is very simple and intuitive: the variation of the standard deviation $\sigma_d(t)$ of the signal current's envelope $e_d(t)$ is tracked using a moving window. Then, a threshold separates the events characterized by high $\sigma_d(t)$ amplitude variations and the

one with low variations, referring to the steady-state of signal. The algorithm is summarized as follows:

1. Detect current signal envelope $e_d(t)$;
2. Fix the moving window size W ($W = 4$ for our simulations);
3. Initialize $\sigma_d(t)$, for $k = 1,...,W$, with the standard deviation of $e_d(t)$, $t = t_1, ..., t_W$;
4. Compute iteratively the mean $\mu_d(t_k)$ and the standard deviation $\sigma_d(t_k)$ of $e_d(t)$, $k = W + 1, \ldots, N$ using:

$$\mu_d(t_k) = \mu_d(t_{k-1}) + \frac{1}{W}[e_d(t_k) - e_d(t_{k-W})], \tag{1}$$

$$\sigma_d^2(t_k) = \frac{1}{W^2}\sigma_d^2(t_{k-1}) + \frac{1}{W-1}[e_d(t_k) - \mu_d(t_k)]^2. \tag{2}$$

5. Choose the detection threshold value and find the starting and stopping time for each event such that:
   - start time t_s is defined as the first point of an event where $\sigma_d(t) > threshold$, $\frac{d\sigma_d(t)}{dt} > 0$ and $\frac{de_d(t)}{dt} > 0$;
   - end time t_e is defined as the last point of an event where $\sigma_d(t) < threshold/3$ and $\frac{d\sigma_d(t)}{dt} < 0$;

In the sequence, the pseudo-code for the implementation of the detector (please, refer to the python code .1 in the Appendix A) is presented:

```
FUNCTION DETECTOR(signal, threshold, W = 4)
    temp ← 0
    on-instants : empty array
    off-instants : empty array
    envelope ← LocalMaxima(Module(signal))
    signal-variance : zeroes array of dimension length(envelope)
    signal-mean : zeroes array of dimension length(envelope)

    for i in [0, W] :
        signal-variance[i] ← Variance(envelope)
        signal-mean[i] ← Mean(envelope)
    EndFor
    v-gradient ← Gradient(signal-variance)
```

env-gradient $\leftarrow$ Gradient(*envelope*)

for *i* in [0, Length*envelope*] :

If (*signal-variance*$[i]$ $>$ *threshold*) **and** (*v-gradient*$[i]$ $>$ 0) **and** (*env-gradient* > 0) **and** (*temp* $== 0$) **Then**

temp $\leftarrow 1$

on-instants$[i] \leftarrow i$

EndIf

If (*signal-variance*$[i]$ $<$ *threshold*) **and** (*v-gradient*$[i]$ $<$ 0) **and** (*temp* $== 1$) **Then**

temp $\leftarrow 0$

on-instants$[i] \leftarrow i$

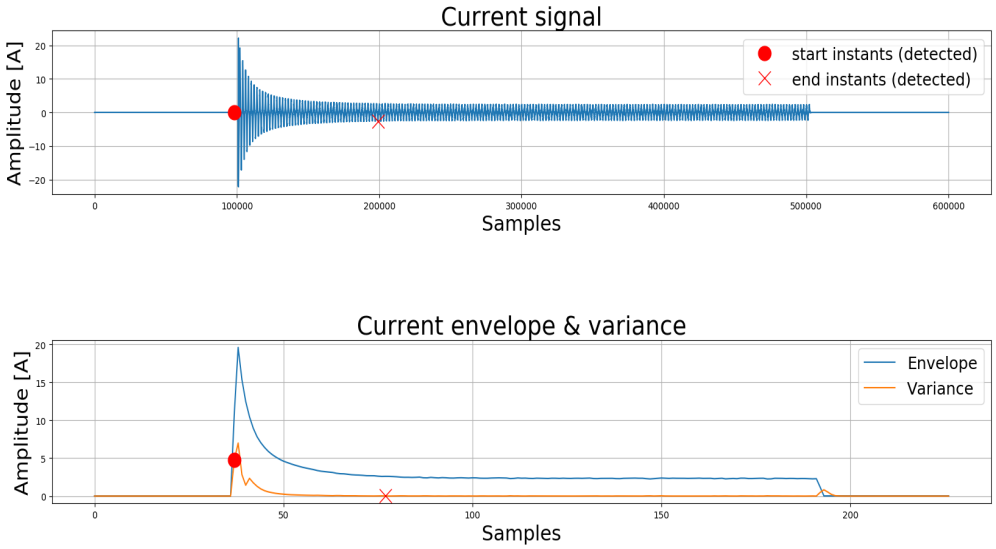
EndIf

EndFor

Return *on-instants, off-instants*

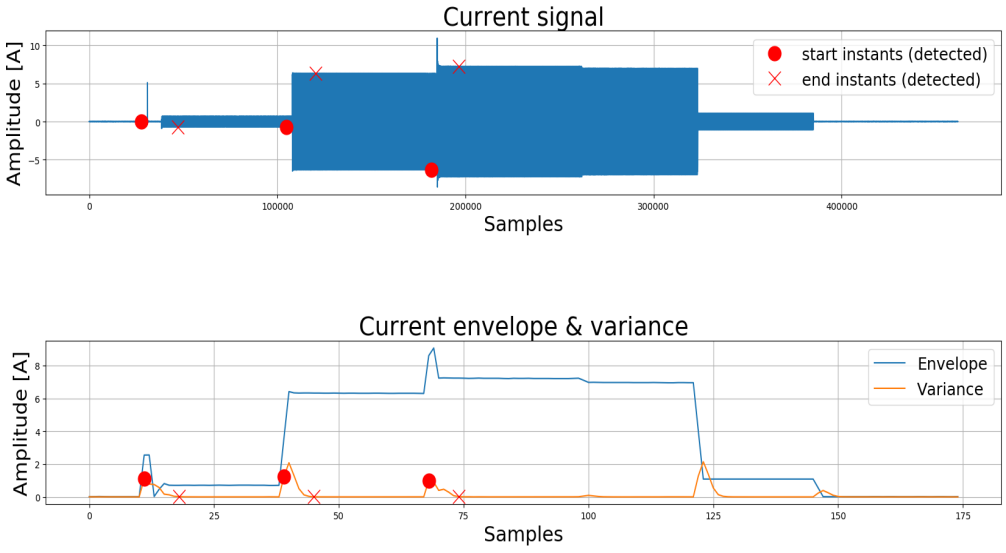
It is noteworthy that $envelope = e_d(t)$, $signal - mean = \mu_d(t_k)$ and $signal - variance = \sigma_d(t)$. As one can see, for the conditions of step 5, a temporary variable $temp$ was included. Since for this work we made an assumption that, for composed signals, each appliance is turned on just when the previous one's transient already finished, $temp$ simply avoid that no other t_s event is detected until the end of the transient that is being measured. This assumption is actually coherent with the signals in both the dataset used (PICON *et al.*, 2016) (RENAUX *et al.*, 2018b). Moreover, in order to avoid wrong detections due to the possible presence of noise, $e_d(t)$ has been smoothed with a moving average filter (please, refer to the python code .2 in the Appendix A).

Figure 2 – t_s and t_e detection for a Drill signal in COOLL Dataset



Source: Own Autorship

Figure 3 – t_s and t_e detection for a composed signal (Led Pannel, Oil Heater Power 1 and Incandescent Lamp) in LIT Dataset



Source: Own Autorship

Figures 2 and 3 show some results from the detection algorithm. Note that, for our purposes, it is just needed to detect turn-on events, thus in the HAND algorithm, a condition over the envelope's gradient was introduced.

3.2 FEATURES

Being able to classify patterns means, in supervised learning, that classifiers were already trained over a dataset that can be basically split into two parts:

1. Label column: gives to every element a target, which is the results that we expect from a correct classification process;
2. Features column: characteristics of an element taken into account to perform classification.

Both COOLL and LIT datasets provide current and voltage signals for each appliance. For the two datasets the same type of features were extrapolated from the signals and collected. In the next sections, the features used to build the dataset for training the classifiers will be shown.

3.2.1 Common Features

Here some of the most common features for signal processing are presented. In particular, in this work, the following features were calculated for the instantaneous power signal calculated as $p(t) = v(t) \times i(t)$, but those could be applied to other type of signals as well.

- **Peak:** Outputs the maximum value of the vector of values (the signal in our case) passed to the function (please, refer to the python code .3 in the Appendix A).
- **Median:** Outputs the value separating the higher half from the lower half of a data sample (please, refer to the python code .4 in the Appendix A).
- **Covariance:** Outputs a value that measures the joint variability between two random variables A and B, in our case the current and the voltage (please, refer to the python code .5 in the Appendix A).
- **Steady-State:** This feature calculates the steady-state value of a signal by taking its envelope, targeting its steady interval and then compiling the mean value (please, refer to the python code .5 in the Appendix A).
- **Harmonics:** This function takes the harmonics of the signal. In particular, we decided to take the harmonics up to the 12th order, since it corresponds to the typical harmonic content of classified loads (ANCELMO *et al.*, 2018). Of course, the more harmonics we take into account, the higher the amount of possible information would be. On the other hand, a high complexity regarding the harmonics computation could cause unacceptable computation burden for the disaggregation mode of the NILM algorithm. Moreover, in Bouhouras *et al.* (2017) authors say that, for harmonic orders higher than 5, the phase of the harmonic currents cannot be ignored, meaning that the harmonic currents should be formed as vectors. In this latter case, voltage measurements should also be performed and synchronized with the measurement of the instantaneous current in order

to obtain the phase of each harmonic. Therefore, as suggested in Hassan *et al.* (2014), in the final experiments over real load measurements just the first five harmonics were individually maintained, while the rest of the harmonic content was included in two different features called TED and TOD, respectively the total even and odd harmonic distortion (please, refer to the python code .7 in the Appendix A).

3.2.2 Electrical Power Features

- **Active Power (P):** Calculated as the mean value of the power, basically gives its DC value of the instantaneous power $p(t)$ (please, refer to the python code .8 in the Appendix A).
- **Reactive Power (Q):** It represents the amount of power that is stored and oscillates back and forth from the source to the device (please, refer to the python code .9 in the Appendix A).
- **Apparent Power (S):** The apparent power S can be view as the potential power that can be transmitted from the source to the device and it is calculated as the product of root mean square (RMS) value of voltage and current (please, refer to the python code .10 in the Appendix A).

3.2.3 V-I Trajectory Features

The following set of features takes into account V-I trajectories of appliances. The codes for calculating those features are the Python interpretation of Matlab codes explained in (Mulinari *et al.*, 2019) . .

As said in Wang *et al.* (2018a), before calculating the features, we need to extrapolate one cycle of the steady-state part of the signal (both for current and voltage), which has to be smoothed with a moving average filter in order to attenuate possible noise components. To generalize more over the analyzed signal interval, instead of taking one of the N cycles inside the latter, the algorithm calculates a cycle where every sample is the average of all the elements with the same phase inside the interval we are analyzing.

The algorithm for cycle extrapolation is detailed as follows (please, refer to the python code .11 in the Appendix A):

```
FUNCTION EXTRAPOLATE CYCLE (current-signal, voltage-signal, n-
samples)
    n-cycles ← int(voltage-signal/(n-samples))
    current-cycles ← current-signal[0 : n-cycles * n-samples]
    voltage-cycles ← voltage-signal[0 : n-cycles * n-samples]
```

```
c-cycles-matrix[n-samples,n-cycles] ← n-cycles vectors
        of n-samples elements from current-cycles
v-cycles-matrix[n-samples,n-cycles] ← n-cycles vectors
        of n-samples elements from voltage-cycles

c-cycle[n-samples, 1] ← Average(c-cycles-matrix)
v-cycle[n-samples, 1] ← Average(v-cycles-matrix)

  Return c-cycle, v-cycle
```

According to Wang *et al.* (2018a), once cycles have been extrapolated, the trajectory is divided in two parts named A and B according to the maximum and minimum points of voltage, thus *vmax* leads to (V_{vmax},I_{vmax}) and *vmin* to (V_{vmin},I_{vmin}). The variable *vmax* then will be the first point of the trajectory, which brings to:

$$A = (V_q, I_q)|q \in 1,2,...,vmin, \tag{3}$$

$$B = (V_q, I_q)|q \in vmin + 1,...,N, \tag{4}$$

where *N* is the number of samples in a cycle. Once the current and voltage cycles are extrapolated, we are ready to calculate the next features:

- **Angle / distance between maximum and minimum points (*angp, dpb*):** The first feature, named as $angp$, refers to the value of the angle between the x-axis and the straight line passing through the minimum and the maximum current point of the V-I trajectory. The second one, named as dpb, is simply the distance between the minimum and the maximum (please, refer to the python code .11 in the Appendix A).
- **Area (*ar*):** This feature represents the area inside the V-I trajectory calculated through the Shoelace formula in Equation 5 for the calculus of polygons' areas (please, refer to the python code .19 in the Appendix A):

$$area = \frac{1}{2}|\sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - (\sum_{i=1}^{n-1} x_{i+1} y_i + x_1 y_n)|. \tag{5}$$

From a coding point of view, the area *ar* of V-I trajectories is calculated as:

$$ar = \sum_i \frac{1}{2}|V_j - V_i|(|I_{i'} - I_i| + |I_{j'} - I_j|)\ i \in [1,...,vmin - 1] \tag{6}$$

where *i'* and *j'* denote the points on part B for which the voltage is closest to the two consecutive points *i* and *j*, respectively, and they are calculated as shown in Equation 7 and 8.

$$\underset{i'}{\operatorname{argmin}} |V_{i'} - V_i|\ i'\ \in (vmin + 1,...,N), \tag{7}$$

$$\underset{j'}{\operatorname{argmin}} |V_{j'} - V_j|\ j'\ \in (vmin + 1,...,N). \tag{8}$$

- **Current Span (*itc*):** This feature, named itc, calculates the value of the current span as $itc = I_{MAX} - I_{MIN}$. Since calculating this feature over just one cycle of the steady state will make it very selective, it is preferable to calculate itc as the difference between the average of the local maxima and the one of local minima inside the steady-state interval (please, refer to the python code .11 in the Appendix A).
- **Length (*L*):** This feature named len calculates the length of the V-I trajectory, obtained summing the distances between consecutive points (please, refer to the python code .15 in the Appendix A).
- **Area with loop direction (*lpa*):** This feature named lpa measures, differently to $area$, the value of the integrated trajectory. Therefore $lpa > 0$ means that the direction of the loop is clockwise, and counterclockwise if $lpa < 0$. The code is the interpretation of the following expression (WANG *et al.*, 2018a) (please, refer to the python code .13 in the Appendix A):

$$lpa = \sum_{u} \frac{1}{2}(V_{u+1} - V_u)(I_{u+1} - I_u),\ \ u = 0,1,\ldots,\ \text{length(signal)} - 1. \tag{9}$$

- **Maximum Distance (*md*):** Named md, calculates the maximum distance from a point in the trajectory and the origin (please, refer to the python code .16 in the Appendix A).
- **Asymmetry (*asy*):** This feature is used to indicate whether the current conduction of the appliance is the same between positive and negative voltage waves. Rotate the trajectory $180°$ around its own symmetry center: trajectory asymmetry is defined by the Hausdorff distance between the rotated trajectory and the original trajectory (please, refer to the python code .14 in the Appendix A).
- **Curvature of mean line (*r*):** This feature characterizes the non-linearity of the appliance. In this feature we first calculate the mean line of VI trajectory as:

$$M = (x_{M,i}, y_{M,i}) | x_{M,i} = \frac{V_i + V_{i'}}{2}, y_{M,i} = \frac{I_i + I_{i'}}{2}\ i \in (1,2,\ldots,vmin). \tag{10}$$

Equation 10 shows that average voltage and current points *i* and *i'* are taken as the $x_{M,i}$ and $y_{M,i}$, respectively, which refer to the coordinate of point *i* in *M*. The mean line of the trajectory is composed of the points in M and its curvature is defined by the linear correlation coefficient of all points in *M* (please, refer to the python code .17 in the Appendix A).

- **Self-intersection (*sc*):** This feature is related to high-order harmonic characteristic of the appliance. To determine whether there is an intersection or not, Equation 11 was used (please, refer to the python code .18 in the Appendix A):

$$((\overrightarrow{ij}) \times (\overrightarrow{ii'}) \cdot (\overrightarrow{ij}) \times (\overrightarrow{ij'})) < 0. \tag{11}$$

3.3 CLASSIFIERS

A classifier is the core of every machine learning project, since it is the element that enables learning and the capacity of a machine to analyze data and suddenly make choices. It is, therefore, always stressed out how the selection of a good classifier is very important for our system's performances, that are usually evaluated by means of some metrics whose values can varies from classifier to classifier.

Two are the macro-categories when we are talking about machine learning (RASCHKA; MIRJALILI, 2019). *Supervised learning* aims at learning a model from labeled training data that allows us to make predictions about unseen future data. On the other hand, *unsupervised learning* usually tries to cluster data, approaching the classification problem with little or no idea on what our true labels should look like.

In this work a supervised approach will be used, and that is why datasets were analyzed and new ones were built. The approach adopted is intuitively deeply connected to our data, that may differ in the number of features or samples, the amount of noise or whether the classes are linearly separable or not. That is why it is important to evaluate a bunch of classifiers before deciding what really suits our problem.

Here in this section, the theory behind the four classifiers used for the project will be discussed, in a way to have an idea of their working principles. In Chapter 4, the tuning process of those classifiers will be shown, followed by the evaluation process discussed in Chapter 5.

3.3.1 Logistic Regression

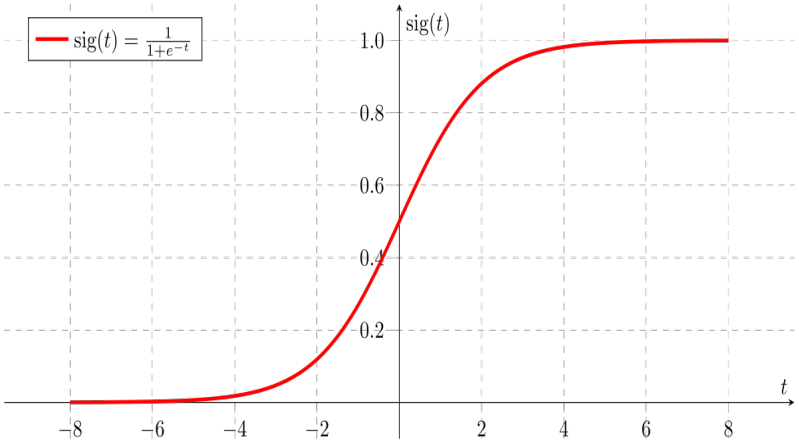
Logistic regression is a classification model that is easy to implement and performs very well on linearly separable classes (RASCHKA; MIRJALILI, 2019). For this algorithm, z is the net input and is calculated as a linear combination of weights w and sample features x. Supposing that $x \in \mathbb{R}^m$ yields:

$$z = w^t x = w_0 x_0 + w_1 x_1 + ... + w_m x_m. \tag{12}$$

The basic idea of logistic regression is to feed the input into a function called Sigmoid function $\phi(z)$ that enables to predict the probability that a particular sample belongs to a particular class (See Figures 4 and 5)).

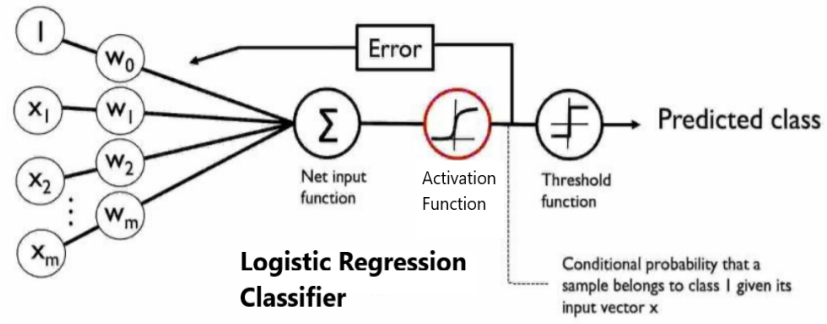
$$\phi(z) = \frac{1}{1 + e^{-z}}. \tag{13}$$

Figure 4 – Graphical representation of the Sigmoid function. Adapted from (RASCHKA; MIRJALILI, 2019).



Source: (RASCHKA; MIRJALILI, 2019)

Figure 5 – Schematic of Logistic Regression. Adapted from (RASCHKA; MIRJALILI, 2019).



Source: (RASCHKA; MIRJALILI, 2019)

Calling $\hat{y}$ the prediction made from the classifier, from the graph of the Sigmoid function comes the intuition that:

$$\hat{y} = \begin{cases} 1, \text{ if } z \geq 0, \\ 0 \; otherwise. \end{cases} \tag{14}$$

Note that, even though this model seems to apply just to binary classification, it can be extended to multiclass classification via the One-versus-Rest technique (RASCHKA; MIRJALILI, 2019). Since we are using *supervised learning*, the classifier model has to be fitted with the training data. This process will iteratively update the weight of a cost function $J(w)$ in a way to minimize its value. The cost function can be defined, for logistic regression, as follows:

$$J(w) = -\sum_{i} y^{(i)} log(\phi(z^{(i)}) + (1 - y^{(i)}) log(1 - \phi(z^{(i)}). \tag{15}$$

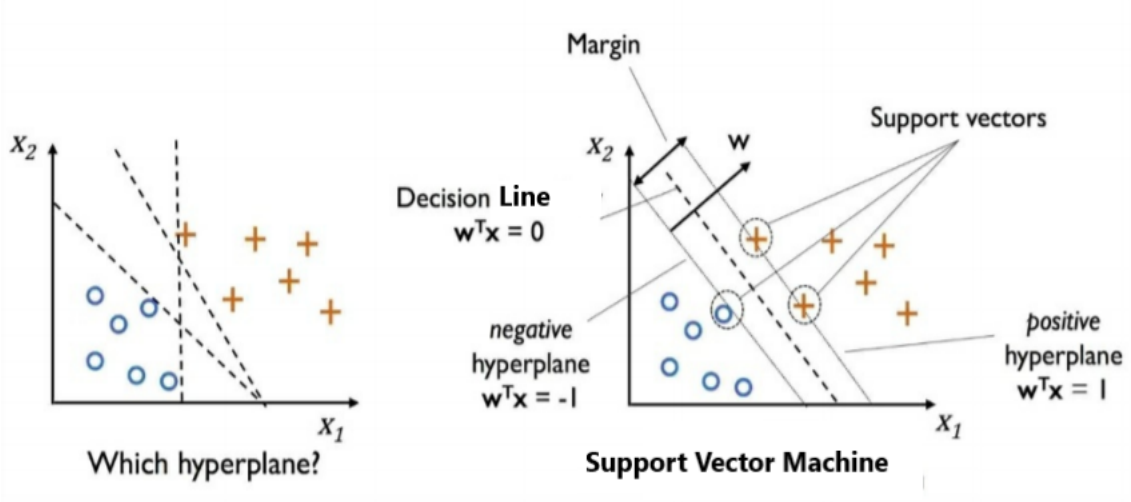
The weights are updated in each iteration over a new training sample with a technique called gradient descent:

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = \eta \sum_{i=1}^{n} (y^{(i)} - \phi(z^{(i)})) x_j^{(i)}. \tag{16}$$

3.3.2 Support Vector Machine

This classifier aims at maximizing a margin defined as the distance between the separating hyperplane (decision boundary) and the training samples that are closest to this hyperplane, which are the so called support vectors.

Figure 6 – Graphical interpretation of SVM. Adapted from (RASCHKA; MIRJALILI, 2019)



Source: (RASCHKA; MIRJALILI, 2019)

The margin that is maximized during training (Figure 6), under the constraint that samples are classified correctly, is represented by the left side of the following equation (RASCHKA; MIRJALILI, 2019):

$$\frac{w^t(x_{pos} - x_{neg})}{||w||} = \frac{2}{||w||}, \tag{17}$$

where x_{pos} and x_{neg} come respectively from the positive and negative hyperplanes that are parallel to the decision boundary.

An interesting characteristic of the Support Vector Machine (SVM) classifier is that it works very well even with the presence of non-linearly separable data. This for mainly two reasons:

1. A slack variable was introduced in Scholkopf *et al.* (1997) allowing to relax linear constraint and control the penalty of misclassification (this will be better describe in Chapter 5 during the evaluation process);
2. SVM classifier can exploit the kernel trick to find separating hyperplanes in high-dimensional space.

In particular, the latter enables us to transform the training data onto a higher dimensional feature space via a mapping function ψ and train a linear SVM model to classify the data in this new feature space. This could be achieved because a similarity function enables to ensure the linearity of the SVM model in the higher dimensional space, leading to a very powerful transformation with excellent

performances in terms of computational costs. One of the most widely used kernels is the Radial Basis Function (RBF) kernel, usually called Gaussian kernel:

$$\mathcal{K}(x^{(i)},x^{(j)}) = exp(-\frac{||x^{(i)} - x^{(j)}||^2}{2\sigma^2}), \tag{18}$$

where $\mathcal{K}(x^{(i)},x^{(j)}) = \psi(x^{(i)})^t\psi(x^{(j)})$ is the kernel function.

3.3.3 Random Forest

Random forest is another type of classifier which basically can be considered as an ensemble of decision trees (PAL, 2005). A decision tree is a classifier that elaborates a sort of map composed of multiple branches and multiple nodes. Metaphorically speaking, a data will go through this map (or tree) where each node correspond to a question, and whose result enable the classifier to select the next node the data will be forward to. This process repeats until one of the so called leaf nodes is reached, moment in which the classifier made its prediction.

Mathematically speaking, decision tree splits data over the features that results in the largest Information Gain (IG) with an iterative process to go down the tree and reach the leaf nodes. The information gain is defined as follows:

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j), \tag{19}$$

where f is the feature to perform the split, D_p and D_j are the dataset of the parent and jth child node, I is the impurity measure, N_p is the total number of samples at the parent node, and N_j is the number of samples in the jth child node.

However, to reduce computational complexity and the combinatorial search space, usually binary decision trees are used, meaning that each parent node is split into two child nodes as follows:

$$IG(D_p, f) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right}). \tag{20}$$

Now, the three impurity measures or splitting criteria that are commonly used in binary decision trees are Gini impurity (I_G), entropy (I_H) and the classification error (I_E).

To conclude, the idea behind Random Forest is to average multiple decision trees, that individually suffer from high variance, to build a more robust model that has a better generalization and is less susceptible to over-fitting. The random forest algorithm can be summarized in four simple steps:

1. Draw a random bootstrap sample of size n (randomly choose n samples from the training set with replacement).
2. Grow a decision tree from the bootstrap sample. At each node:
    - Randomly select d features without replacement.

- Split the nodes using the features that provides the best split according to the objective function, for instance, maximizing the information gain.

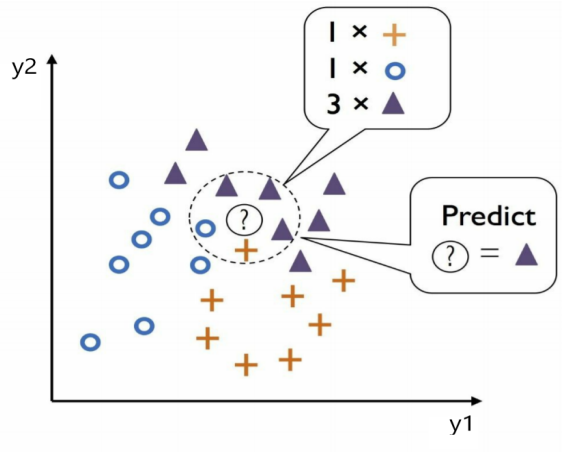
3. Repeat the steps 1-2 k times.
4. Aggregate the prediction by each tree to assign the class label by majority voting.

3.3.4 K-Nearest-Neighbors

The last supervised learning algorithm is the k-nearest-neighbor (KNN) classifier. KNN classifier operates in a very simple fashion following three steps:

1. Choose the number of k and a distance metric;
2. Find the k-nearest neighbors of the sample that we want to classify;
3. Assign the class label by majority vote.

Figure 7 – Working principle of KNN. Adapted from (RASCHKA; MIRJALILI, 2019)



Source: (RASCHKA; MIRJALILI, 2019)

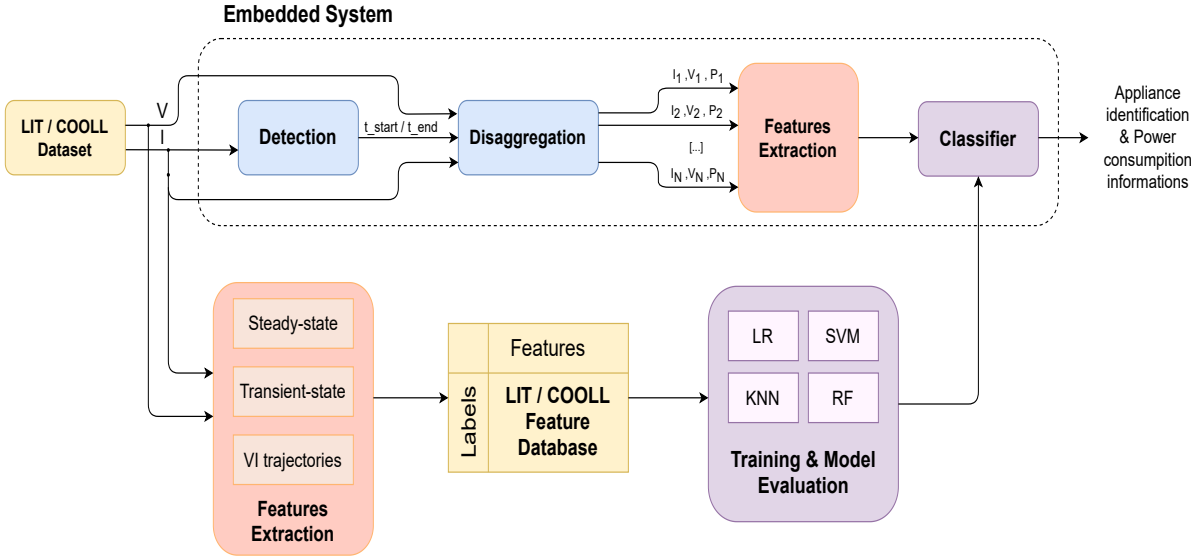
Based on the chosen distance metric, the KNN algorithm finds the k samples in the training dataset that are closest to the point that we want to classify – for example the element (?) in Figure 7. The class label of the new data point is then determined by a majority vote among its k nearest neighbors.

The reason why this type of classifier is different than the ones of the previous sections is that it does not learn a discriminative function from the training data, but instead memorizes the training dataset. The main advantage of such a memory-based approach is that the classifier immediately adapts as we collect new training data.

4 METHODS

In this chapter, the implementation of the whole project is presented. The sketch of the model that has been developed for this work is displayed in Figure 8.

Figure 8 – Project schematic.



Source: Own Autorship

The NILM algorithm proposed in this work has to analyze current and voltage waveforms characterized by the presence of multiple appliances, with the respective turn-on and off events. Therefore, the proposed method has to be capable of detecting those events, disaggregate appliances and recognize them. To do so, machine learning techniques have been applied to train models for load classification. In particular, the project's workflow used two publicly available dataset of load waveforms to optimize and evaluate a possible prototype of a NILM algorithm. This procedure is represented on the lower branch of Figure 8.

Moreover, the algorithm was finally validated in laboratory with some measurements from real appliances. This procedure consisted in two phases:

1. collect waveforms from different appliances and build a new feature dataset for classifiers' training and model evaluation;
2. perform new measurements for testing the algorithm performances.

Laboratory measurements were useful to study the feasibility of the project, since this allowed to study NILM in the perspective of its final implementation, enabling to understand better what should be improved or changed. In the next paragraphs, we detail each of the blocks presented in Figure 8 and explain their function or utility.

4.1 DETECTION METHOD AND TUNING

In this work, all the current and voltage signal were already registered in the respective datasets (RENAUX *et al.*, 2018b) and (PICON *et al.*, 2016). Even in a real life application of the project, the same approach would be used: a finite time window from signal of the monitored system is extrapolated, then appliances detection, extrapolation and classification is performed. It is important to emphasize that, during this work, it was taken into account the assumption that transients of signals always appear in sequence.

4.1.1 Detector Tuning

In Chapter 3, the detector working principle and pseudo-codes implementation were discussed. However, the latter was not completely automatize since every time we had to manually choose the threshold value, used for transient detection. Since the whole project has to be a plug-in device able to perform NILM, here two ways of automatically setting a threshold are presented.

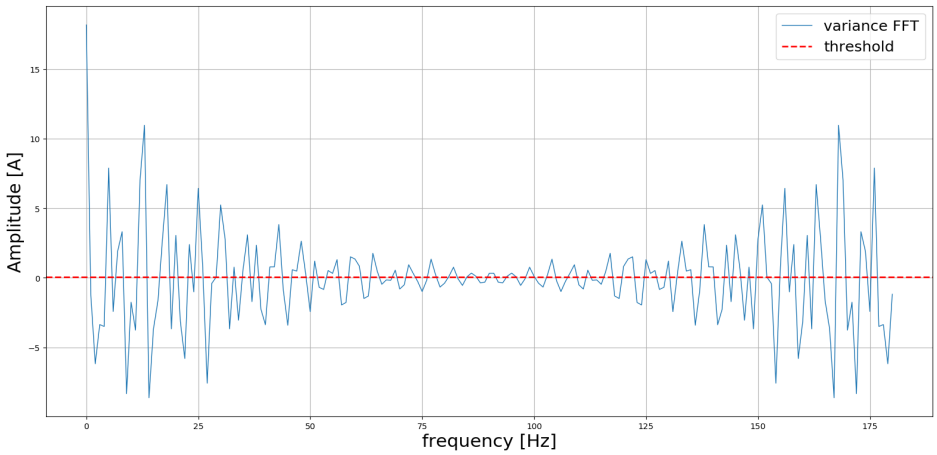
4.1.1.1 FFT-based threshold

A first way to calculate an adaptive threshold could be considering the Fourier transform of the signal envelope variance (remember that in our case the envelope is taken from the current signal). The intuition behind the calculation is that, for our cases, the variance shape will always follow the same pattern: high values intervals (corresponding to the transients) alternated with the presence of very low values intervals (corresponding to the steady-states).

Hence, the Fourier transform will be characterized by a set of high frequency components with high amplitude, while the remaining frequencies will have a low amplitude. By taking the absolute value of the FFT and computing its mean value, an adaptive threshold could be directly calculate inside the detector algorithm without passing it as a parameter (please, refer to the python code .20 in the Appendix A).

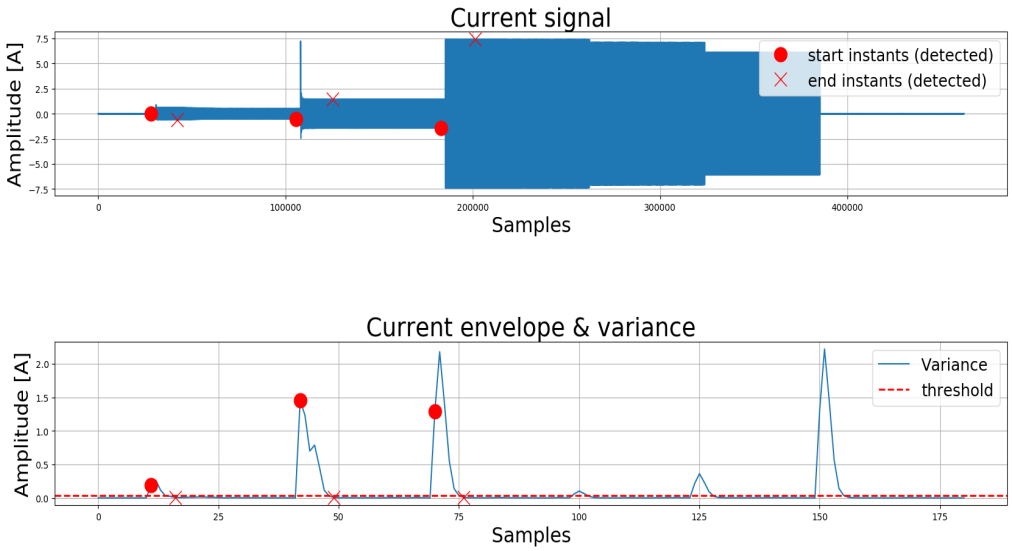
The signal analyzed (Figure 9) comes in this case from the LIT dataset and corresponds to an aggregate signal with three loads connected. However, as shown in Figure 10, the threshold might appear to be too low: in this way even some noise in the signal could be wrongly detected as a start/end instant of a transient. Smoothing the signal in this case will not be a good solution because, even though noise is reduced, the threshold will be reduced as well. A way to reduce wrong detections can then be increasing the threshold, done in this work by simply multiplying it by the F parameter already present in the detector algorithm (Figure 11).

Figure 9 – Fourier transform of the variance for the 3E0N0Q0 signal in LIT Dataset



Source: Own Autorship

Figure 10 – 3E0N0Q0 transients detection with FFT-based threshold.



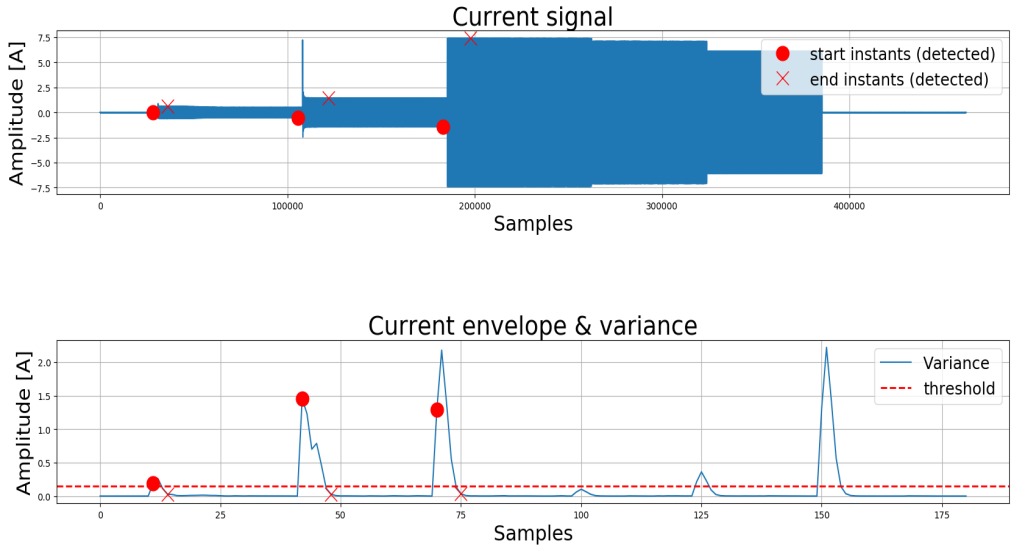
Source: Own Autorship

While increasing the threshold yields good results in this case, further simulations show that even in this case, we will face another trade-off, since the higher is the number of appliances in the same signal, the higher the threshold will be, and so some low power signals may not be detected.

4.1.1.2 Local Peak-based Threshold

A second approach to find an adaptive threshold for the transient detection could be analyze the variance in a way to find its local peaks. Once those are founded, we can simply average their values to

Figure 11 – 3E0N0Q0 transients detection with FFT-based threshold augmented by a factor of F.



Source: Own Autorship

obtain the value of the threshold (please, refer to the python code .21 in the Appendix A).

The intuition behind this method is that we know that, as said in the previous section, the variance signal shape always follows the same pattern alternating high values intervals with ones with very small values.

We know that the presence of noise in the signal might confuse our detector by giving some false positives. However, noise will be likely a low amplitude component compared to the peaks relative to transient intervals. Therefore, the solution proposed in this section avoids wrong transient detection due to the possible presence of noise.

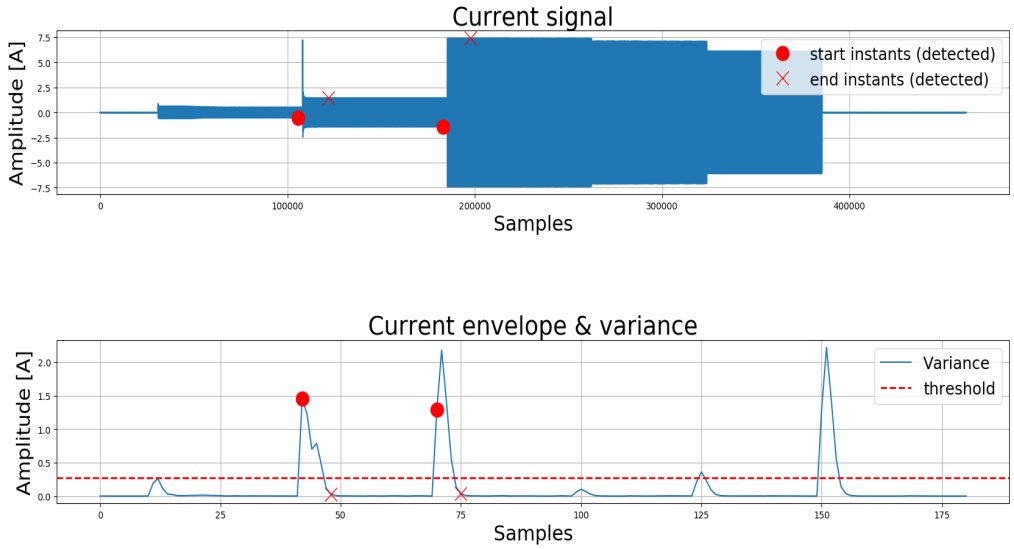
Of course we can suddenly observe that this solution might come out with a threshold too much high and, thus, some low amplitude signals may not be detected. Figure 12 shows, for example, that the first appliance has not been detected. One solution in this case could be decrease the threshold by a factor F (Figure 13) already present in the detector algorithm.

However, even this kind of approach shows the same trade-off discussed for the FFT-based threshold, thus a very precise tuning is necessary.

4.1.2 Disaggregation

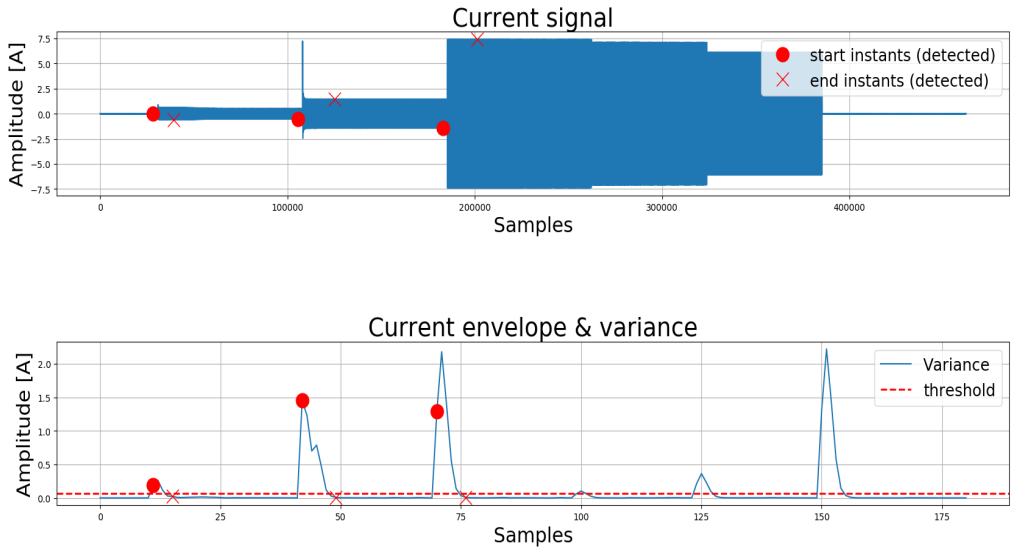
While for the COOLL dataset, only waveforms referring to one single appliance are registered, it is not the same for the LIT dataset, where waveforms with 1, 2, 3, or even 8 appliances connected to the same system are present. This actually represents a more realistic scenario, since NILM should be

Figure 12 – 3E0N0Q0 transients with local peaks-based threshold.



Source: Own Autorship

Figure 13 – 3E0N0Q0 transients with local peaks-based threshold decreased by a factor F.



Source: Own Autorship

performed over systems in which multiple devices are connected and need to be monitored. In this case, to perform a good classification, isolating the various appliances from the whole signal analyzed must be done.

The disaggregation process employed in this project works very simply and accomplish the following steps, as suggested in Wang *et al.* (2018b):

1. Isolate one transient through the respective start and end instants provided by the detector;

2. Buffer a part of the signal ($c - pre$) right before the transient and with its same length;
3. Buffer as well a part of the signal right after the transient and with its same length for the calculation of steady state features;
4. Align both the transient and the steady-state intervals with the signal $c - pre$;
5. Subtract $c - pre$ from both the signals, in a way to isolate the appliance that has to be analyzed by cleaning it from the components of the previous appliances;
6. Smooth the signal to compensate errors from the subtraction operation.

In particular, the algorithm could be written as follows:

FUNCTION **DISAGGREGATION** (*on-instant, off-instant,current-signal, voltage-signal*)

length $\leftarrow$ (*off-instant - on-instant*)

c-transient $\leftarrow$ *current-signal*[*on-instant* : *off-instant*]

v-transient $\leftarrow$ *voltage-signal*[*on-instant* : *off-instant*]

c-steady $\leftarrow$ *current-signal*[*off-instant* : *off-instant* + *length*]

v-steady $\leftarrow$ *voltage-signal*[*off-instant* : *off-instant* + *length*]

c-pre $\leftarrow$ *current-signal*[*on-instant* $-$ *length* : *on-instant*]

$\theta_t \leftarrow$ Align(*c-transient, c-pre*)

$\theta_s \leftarrow$ Align(*c-steady, c-pre*)

c-transient $\leftarrow$ *current-signal*[*on-instant* $+ \theta_t$: *off-instant* $+ \theta_s$] $-$ *c-pre*

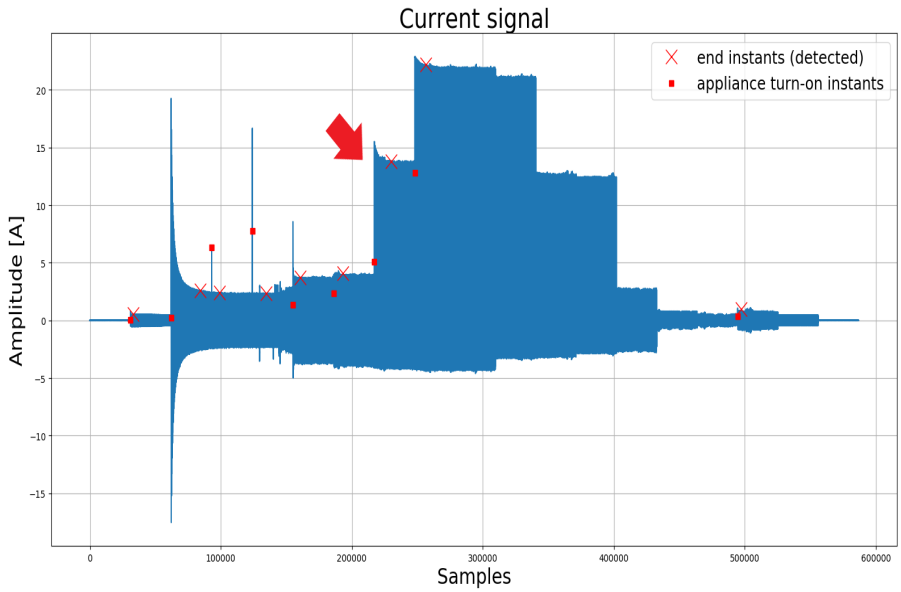
c-steady $\leftarrow$ *current-signal*[*off-instant* $+ \theta_s$: *off-instant* + *length* $+ \theta_s$] $-$ *c-pre*

Return *c-steady, c-transient, v-steady, v-transient*

The function returns the transient and steady-state intervals isolated from other signals components. Single appliances are isolated, as presented in point 5, subtracting them from the previous part of the whole signal. This is actually not the most appropriate approach, because implies a linear simplification that obviously is not correct for appliances. However, even though not theoretically correct, the simplification presents relevant classification results, as presented in Mulinari *et al.* (2019) and Ancelmo *et al.* (2019).

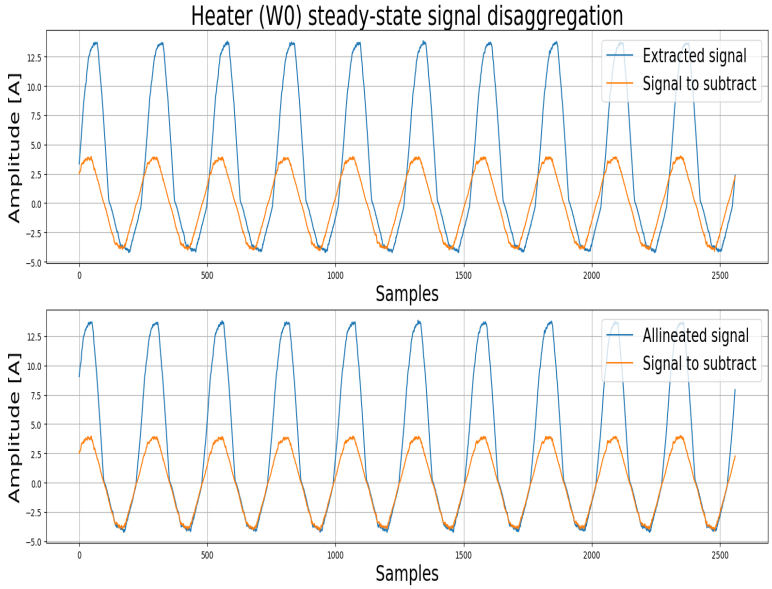
In Figures 14, 15 and 16, the performance of the function implemented in this project is shown (please, see the python code .31 and its explanation in the Appendix A).

Figure 14 – LIT dataset: the signal pointed with the red arrow will be now extrapolated.



Source: Own Autorship

Figure 15 – In this case $part =' s'$, thus a W0's steady state interval of n_cycles has been extrapolated

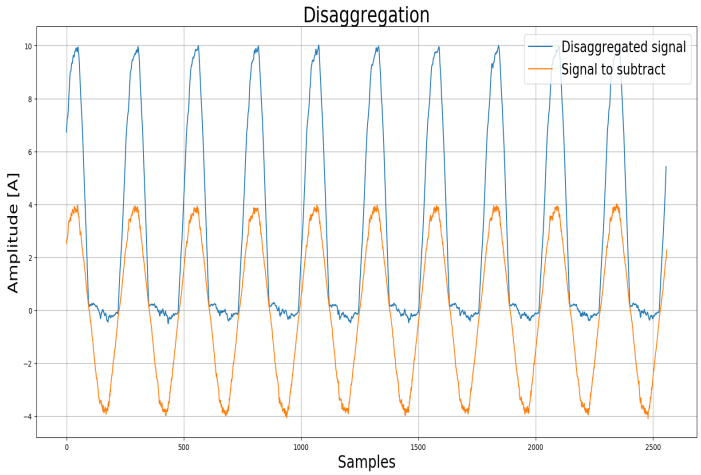


Source: Own Autorship

4.2 DATASETS

In this section, the datasets used will be described. One of them is a reduced dataset, conceived to validate the NILM algorithm with real appliance measurements. On the other hand, the initial tests required a more ideal environment, therefore two publicly available datasets were use, i.e.:

Figure 16 – W0, after being extrapolated and aligned, is subtracted by a factor of $temp$ to be isolated from other appliances



Source: Own Autorship

- COOLL Dataset: https://coolldataset.github.io/
- LIT Dataset: http://dainf.ct.utfpr.edu.br/~douglas/LIT_Dataset/index.html

4.2.1 COOLL Dataset

The Controlled On/Off Loads Library (COOLL) is a dataset of high-sampled electrical current and voltage measurements representing individual appliances consumption (PICON *et al.*, 2016). The measurements were taken on June 2016 at the PRISME laboratory of the University of Orléans, France. The appliances are mainly controllable appliances (meaning that we can precisely control their turn-on/off time instants). In total, 42 appliances of 12 types were measured at a 100 kHz sampling frequency. All the loads of COOLL Dataset are collected in Table 1.

Table 1 – COOLL Dataset

N°	Appliance type	Number of appliances	Number of current signals (20 per appliance)
1	Drill	6	120
2	Fan	2	40
3	Grinder	2	40
4	Hair dryer	4	80
5	Hedge trimmer	3	60
6	Lamp	4	80
7	Paint stripper	1	20
8	Planer	1	20
9	Router	1	20
10	Sander	3	60
11	Saw	8	160
12	Vacuum cleaner	7	140

Source: (PICON *et al.*, 2016)

The appliances in the COOLL dataset are measured individually, one at a time. The dataset does not contain a scenario where several appliances are measured simultaneously. Moreover, the selected appliances are chosen so that the control of the turn-on time instant is possible, thus by pressing the switch-on button beforehand, the appliance can be operated electronically (with controlled triacs). Each measurement lasts 6 seconds with a pre-trigger duration of 0.5 second (the pre-trigger of the first few measurements being different and equal to 1 second) and a post-stop duration of 1 second. These durations correspond, respectively, to the time where the appliance is off before the turn-on and after the turn-off.

For each appliance, 20 controlled measurements are made, for a total of 840 waveform for both current and voltage signals. Each measurement corresponds to a specific action delay ranging from 0 to 19 ms with a step of 1 ms. This way, it is possible to cover the whole time-cycle duration of the 50 Hz mains voltage which is 20 ms. This action delay corresponds to the time with which the turn-on action is delayed with respect to the beginning of a specific time-cycle of the mains voltage. The turn-off action delay is fixed to 0 ms for all measurements.

Since this dataset contains just single appliance waveform, it was considered more suitable for the first simulations of the algorithm. As a matter of fact, in this scenario we can, for instance, use an algorithm which analyze a waveform, calculate some features and send them to a classifier. This allowed to have a first evaluation of the algorithm without the possible bias given by the introduction of the detection and disaggregation algorithms.

In the dataset all the waveforms are provided as $.flac$ files, where FLAC (Free Lossless Audio Codec) is an audio format similar to MP3, but lossless, meaning that audio is compressed in FLAC without any loss in quality. Moreover the waveforms are all normalized, meaning that once extracted they will have to be multiplied by the respective scale factor (already provided in the dataset specifications). Please, see .22 in Appendix A for the python code.

4.2.2 LIT Dataset

The LIT Dataset was conceived and engineered to provide data for evaluation of NILM Systems. The development process behind the LIT Dataset started by the evaluation of exiting datasets, comparing their features and then by stating the requirements of the recorded waveforms of voltage and current when single and multiple loads were monitored under several conditions. In particular, this dataset is composed of three distinct classes of load monitoring: Synthetic, Simulated, and Natural.

For this work, the Synthetic dataset was used, characterized by waveforms collected with a jig that precisely controls the on and off times in a scenario of controlled load shaping. The jig controls each individual load, up to eight, using TRIACs and relays, hence, allowing for on-times at specified angles of the mains sine wave. The on and off times are recorded in the waveform with a resolution better that 5 ms,

allowing the identification of the mains semi-cycle where the on or off load-event occurred. Typically, waveforms are 30 seconds long at sampling rates above 15 KHz, achieving 256 samples per cycle. All the appliances of LIT Dataset are presented in Table 2.

Table 2 – LIT Dataset

ID	Load	Model	Brand	Avg. Power
A0	Microwave Oven Standby	CMS18BBHNA	Consul	4.5 W
B0	Led Lamp	TKL 06	Taschibra	6 W
C0	CRT Monitor	CPD-17SF1	Sony	10 W
D0	Led Pannel	DL500	Citiaqua	13 W
E0	Fume Extractor	TS-153	Toyo	23 W
F0	Led Monitor	DL-500	AOC	26 W
G0	Asus Phone Charger	AD2037020	Asus	38 W
H0	Soldering Station	WLC100	Weller	40 W
I0	Motorola Phone Charger	SA-A390M	Motorola	50 W
J0	Lenovo Laptop	LS-PAB70	Lenovo	70 W
K0	Fan	V-45	Mondial	80 W
L0	Resistor	100R 10 %	Ohmtec	80 W
M0	Vaio Laptop	PCG-61112L	Sony	90 W
N0	Incandescent Lamp	Centra A CL 100	Osram	100 W
O0	Drill Speed 1	Impact Drill (0.46hp)	Bosch	165 W
P0	Drill Speed 2	Impact Drill (0.46hp)	Bosch	350 W
Q0	Oil Heater Power 1	NYLA-7	Pelonis	520 W
R0	Oil Heater Power 2	NYLA-7	Pelonis	750 W
S0	Microwave Oven On	CMS18BBHNA	Consul	950 W
T0	Nilko Air Heater	KN565	Nilko	1120 W
U0	Hair Dryer Eleganza - Fan 1	Eleganza 2200	GAMA Italy	365 W
V0	Hair Dryer Eleganza - Fan 2	Eleganza 2200	GAMA Italy	500 W
W0	Hair Dryer Super 4.0 - Heater 1	SL-S04	Super 4.0	660 W
X0	Hair Dryer Super 4.0 - Heater 2	SL-S04	Super 4.0	1120 W
Y0	Hair Dryer Parlux - Fan 1	Advance	Parlux	660 W
Z0	Hair Dryer Parlux - Fan 2	Advance	Parlux	660 W

Source: (RENAUX *et al.*, 2018b)

The LIT Synthetic dataset provides, in particular, four type of waveforms:

- 26 groups of 16 waveforms, each group measuring 1 single appliance.
- 42 groups of 16 waveforms, each group measuring 2 appliances.
- 33 groups of 16 waveforms, each group measuring 3 appliances.
- 6 groups of 16 waveforms, each group measuring 8 appliances.

The fundamental difference between COOLL and LIT dataset is the presence of aggregated waveforms in the latter, which allows to reproduce a more real scenario where more appliances could switch on together. In this case, simulations required the use of a detection and disaggregation algorithm as mentioned in the sections above. Another important characteristic is that, for every waveform, turn-on and turn-off events are registered, meaning that we can use them to evaluate the performance and the accuracy of the detection algorithm before going on with real measurements.

Each waveform is provided as a $.mat$ file that includes:

- iShunt: A vector containing the current signal collected by shunt sensor;
- iHall: A vector containing the current signal acquired by hall sensor;
- vGrid: A vector containing the voltage grid signal acquired;
- events: A numbered vector with the turn-on and turn-off events information. The value '1' is set at the exact sample of a turn-on event and its complementary '-1' is set for a turn-off event. All the other samples are set with a null value;
- labels: A string valued vector to correlate each turn-on and turn-off event indicated at the *i* events and *i* vector with the correspondent load ID;
- duration: This is a single integer to identify the acquisition duration in seconds.

For this work the $iShunt$ and $vGrid$ signals where used as a current and voltage signal; moreover the events vector was used from the modified detector to know the starting instants of the appliances in each waveform (to use a $.mat$ file in python, please see .23 in Appendix A).

4.2.3 Lab Dataset

This small dataset was conceived for a final validation of the whole algorithm over a real task of measuring and analyzing appliance waveforms. The whole dataset was built in May-June 2021 in the MElting Lab (Industrial Automation and Power Electronics Laboratory) of the engineering department Enzo Ferrari of the University of Modena and Reggio Emilia. With the help of a Tektronix TBS2104 oscilloscope, we performed the measurements of the 5 appliances in Table 3.

Table 3 – Lab Dataset

N°	Label	Load	Appliance type
1	LA0	Coffee Machine	Caffitaly Capsule Machine
2	LB0	PC Off Charge	Lenovo ideapad 330S-15IKB
3	LC0	PC On Charge	Lenovo ideapad 330S-15IKB
4	LD0	Solder On Sleep	Soldering Station JBC CD-25QE
5	LF0	Solder On Start	Soldering Station JBC CD-25QE

Source: Own Authorship

Waveforms here are provided as interval of 10 seconds at a sampling rate of 2.5 kHz and they correspond to current measurements obtained with the TCP0030A CAL Tektronix current probe, which provides greater than 120 MHz of bandwidth with selectable 5 A and 30 A measurement ranges. It also provides good low-current measurement capability and accuracy to current levels as low as 1 mA. However, no voltage measurements were done and the corresponding signals were virtually created by means of an algorithm that creates a 220V, 50Hz signal and aligns it with the current one. In this way we ensure the calculus of features such as active, reactive, apparent power, or VI trajectories. In particular, Lab Dataset provides three type of waveforms:

- 5 groups of 10 waveforms each measuring 1 single appliance.
- 7 groups of 10 waveforms each measuring 2 appliances.
- 2 groups of 10 waveforms each measuring 3 appliances.

All those waveforms were used to test and validate the NILM algorithm written and optimally tuned to work with LIT and COOLL datasets. In this case, no turn-on and turn-off instance were recorded and this represents a real scenario where we are required to apply a detection algorithm to spot whether there is an appliance or not.

4.3 TRAINING AND MODEL EVALUATION

In Chapter 3, the way features were extracted from signals was shown. Since a supervised learning approach has been used, every voltage and current signals from the COOLL, LIT and Lab datasets were analyzed in order to extract and register the corresponding features, thus building three datasets collecting those results labeled with the respective appliance.

The next step in the sequence will be training a classifier. *Training* means that a classifier is fitted with the data of a dataset and compiled, in a way to update and adapt its inner structure to the data we want to classify. Doing the training process, two are the most common problems that are worth to tackle, thus having a more robust classifier:

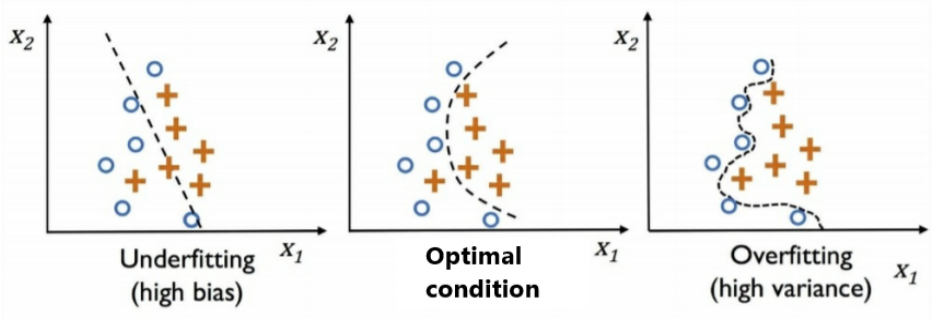
- Overfitting, that is when a model performs very well on training data but does not generalize well to test data.
- Underfitting, which means that our model is not complex enough to capture well the patterns in the training data and therefore suffers of low performance on unseen data (see Figure 17).

In particular, overfitting is caused by high variance, where the *variance* measures the consistency of the model prediction for a particular sample instance if we were to re-train the model multiple times, for example, on different subsets of the training dataset. On the other hand, underfitting is caused by high bias, where the *bias* measures how far off the predictions are from the correct values in general if we re-build the model multiple times on different training datasets.

The trained process turned out to be the most computational expensive part of this work, since lot of calculus for classifier training and model evaluation are required. In fact, before saving the model of the classifier, some previous analysis are usually execute to increase its performances.

The whole training passage was done on Google Colab, which is an online platform that allows to write and execute arbitrary python code through the browser, and is specially well suited for machine learning, data analysis and education, giving the possibility of exploiting the power of Google servers. In the next sections training and model evaluation techniques are explained.

Figure 17 – Graphical visualization of overfitting and underfitting. Adapted from (RASCHKA; MIRJALILI, 2019).



Source: (RASCHKA; MIRJALILI, 2019)

4.3.1 Data Pre-processing

Before being fed into a classifier, a dataset has to be pre-processed. This is because classifier training quality may depend on the dataset it has been fitted, therefore building good training sets can make the difference in the final classification performance.

First of all is important to identify missing values in tabular data, for example NaN (not a number) data, that can be related to a whole sample or a single feature. When its a whole sample which is corrupted, this one is simply eliminated from the dataset. By the way, more often just one or few features are corrupted: in this case the elements are individuated and substituted. Many ways of substituting the elements are available but, for this work, mean imputation was used, simply replacing each single NaN with the mean value of the entire feature column.

Another tip applied is class labels encoding. Many machine learning libraries require that class labels are encoded as integers values. Although most estimators for classification convert class labels to integers internally, it is considered good practice to provide class labels as integer arrays to avoid technical glitches. Then, once the labels are encoded, we can always come back to the original label's name by using, for example, a dictionary.

Finally, a training dataset is built. To do that we split the dataset into a training dataset and into a test one for model evaluation. The split procedures allows to generalize over the used dataset and to evaluate model performance for both training and test datasets by checking their prediction accuracy. The whole algorithm in python for the dataset pre-processing has been made available in the Appendix A at .27.

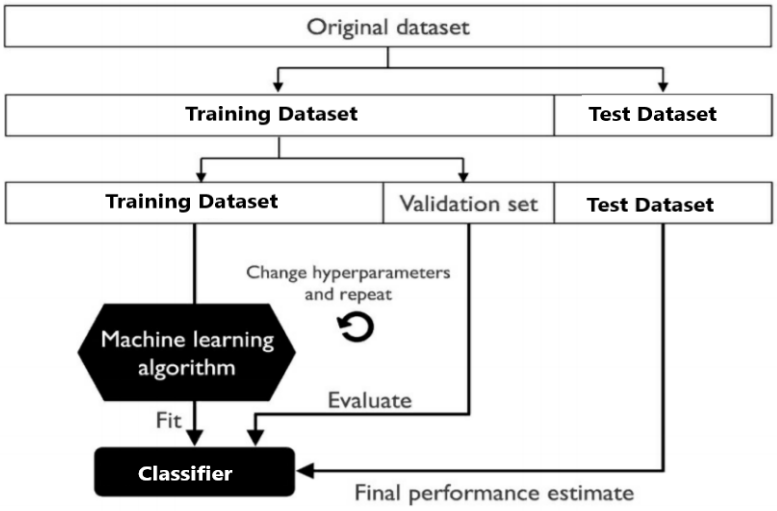
Many machine learning and optimization algorithms also require feature scaling for optimal performance. Usually standardization is used for this purpose. Standardization shifts the mean of each feature so that it is centered at zero and each feature has a standard deviation of one:

$$x'_j = \frac{x_j - \mu_j}{\sigma_j}. \tag{21}$$

This, for instance, can help gradient descent learning in logistic regression to converge more quickly (RASCHKA; MIRJALILI, 2019).

A further data splitting could be done. A *model selection* will follow the pre-processing step, where the term model selection refers to a given classification problem for which we want to select the optimal values of tuning parameters. However, if we reuse the same test dataset over and over again during model selection, it will become part of our training data and thus the model will be more likely to overfit (XU; GOODACRE, 2018). Thus usually data are separated into three parts: a training set, a validation set and a test set. Figure 18 shows how this process works.

Figure 18 – Typical way of splitting data. Adapted from (RASCHKA; MIRJALILI, 2019).



Source: (RASCHKA; MIRJALILI, 2019)

In particular, in the next sections some model selection techniques will be discussed and all of them employ a validation set, being very robust techniques for optimizing classifiers' performances.

4.3.2 Sequential Backward Selection

Sequential Backward Selection (SBS) is a feature selection technique. The reason why we need to select some features is that there might be the possibility that some features are decreasing the performance of our classifier.

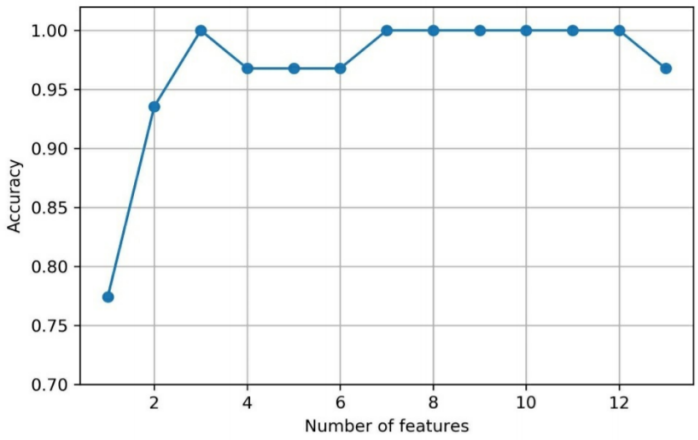
This feature selection algorithm automatically selects a subset of features that are most relevant to the problem, to improve computational efficiency or reduce the generalization error of the model by removing irrelevant features or noise (Figure 19). SBS, in particular, simply sequentially removes features from the full feature subset until the new feature subspace contains the desired number of features. The steps operated by the SBS algorithm are presented as follows (RASCHKA; MIRJALILI, 2019):

1. Initialize the algorithm with $k = d$, where d is the dimensionality of the full feature space X_d;

2. Determine the feature x^- that maximizes the criterion: $x^- = argmax J(X_k - x)$, where $x \in X_k$;
3. Remove the feature x^- from the feature set : $X_{k-1} = X_k - x^-$; $k = k - 1$;
4. Terminate if k equals the number of desired features, otherwise go to step 2.

For example, the criterion calculated by the criterion function can simply be the difference between in performance of the classifier before and after the removal of a particular feature (please, refer to the python code .24 in the Appendix A).

Figure 19 – SBS: the graph tracks the accuracy achieved by a classifier over the algorithm iterations; in this example, even with three features a very good accuracy level can be reached.



Source: (RASCHKA; MIRJALILI, 2019)

4.3.3 Principal Component Analysis

As seen in previous section, dimensionality reduction might be a good idea when build a new dataset, because we may not be sure that all features really provide discriminative information to our classifiers. Actually, there are two main categories of dimensionality reduction techniques: feature selection and feature extraction. With feature selection, we select a subset of the original features, whereas in feature extraction, we derive information from the features set to construct a new feature subspace.

In practice, feature extraction is not only used to improve storage space or the computational efficiency of the learning algorithm, but can also improve the predictive performance (NASREEN, 2014). Principal Component Analysis (PCA) is an unsupervised linear transformation for features extraction and dimensionality reduction.

PCA helps us to identify patterns in data based on the correlation between features. PCA aims to find the directions of maximum variance in high-dimensional data and projects it onto a new subspace with equal or fewer dimensions than the original one. The orthogonal axis (principal components) of the new subspace can be interpreted as the directions of maximum variance given the constraint that the new feature axes are orthogonal to each other.

In particular, PCA directions are highly sensitive to data scaling, thus we need to standardize the features before PCA if the features were measured on different scales and we want to assign equal importance to all features. The approach of PCA can be summarized in the next steps:

1. Standardize the d-dimensional dataset;
2. Construct the covariance matrix;
3. Decompose the covariance matrix into its eigenvectors and eigenvalues;
4. Sort the eigenvalues by decreasing order to rank the corresponding eigenvectors;
5. Select k eigenvectors which correspond to the k largest eigenvalues, where k is the dimensionality of the new feature subspace ($k \leq d$);
6. Construct a projection matrix W from the top k eigenvectors;
7. Transform the d-dimensional input dataset X using the projection matrix W to obtain the new k-dimensional feature subspace.

All the math behind PCA is described in details in Raschka e Mirjalili (2019) (please, refer to the python code .25 in the Appendix A). An example of the employment of PCA is shown in Figure 20.

4.3.4 Grid Search

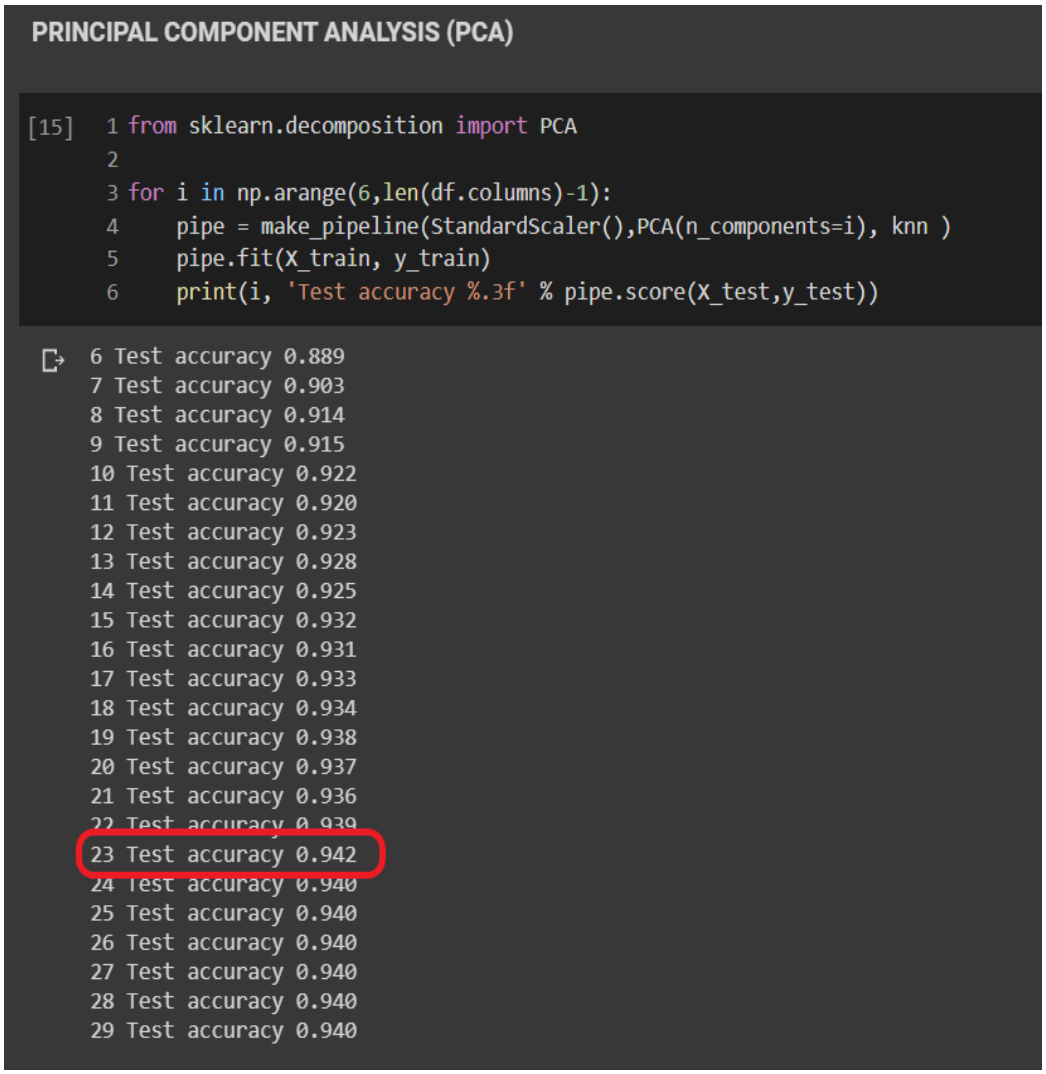
When training our classifiers, having a good dataset can make the difference in the final classification performances. However, not just the dataset determines the goodness of our final results, since lot of parameters of the classifiers themselves play a big role in the process. In this section, a technique called Grid Search for parameters tuning is discussed.

In machine learning, we have two types of parameters: those that are learned from the training data, for example, the weights in the logistic regression, and the parameters of a learning algorithm that are optimized separately. The latter are the tuning parameters, also called hyperparameters of a model, such as for example the regularization parameter in logistic regression or the depth parameter of a decision tree (RASCHKA; MIRJALILI, 2019).

Grid Search is a tuning technique that can further helps improving the performance of a model by finding the optimal combination of hyperparameters values. Its approach is very simply: Grid Search evaluates the model performance by studying the results obtained for each possible combination of the parameters given in terms of prediction accuracy (please, refer to the python code .26 in the Appendix A).

Notice that the values of the parameters we want to analyze have to be provided manually. Grid Search helps us in finding the correct tuning of the parameters, but actually those are the parameters values directly chosen by us, meaning that probably better combinations of the parameters with different values exist and are not among the ones provided.

Figure 20 – PCA – KNN model evaluation: in this case reducing the dataset to 23 features the best result is reached.



Source: Own Autorship

4.4 MODEL EVALUATION METRICS

For model evaluations, in this work, results in terms of accuracy over class predictions were always taken into account. In a measurement of a set, accuracy is the closeness of the measurements to a specific value, being a really suitable metric for classification purposes. More commonly, it is a description of systematic errors, a measure of statistical bias. Low accuracy causes a difference between a result and a true value. In our case, accuracy will be defined as follows:

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} = 1 - ERR, \tag{22}$$

where TP (FP) are the true (false) positive events and TN (FN) are the true (false) negative ones, while ERR can be understood as the sum of all false predictions divided by the number of total ones:

$$ERR = \frac{FP + FN}{FP + FN + TP + TN}. \tag{23}$$

In particular, the techniques used for evaluation purposes base their results on the accuracy. Validation curve, for example, is a tool that can be implemented to observe how the train and test datasets behave as a parameter of the classifier is varied. This method is useful when we want to address over and under fitting.

Another tool is the learning curve, which again helps to spot over and under fitting by looking at the performances of train and test datasets, this time as we vary the number of samples that belong to them (RASCHKA; MIRJALILI, 2019). It is therefore a tool to find out how much a machine model benefits from adding more training data and whether the estimator suffers more from a variance error or a bias error.

However, base the results just on one single metric is not a good approach at all, since we might not focus on characteristic of the dataset which are impacting in some way the final performance of the classifier. For example, suppose that an unbalanced dataset is being analyzed. Usually this kind of datasets are characterized by classes provided by a large number of samples and others with just a few number of them. In this case, a large number of true positive events in the former might overshadow bad classification results for the classes with few samples, leading to a high model accuracy even though some misclassifications occur.

For this reason, the Receiver Operating Characteristics (ROC) was introduced (FAWCETT, 2006). This technique is used to evaluate the dataset, measuring how finely the classes are separated among each other by looking at the true positive and false positive rate. Therefore, this method allows to analyze the quality of the dataset we are working with, which of course will has a direct impact over the performances of the classifiers.

4.5 EMBEDDED SYSTEM

The whole project was first conceived in python with the help of Spyder (RAYBAUT, 2009) editor and Google Colab (LLC, 2014) . However, the idea was to implement the whole project in an embedded system, in a way to have a device ready to be plugged in some system to operate load monitoring.

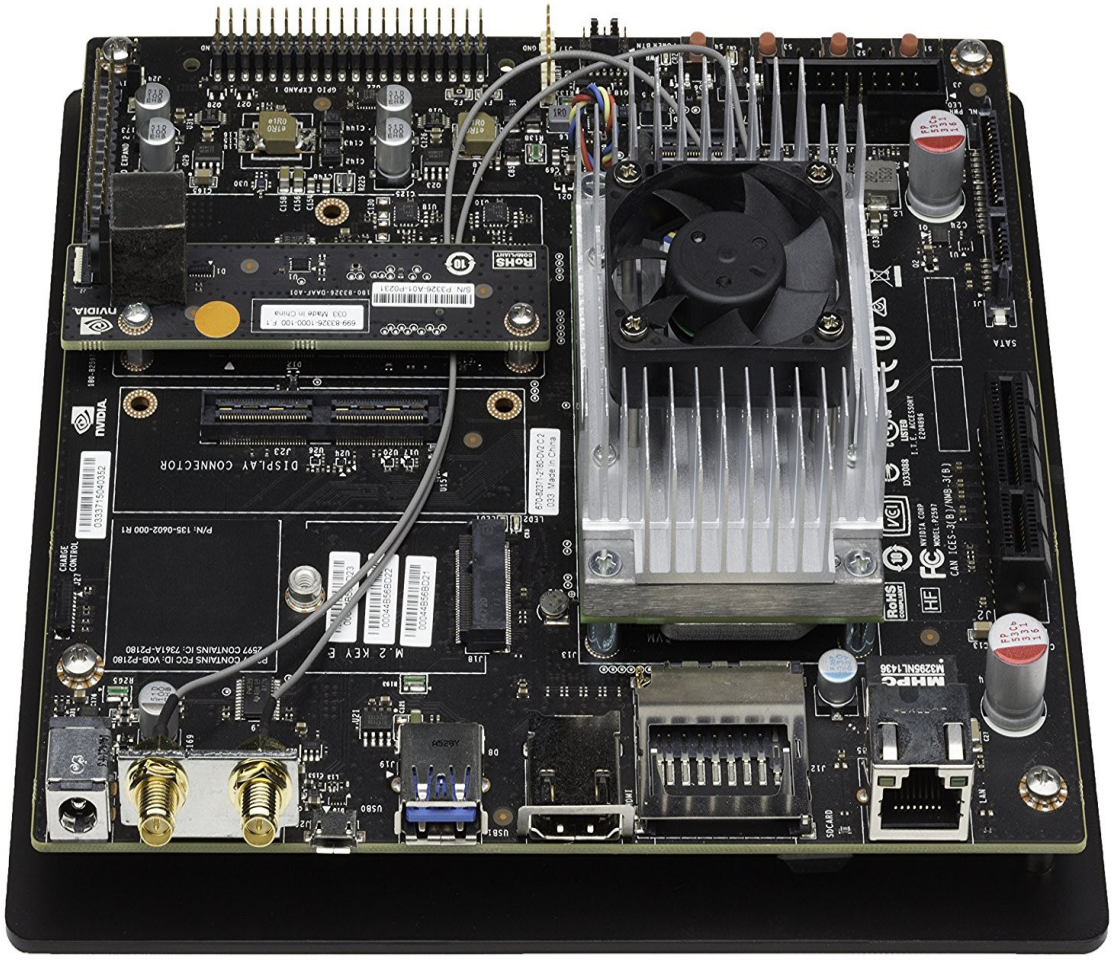
For this work the NVIDIA Jetson TX1 was chosen as the embedded system. The NVIDIA Jetson TX1 is a full-featured development platform for visual computing; it is ideal for applications requiring high computational performance in a low power envelope (the whole board just consumes 10 watts of power).

Jetson TX1 comes with a GNU/Linux environment, includes support for many common APIs and is supported by NVIDIAs complete development tool chain. In particular, the board employs quad-core ARM Cortex-A57, 4GB LPDDR4, integrated 256-core Maxwell GPU, where Maxwell is the codename

for a GPU microarchitecture developed by NVIDIA.

The GPU is the main reason why the board has been chosen: despite CPU who always runs codes in sequence, the main feature of GPU is the ability of parallelizing codes and increasing speed more easily easily than a CPU. Therefore, the use of GPU could enhance our system performance a lot, allowing to analyze more signals at the same moment, thus increasing capacity of our system, being able to monitor larger enviroments.

Figure 21 – NVIDIA Jetson TX1 Developer Kit



Source: https://www.nvidia.com

The board (presented in Figure 21), as previously said, uses a Linux environment with Ubuntu 18.04 and a 64 bit architecture. In particular the board was provided with python 3.5, that was upgraded to the 3.6 version since some scikit-learn libraries had this requirements (PEDREGOSA *et al.*, 2011).

5 RESULTS AND DISCUSSIONS

This chapter summarizes the main results obtained in this work. Fist, we present the detection results. In the sequence, results for the classification are discussed. Then, results for the embedded system are detailed. Finally, results for the Lab Datasets are showed.

5.1 DETECTION

This section presents the results obtained for detecting transient events. The non-trivial problem of the detection comes from the fact that we have to find a robust solution that also generalizes over a wide multiplicity of signal, whose shapes cannot be checked one by one because of their very large number.

In this work, good classification performance ($>98\%$) was achieved, but this would not be possible without adapting our detection method in a different fashion for both the COOLL and the LIT dataset. The used algorithms are based on already known information about the datasets, giving the security that appliances have been detected properly. For example, in COOLL dataset all the waveforms last 6 seconds and around 3 to 5 seconds the devices are always in their steady state, being sure that our steady features are properly calculated. Moreover, for the final results, it turned out that it was not useful to calculate the features in the transient interval to have excellent (>98%) classification performances. Additionally, COOLL dataset only provides single appliance waveforms, therefore no detector was needed.

The LIT dataset is a bit different, since even multiple appliance waveforms are present and thus a detector is needed. In this case a modified detector was used, working with the same principle of the HAND algorithm, but exploiting the start instants directly provided by the LIT dataset, as shown in Figure 22.

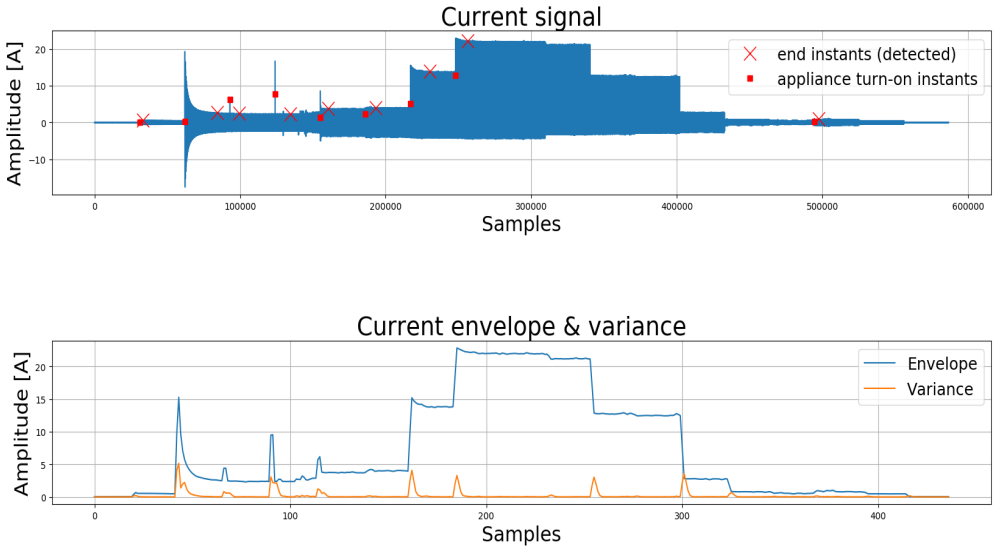
However, this does not represent the real scenario, where we are not suppose to know information about loads' signals (specially about turn on and off events). In this case, we already discussed in chapter 3 that the HAND algorithm for transients detection was implemented (Nait Meziane *et al.*, 2017). The reason why the project moved to the use of a modified detector is that the HAND algorithm was not too precise in aggregated waveforms. In fact, HAND uses the variance of the current envelope to detect transient intervals, by intuitively capture those parts of the signals where the current varies the most.

Table 4 refers to tests conducted with the HAND detector over a group of 16 waveforms (LIT Dataset), each having 8 loads connected. As we can observe, not all the loads' transients are captured, loosing the acquisition of the respective features.

During this work, the approach of HAND highlighted in particular three problems:

1. the settling of an adaptive threshold could avoid the detector to sense appliances operating at very

Figure 22 – 8E0P0I0M0N0H0W0Y0 signal analyzed with the modified detector: here all the transients are detected correctly.



Source: Own Authorship

Table 4 – Detector tests: aggregate signals with 8 appliances (D0, G0, P0, Q0, M0, N0, H0, E0) in LIT Dataset (RENAUX *et al.*, 2018b)

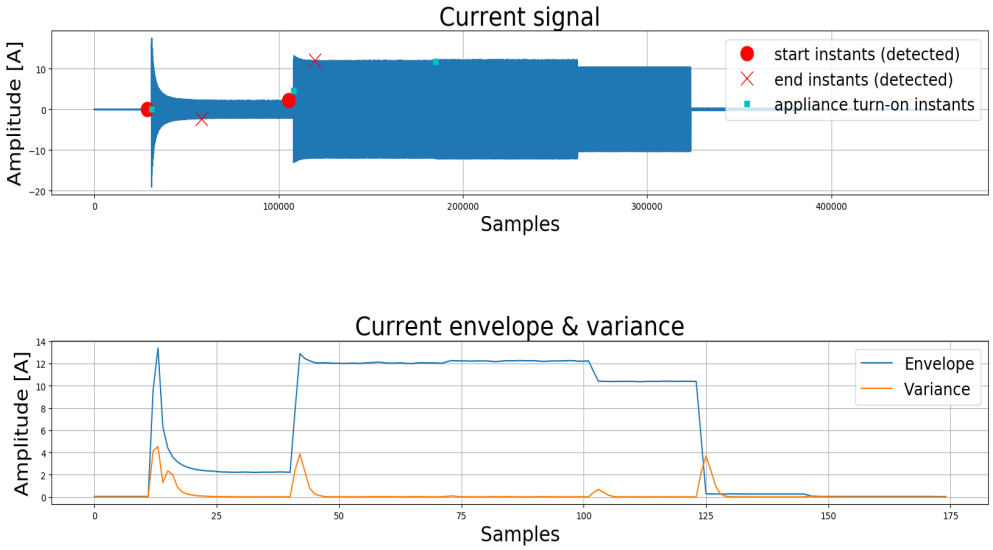
Waveform Number	D0	G0	P0	Q0	M0	N0	H0	E0
80000	No	No	Yes	Yes	No	No	No	No
80001	Yes	No	Yes	Yes	No	Yes	No	No
80002	Yes	Yes	Yes	Yes	Yes	No	No	No
80003	Yes	Yes	Yes	Yes	Yes	No	No	No
80004	Yes	Yes	Yes	Yes	Yes	Yes	No	No
80005	Yes	Yes	Yes	Yes	Yes	Yes	No	No
80006	Yes	Yes	Yes	Yes	Yes	No	No	No
80007	Yes	Yes	Yes	Yes	No	Yes	No	No
80008	Yes	No	Yes	Yes	No	Yes	No	No
80009	Yes	No	Yes	Yes	No	Yes	No	No
80010	Yes	Yes	Yes	Yes	Yes	No	No	No
80011	Yes	Yes	Yes	Yes	Yes	Yes	No	No
80012	Yes	Yes	Yes	Yes	Yes	No	No	No
80013	Yes	Yes	Yes		Yes	Yes	No	No
80014	Yes	Yes	Yes	Yes	Yes	No	No	No
80015	Yes	Yes	Yes	Yes	No	Yes	No	No

Source: Own Authorship

low current values (see Figure 23);

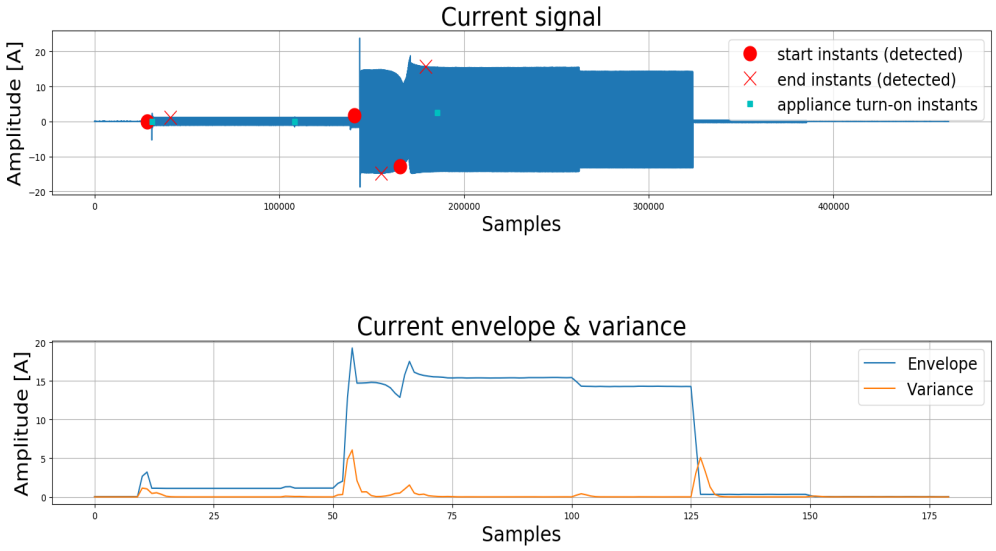
2. the appliances whose transient swings many times before reaching the steady-state will be detected multiple times: this can result in different features values that can lead to wrong predictions (see Figure 24);
3. since variance is sensitive to signal variations, the presence of noise can cause false positive events detection (see Figure 25).

Figure 23 – Third load overlapped with the previous ones and is not detected because its amplitude is relatively small.



Source: Own Authorship

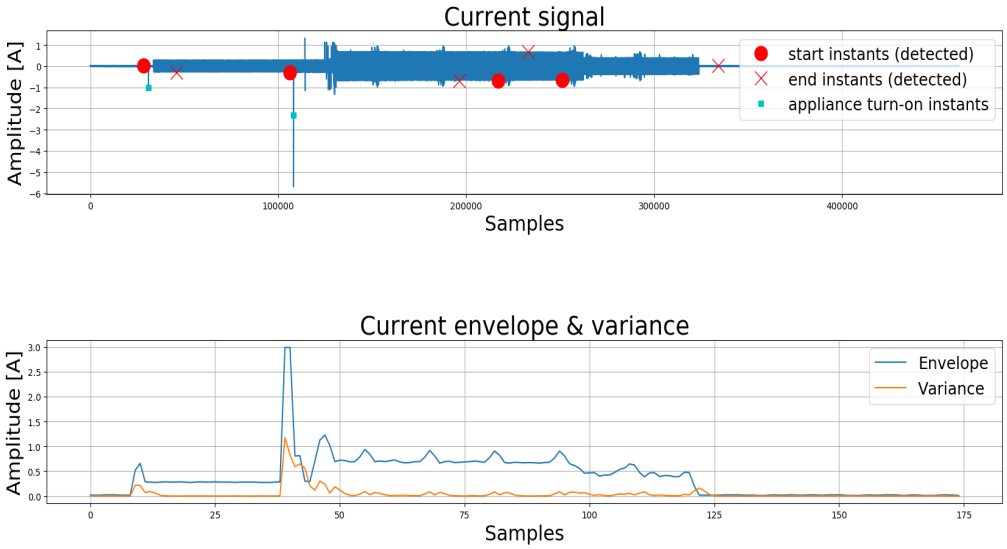
Figure 24 – Third load is characterized by a transient that swings two times leading to two detections instead of one.



Source: Own Authorship

Moreover, it was observed that the algorithm also has problems in targeting those signals whose transient changes very smoothly, hence signals with very low envelope variance. Therefore the following results will not be related to signals "realistically detected", since the uncertainties connected to HAND detector required the signals to be checked one by one, and also not giving the security that good results could be achieved.

Figure 25 – A highly noised signal produce many wrong detections.



Source: Own Authorship

5.2 CLASSIFICATION RESULTS

In this section, the results of the classifiers' training are presented. For the project, the choice of a proper model was conducted taking into account the prediction accuracy of four classifiers: Logistic Regression (LR), Support Vector Machine (SVM), K-Nearest-Neighbors (KNN), and Random Forest (RF). For the training process always the same procedure was employed, which consists in using in sequence:

- Grid Search;
- Sequential Backward Selection (SBS);
- Grid Search;
- Principal Components Analysis (PCA).

A first analysis with grid search allows to optimize the choice of the hyperparameters to increase model accuracy. Then, SBS is performed to target the features of the dataset who lowers prediction performances: those features will be then removed to reduce the dimension of the dataset, also decreasing the computational complexity of the system. Grid search is therefore applied another time, to search for a new optimal combination of parameters over the reduced dataset. In the end, a control with PCA is conducted, to see if there is actually some way of further increase the performances of classifiers. Here follows the results obtained, written in term of the percentage of the prediction accuracy over the test dataset:

In the case of Table 5, PCA was not employed in the end, since it would be just an additional task for classifiers that already have excellent (>98%) performances. Even after SBS, where for all cases

Table 5 – COOLL Dataset Test Accuracies (%)

Classifier	Preliminary test	Grid Search	SBS	Grid Search
LR	99.4	99.4	**99.4**	99.4
SVM	99.4	99.4	**99.4**	99.4
KNN	98.2	98.2	**98.8**	98.8
RF	99.4	99.4	**99.4**	99.4

Source: Own Authorship

the dimension of the dataset was reduce from 26 to 17 features, the accuracy of classifiers does not change at all, an indication that the selected features are robust.

The results of Table 5 are promising. In this case, the dataset played a big role, since it provides just waveforms relative to individual appliance measurements. As well as we will see for the LIT dataset, individual appliances are easier to analyze, since no disaggregation is performed and detection can be more accurate.

On the other hand, even though the accuracy reached are very good (>98%), our main purpose is to generalize as much as we can to possibly classify other appliances or, at least, correctly classify the appliances already present in the dataset under different conditions. For example, when appliances overlap because they are connected at the same system, their time constants could change. That is why just analyzing individual waveforms is not enough to implement a NILM module, therefore we moved to LIT dataset:

Table 6 – LIT Dataset Test Accuracies (%)

Classifier	1 APP	2 APP	3 APP	8 APP	Preliminary test	Grid Search	SBS	Grid Search	PCA
LR	97.6	98.5	94.3	90.3	88.2	88.7	87.4	87.4	**94.3**
SVM	97.6	98.9	98.4	90.9	96.2	96.2	97.0	95.4	**97.3**
KNN	95.2	98.9	96.5	91.6	94.5	94.5	95.6	95.6	**96.8**
RF	98.8	99.3	97.5	96.8	97.6	97.9	98.1	**98.1**	97.3

Source: Own Authorship

To analyze better the results obtained for the LIT dataset (Table 6), we decided first to investigate the feature datasets relative single appliance waveforms (1 APP), then analyze the ones with two (2 APP), three (3 APP) and eight appliances (8 APP) alone, in order to understand how the algorithm works in different scenarios.

From the results, it is evident that the dataset referring to waveforms having 8 appliances brings down classifiers' accuracy. Probably, despite the detection algorithm that we already said not to be that robust, a simple reason could be that the waveforms are not enough to give useful information (MULINARI *et al.*, 2019).

In fact, while 2 APP and 3 APP features dataset recorded respectively 1344 and 1584 measurements, 8 APP has just 768. That's because 2 APP waveform dataset, for example, contains information about just 7 different appliances providing an amount of 672 signals. On the other hand, 8 APP dataset

provides only 96 waveforms that must be used to calculate the features for 17 different appliances, meaning that we have relatively low information content for every appliance .

However, if compared to the 1 APP dataset where each appliance is analyzed just 16 times, 8 APP should have better results. The reason why accuracies are not as we expected, is the same reason why COOLL dataset achieve good results in the order of 98% of accuracy: the detection in waveforms containing just one appliance is far more simple and does not need disaggregation, leading to more accurate and selective results, thus facilitating appliances' clustering. It is true then that in 8 APP same appliances are evaluated more times, but we have also to take into account that:

- performing disaggregation inevitably introduces some errors;
- detection is not completely reliable;
- appliances waveforms slightly changes when overlap with others.

Therefore this might explain why 8 APP does not achieve the same good accuracy values of 1, 2 and 3 APP (note that this is not the same for the random forest classifier, as will be explained later).

It is important to say that the LR, SVM and KNN classifiers require, for optimal performance, feature scaling before model fitting. Returning to Table 6, after SBS the dataset dimension was reduce from 30 features to:

- 23 for LR;
- 28 for SVM;
- 21 for KNN;
- 24 for RF.

Once again, the control made by SBS is useful, since it enables us to target those features who are limiting and confusing our classifiers. Of course the features that has been deleted are not the same for every classifier because of their internal properties. In fact with the results obtained, the choice for the best model was made between SVM and RF classifier.

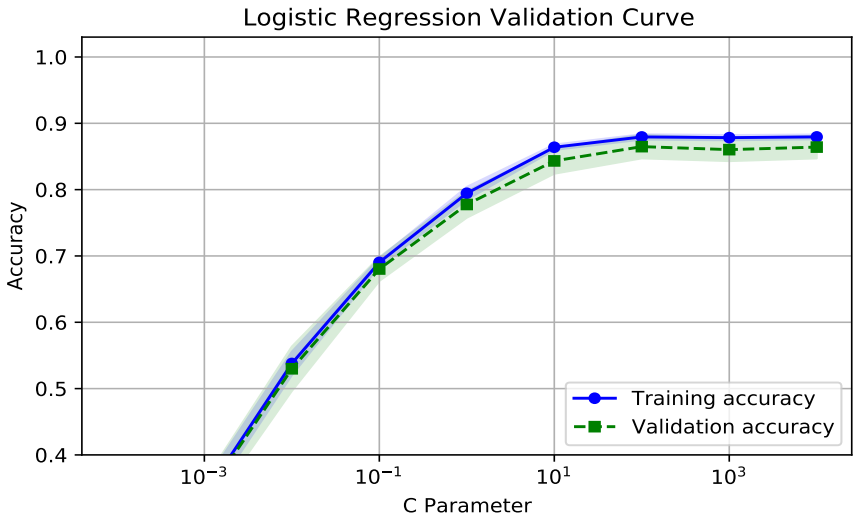
5.2.1 Detailed Analysis

As already discussed in section 4.4, some other tools can be used to see if the models were chosen correctly, taking into account the possibility that, despite the results obtained, there might be a further improvement in terms of over/under fitting or problems in the dataset. In particular, Learning and Validation curves have been evaluated, as well as the Receiver Operating Characteristic (ROC) for every classifier that was used in the case of the LIT Dataset. In the following sections the curves obtained are shown and analyzed.

5.2.1.1 Logistic Regression

Figure 26 shows the performance of the Logistic Regression model as we varying the C parameter. As we can observe, for values higher than $C = 10$, the accuracy seems to saturate. In this case we might want to keep $C = 10$ instead of higher values so we have an higher strength of the regularization parameter λ ($C = 1/\lambda$) and therefore staying away from an over fitting scenario.

Figure 26 – Validation curve with respect to the C parameter.



Source: Own Authorship

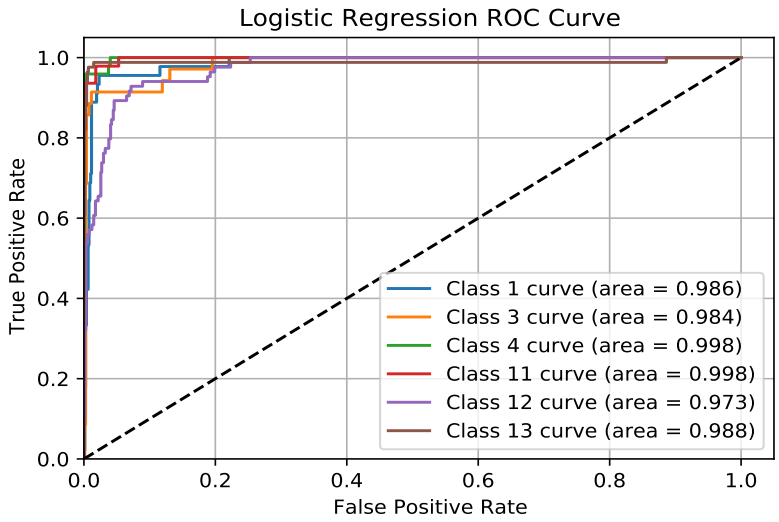
Figure 27 shows that for values higher than 1500 training samples there's a very low variance between the two models compared, meaning that increase the number of training samples is not really worth it.

Figure 27 – Learning curve Logistic Regression.

Source: Own Authorship

Figure 28 shows different ROC curves, each one calculated for a different appliance using the One-Versus-Rest technique. This curves allow us to explore more classifiers, understanding their true/false negative prediction rate. Figure 28 also displays curves' area (AUC area), whose values gets closer to 1 when we have a high number of true/false positive, thus correct classifications. Here we can observe how all the calculated curves are close to an ideal curve with AUC = 1 , meaning that the model performs very well over our multi-class classification problem.

Figure 28 – ROC curve Logistic Regression.



Source: Own Authorship

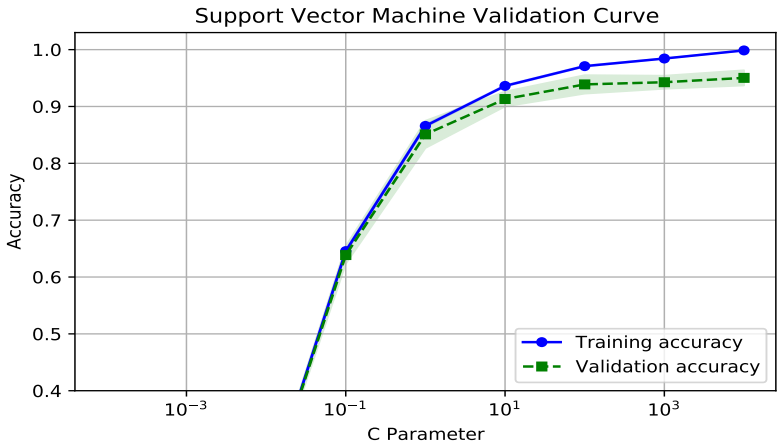
5.2.1.2 Support Vector Machine

Figure 29 shows that, in this case, if we chose values of C higher than 100 we start to over-fit the dataset: even though we can observe that the variance in this interval is not that high, it is important to keep variance as low as possible if we want the model to generalize over new appliances.

Figure 30 shows that for a number of samples higher than 2000 variance between the validation and the training set starts to reduce very slowly. In this case, we could think about using and higher number of training samples, for instance 2500. Nevertheless, remember that the validation set is a subset of training samples, therefore an higher number of training samples could cause over-fitting when we have to compare results with test dataset.

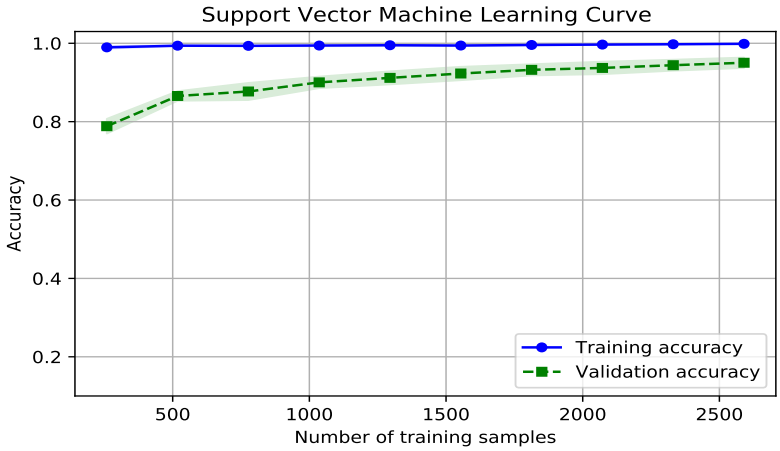
Figure 31 shows the behavior of the ROC curve for the SVM model: even here we can observe results similar to LR, although with better AUC values, meaning that the SVM model is able to distinguish very well among the different classes of the dataset. Notice that for some classes the AUC value is equal to 1, meaning that for those loads SVM is able to classify correctly every sample.

Figure 29 – Validation curve with respect to C parameter.



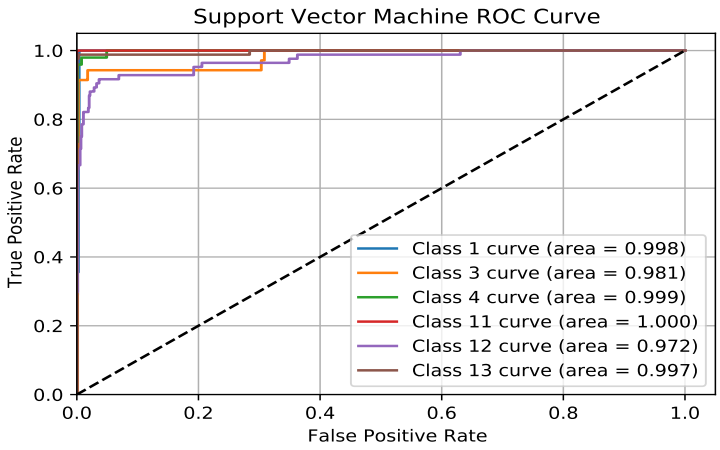
Source: Own Authorship

Figure 30 – Learning Curve SVM.



Source: Own Authorship

Figure 31 – ROC curve for SVM.

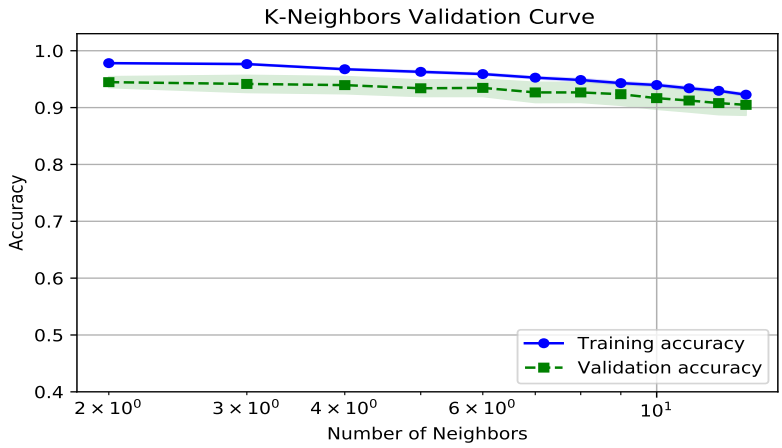


Source: Own Authorship

5.2.1.3 K-Nearest-Neighbors

We observe here from Figure 32 that the variance obtained through out all the curves should not affect too much, since it is always maintained very low. However, we can look at the curves with more meticulous eyes and observe that there seems to be a trade off between bias and variance as the number of neighbors is increased, therefore the latter should not be higher than 4 or 5.

Figure 32 – Validation curve with respect to the number of neighbors.

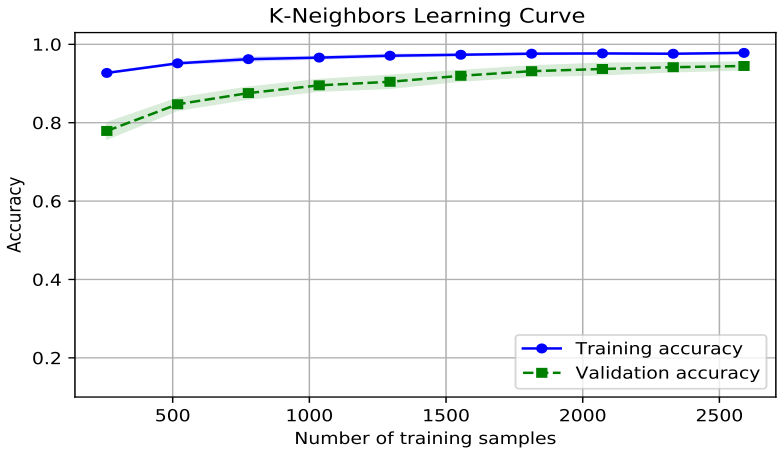


Source: Own Authorship

Figure 33 suggests that for a training dataset with a number of elements higher than 2000, the bias between the training and the validation dataset is reduced and acceptable. Again, we always have to think that our dataset is usually split into training and test set, for the purposes of model evaluation. The validation set was introduced to avoid the problem of over fitting and helps in generalize more our model. However, the validation set is extracted directly from the training dataset, thus we have to pay attention in increasing is number of training samples, because we might fall into an over fitting condition again.

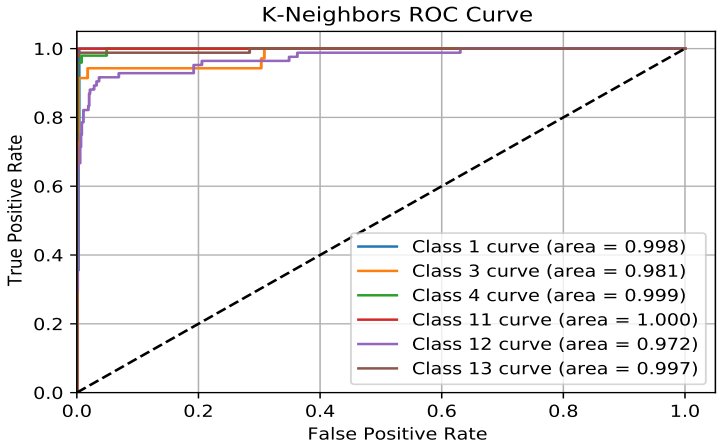
Figure 34 shows the behavior of the ROC curve for the KNN model: as we can observe even here the KNN model seems to be able to distinguish very well among the different classes of the dataset.

Figure 33 – Learning curve K-Neighbors.



Source: Own Authorship

Figure 34 – ROC for K-Neighbors.



Source: Own Authorship

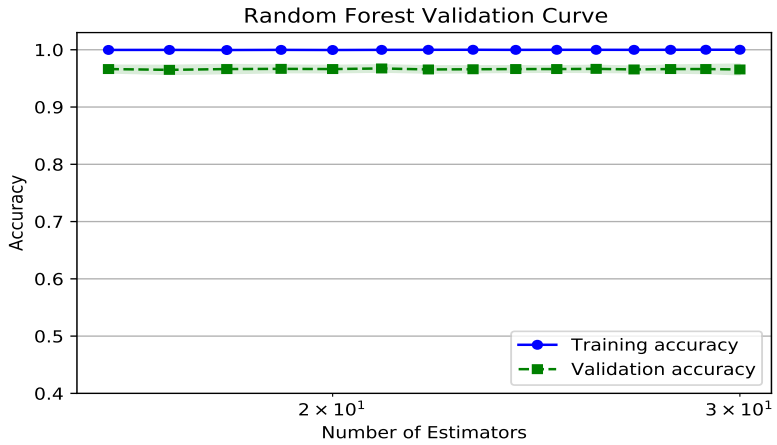
5.2.1.4 Random Forest

Figure 35 shows a particular situation. First of all, we observe that the profile of the two curves is the same and is more or less constant, meaning that augmenting the complexity of the classifier by increasing the number of estimators does not really worth it. Second, we noticed that even though the level of bias is good (meaning that a low error is obtained), there is a constant variance that does not reduce, which might be a symptom of overfitting.

Figure 36 shows that for a training dataset's dimension higher than 2000 the performance of the validation set are better and the variance remains approximately constant.

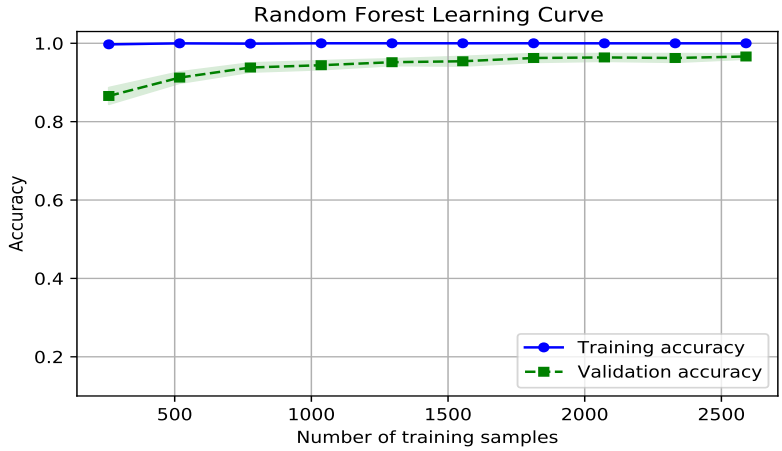
Figure 37 shows the behavior of the ROC curve for the Random Forest model: also here classifier performs very well in separating different classes.

Figure 35 – Validation curve with respect to the number of estimators.



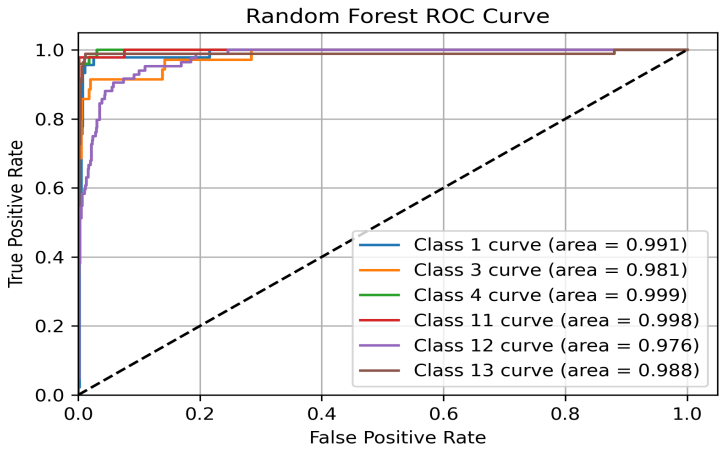
Source: Own Authorship

Figure 36 – Learning curve Random Forest.



Source: Own Authorship

Figure 37 – ROC for Random Forest.

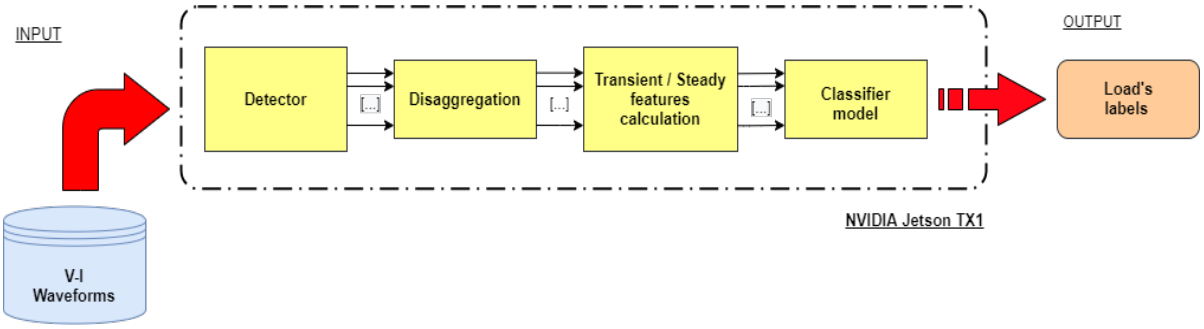


Source: Own Authorship

5.3 RESULTS IN THE EMBEDDED SYSTEM

This section shows the results obtained after the implementation of the algorithms on the board. In particular, the test were conducted off-line with already recorded waveforms as shown in the scheme of Figure 38.

Figure 38 – Monitoring test process.



Source: Own Authorship

The classifier model represents the already trained classifier that has been chosen for the tests. Fortunately, python provides a way to save models once they are trained and import them for classification, in a way that we will not have to repeat the training session every time we need to make predictions (please, refer to the python code .28 in the Appendix A).

NVIDIA Jetson TX1 comes with the python version 3.5 that was updated to 3.6 because of some libraries requirements. In particular, since the board embed a Linux operational system, it was more practical to run all the tests directly from the terminal. In particular, the following algorithm will be used to test our two datasets:

```
FUNCTION TEST(signal-to-analyze)
    Import : Classifier
    Import : Standard-Scaler (if required)
    Import : Detector
    Import : Disaggregator
    Import : Features
    (on-instants, off-instants) ← Detector(signal-to-analyze)
    For (i,j) in (on-instants, off-instants)
        (load-I, load-V) ← Disaggregator(signal-to-analyze[i:j])
        load-features ← Standard-Scaler(Features(load-I,load-V))
        OUTPUT Classifier.Prediction(load-features)
    EndFor
```

```
EndTEST
```

---

where we suppose that *signal-to-analyze* contains both the current and the voltage signal .

5.3.1 COOLL Dataset

In the case of COOLL Dataset, all classifiers achieved excellent accuracies ($\geq$ 98%), as demonstrated in the previous sections, hence the tests were run with all of them. The next figures (Figure 39 – 48) will show some of the results obtained on the NVIDIA Jetson TX1 board (please, refer to the python code .29 in the Appendix A).

Figure 39 – Saw_5 load classification.

```
>>>
>>> # SAW_5 RECOGNITION
...
>>> exec(open('logistic_regression.py').read())
Appliance name:  Saw_5
Execution time: 0.0014455318450927734
>>> exec(open('support_vector_machine.py').read())
Appliance name:  Saw_5
Execution time: 0.001813650131225586
>>> exec(open('k_neighbors.py').read())
Appliance name:  Saw_5
Execution time: 0.004002571105957031
>>> exec(open('random_forest.py').read())
Appliance name:  Saw_5
Execution time: 0.1185457706451416
>>>
```

Source: Own Authorship

Figure 40 – Sander_1 load classification.

```
>>>
>>> # SANDER_1 RECOGNITION
...
>>> exec(open('logistic_regression.py').read())
Appliance name:  Sander_1
Execution time: 0.0014808177947998047
>>> exec(open('support_vector_machine.py').read())
Appliance name:  Sander_1
Execution time: 0.0018804073333740234
>>> exec(open('k_neighbors.py').read())
Appliance name:  Sander_1
Execution time: 0.004082918167114258
>>> exec(open('random_forest.py').read())
Appliance name:  Sander_1
Execution time: 0.1182718276977539
>>>
```

Source: Own Authorship

Figure 39 – 48 show how tests were conducted from terminal of the NVIDIA board. A python script is executed, then its result displays what was the predicted class name of the analyzed appliance and also how much it took, in terms of milliseconds, from the whole algorithm to be run.

Figure 41 – Drill_1 load classification.

```
>>>
>>> # DRILL_1 RECOGNITION
...
>>> exec(open('logistic_regression.py').read())
Appliance name:  Drill_1
Execution time: 0.001783132553100586
>>> exec(open('support_vector_machine.py').read())
Appliance name:  Drill_1
Execution time: 0.0021262168884277344
>>> exec(open('k_neighbors.py').read())
Appliance name:  Drill_1
Execution time: 0.004431009292602539
>>> exec(open('random_forest.py').read())
Appliance name:  Drill_1
Execution time: 0.11881065368652344
>>>
```

Source: Own Authorship

Figure 42 – Drill_5 load classification.

```
>>>
>>> # DRILL_5 RECOGNITION
...
>>> exec(open('logistic_regression.py').read())
Appliance name:  Drill_5
Execution time: 0.0015001296997070312
>>> exec(open('support_vector_machine.py').read())
Appliance name:  Drill_5
Execution time: 0.0020503997802734375
>>> exec(open('k_neighbors.py').read())
Appliance name:  Drill_5
Execution time: 0.004495859146118164
>>> exec(open('random_forest.py').read())
Appliance name:  Drill_5
Execution time: 0.10725736618041992
>>>
```

Source: Own Authorship

Figure 43 – Hair_Dryer_2 load classification.

```
>>>
>>> # HAIR_DRAYER_2 RECOGNITION
...
>>> exec(open('logistic_regression.py').read())
Appliance name:  Hair_drayer_2
Execution time: 0.001135110855102539
>>> exec(open('support_vector_machine.py').read())
Appliance name:  Hair_drayer_2
Execution time: 0.002148866653442383
>>> exec(open('k_neighbors.py').read())
Appliance name:  Hair_drayer_2
Execution time: 0.0038123130798339844
>>> exec(open('random_forest.py').read())
Appliance name:  Hair_drayer_2
Execution time: 0.12087678909301758
>>>
```

Source: Own Authorship

As one can observe, the result achieved are almost perfect (100%). Figure 48 shows the case in which both the KNN and RF classifiers misclassifies the load. One of the solutions to compensate the error might be, for example, providing the dataset with some additional features to add informations for the classifiers, thus improving selectivity. Another solution, which is a common practice when multiple classifiers has very good accuracies, could be using together the classifiers to build an ensemble model,

Figure 44 – Hair_Dryer_3 load classification.

```
>>>
>>> # HAIR_DRAYER_3 RECOGNITION
...
>>> exec(open('logistic_regression.py').read())
Appliance name:  Hair_drayer_3
Execution time: 0.001638650894165039
>>> exec(open('support_vector_machine.py').read())
Appliance name:  Hair_drayer_3
Execution time: 0.002026796340942383
>>> exec(open('k_neighbors.py').read())
Appliance name:  Hair_drayer_3
Execution time: 0.003665447235107422
>>> exec(open('random_forest.py').read())
Appliance name:  Hair_drayer_3
Execution time: 0.12294363975524902
>>>
```

Source: Own Authorship

Figure 45 – Hedge_Trimmer_2 load classification.

```
>>>
>>> # HEDGE_TRIMMER_2 RECOGNITION
...
>>> exec(open('logistic_regression.py').read())
Appliance name:  Hedge_trimmer_2
Execution time: 0.0014939308166503906
>>> exec(open('support_vector_machine.py').read())
Appliance name:  Hedge_trimmer_2
Execution time: 0.002119302749633789
>>> exec(open('k_neighbors.py').read())
Appliance name:  Hedge_trimmer_2
Execution time: 0.004534244537353516
>>> exec(open('random_forest.py').read())
Appliance name:  Hedge_trimmer_2
Execution time: 0.11826086044311523
>>>
```

Source: Own Authorship

Figure 46 – Paint_Stripper_1 load classification.

```
>>>
>>> # PAINT_STRIPPER_1 RECOGNITION
...
>>> exec(open('logistic_regression.py').read())
Appliance name:  Paint_stripper_1
Execution time: 0.0014615058898925781
>>> exec(open('support_vector_machine.py').read())
Appliance name:  Paint_stripper_1
Execution time: 0.0021698474884033203
>>> exec(open('k_neighbors.py').read())
Appliance name:  Paint_stripper_1
Execution time: 0.004288673400878906
>>> exec(open('random_forest.py').read())
Appliance name:  Paint_stripper_1
Execution time: 0.12096667289733887
>>>
```

Source: Own Authorship

able to make its prediction by majority voting.

In conclusion, it is also interesting to see how the complexity of a classifier influences the execution time (reported in Figures 39 – 51) in milliseconds). The tests evidence, as a matter of fact, that in general the Random Forest classifier takes almost 100 times more than other classifiers to be executed: that is because multiple decision trees are evaluated for the final classification.

Figure 47 – Router_1 load classification.

```
>>>
>>> # ROUTER_1 RECOGNITION
...
>>> exec(open('logistic_regression.py').read())
Appliance name:  Router_1
Execution time: 0.0014774799346923828
>>> exec(open('support_vector_machine.py').read())
Appliance name:  Router_1
Execution time: 0.002118825912475586
>>> exec(open('k_neighbors.py').read())
Appliance name:  Router_1
Execution time: 0.004248142242431641
>>> exec(open('random_forest.py').read())
Appliance name:  Router_1
Execution time: 0.11884784698486328
>>>
```

Source: Own Authorship

Figure 48 – Lamp_2 load classification.

```
>>> # LAMP 2 RECOGNITION
...
>>> exec(open('logistic_regression.py').read())
Appliance name:  Lamp_2
Execution time: 0.0013387203216552734
>>> exec(open('support_vector_machine.py').read())
Appliance name:  Lamp_2
Execution time: 0.008597850799560547
>>> exec(open('k_neighbors.py').read())
Appliance name:  Lamp_3
Execution time: 0.003835439682006836
>>> exec(open('random_forest.py').read())
Appliance name:  Lamp_1
Execution time: 0.11259126663208008
>>>
```

Source: Own Authorship

5.3.2 LIT Dataset

For the LIT Dataset the accuracies achieved by the classifiers were not as good as for the COOLL Dataset: here the best one is the 98.1% obtained using Random Forest. Therefore the tests have been conducted just using the RF model, because in this tests we were pretty much interested in seeing how the NILM algorithm works with aggregated waverforms, thus higher prediction accuracy was needed. Figures 49 – 51 show some of the results obtained (please, see how the tests were done in python .30).

Even in this case the system works very well. However, sometimes some errors appeared in tests, specially when waveforms measuring 8 appliances are examined more likely some misclassifications can occur. In particular, note that when analyzing waveforms with 8 appliances always 9 are detected: this happens because, when building the LIT dataset, the team faced a bug in the script that automate the waveform acquisition (RENAUX *et al.*, 2018c), thus always repeating at the end of a signal the first appliance that turned on.

As previously discussed for the results obtained in the COOLL dataset, in Figures 49 – 51, we can see that the execution times are the same in LIT dataset ($\approx 0.14ms$) when signals with just one

Figure 49 – Results obtained from different waveforms including 1 appliance.

```
nvidia@tegra-ubuntu:/media/nvidia/187A-23B9/LIT_Dataset/Test_waveforms$ python3
Python 3.6.10 (default, Dec 19 2019, 23:04:32)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import test as ts
>>>
>>> # SIGNALS WITH 1 APPLIANCE
...
>>> ts.test('1/F0.mat')
Appliance 1  name:  F0
 Execution time: 0.13469290733337402
>>>
>>> ts.test('1/J0.mat')
Appliance 1  name:  J0
 Execution time: 0.1278064250946045
>>>
>>> ts.test('1/O0.mat')
Appliance 1  name:  O0
 Execution time: 0.13032317161560059
>>>
```

Source: Own Authorship

appliance are analyzed. Of course, when appliances started to be more, the execution time increases, even because the function for disaggregation (.31) will be involved.

To conclude we can say that, after the model evaluation process, the classifier was included in an algorithm that, in sequence, applies detection, disaggregation, features' calculation, and classification processes. This algorithm was used in simulations over the test dataset to emulate a device capable of collecting external data and recognize the respective loads. In the simulation, the execution time of all the classifier models, when analyzing one single appliance, were recorded:

- LR: 1.47550 ± 0.0054 μs;
- SVM: 2.71 ± 0.6 μs;
- KNN: 4.140 ± 0.09 μs;
- RF: 117.74 ± 1.4 μs.

It is interesting to see how the complexity of a classifier influences the execution time. The tests evidence that, in general, the RF classifier takes almost 100 times more than other classifiers to be executed. That is because multiple decision trees (18 in our case) are evaluated for the final classification. Moreover, to see how much the whole process takes, also the execution times related to aggregated waveforms were analyzed, i.e.:

- 1APP: 0.13094 ± 0.0020 ms;
- 2APP: 0.426 ± 0.07 ms;
- 3APP: 0.688 ± 0.08 ms;
- 8APP: 2.987 ± 0.08 ms.

Note that, in this case, the RF classifier was implemented. In particular, for 1APP waveforms,

Figure 50 – Results obtained from different waveforms including 2 and 3 appliances.

```
>>>
>>> # SIGNAL WITH 2 APPLIANCES
...
>>> ts.test('2/A0_M0.mat')
Appliance 1  name:  A0
Appliance 2  name:  M0
 Execution time: 2.8883442878723145
>>>
>>> ts.test('2/H0_B0.mat')
Appliance 1  name:  H0
Appliance 2  name:  B0
 Execution time: 0.4988374710083008
>>>
>>> ts.test('2/N0_Q0.mat')
Appliance 1  name:  N0
Appliance 2  name:  Q0
 Execution time: 0.35387110710144043
>>>
>>> # SIGNAL WITH 3 APPLIANCES
...
>>> ts.test('3/P0_U0_Z0.mat')
Appliance 1  name:  P0
Appliance 2  name:  U0
Appliance 3  name:  Z0
 Execution time: 0.8433053493499756
>>>
>>> ts.test('3/S0_I0_Y0.mat')
Appliance 1  name:  S0
Appliance 2  name:  I0
Appliance 3  name:  Y0
 Execution time: 0.6418187618255615
>>>
>>> ts.test('3/T0_V0_M0.mat')
Appliance 1  name:  T0
Appliance 2  name:  V0
Appliance 3  name:  M0
 Execution time: 0.5784451961517334
>>>
```

Source: Own Authorship

the execution time takes a little bit longer than the RF result previously registered due to the detection and disaggregation processes. As expected, we can see that higher the number of appliances, the longer it takes to analyze them. This problem may be solved if all the signals are processed in parallel.

Figure 51 – Results obtained from different waveforms including 8 appliances.

```
>>>
>>> # SIGNAL WITH 8 APPLIANCES
...
>>> ts.test('8/D0_M0_S0_G0_H0_N0_R0_E0.mat')
Appliance 1  name:  D0
Appliance 2  name:  M0
Appliance 3  name:  S0
Appliance 4  name:  G0
Appliance 5  name:  H0
Appliance 6  name:  N0
Appliance 7  name:  R0
Appliance 8  name:  E0
Appliance 9  name:  D0
 Execution time: 3.139965295791626
>>>
>>> ts.test('8/Q0_H0_N0_M0_P0_E0_I0_V0.mat')
Appliance 1  name:  Q0
Appliance 2  name:  H0
Appliance 3  name:  N0
Appliance 4  name:  M0
Appliance 5  name:  P0
Appliance 6  name:  E0
Appliance 7  name:  I0
Appliance 8  name:  V0
Appliance 9  name:  Q0
 Execution time: 2.9609110355377197
>>>
>>> ts.test('8/X0_E0_H0_I0_M0_P0_N0_D0.mat')
Appliance 1  name:  X0
Appliance 2  name:  E0
Appliance 3  name:  H0
Appliance 4  name:  I0
Appliance 5  name:  M0
Appliance 6  name:  P0
Appliance 7  name:  N0
Appliance 8  name:  D0
Appliance 9  name:  X0
 Execution time: 2.860271453857422
>>>
```

Source: Own Authorship

5.4 LAB TESTS ON REAL MEASUREMENTS

In this section the result obtained for the Lab dataset are shown. As we said in section 4, a part of the project, whose aim was to produce an algorithm prototype for a NILM system, was dedicated to realize some appliance measurements on a real scenario to test the project even on real world applications.

In section 5.2 and 5.3 we saw that, despite some issues about detection, classifiers reached good accuracies ($> 97\%$) over test waveforms of LIT and COOLL datasets. Moreover, results on NVIDIA board confirmed that the algorithm can be easily implemented on an embedded system in order to analyze loads' waveforms. But what happens when we analyze real signals? For this purpose we create in laboratory a reduced set of waveforms with new appliances to further analyze the NILM algorithm.

As explained in Section 4, here, no turn-on or off events were recorded because there was no such a precise way of doing so, but even because this scenario resembles more the task of a possible real implementation of the algorithm. Even though aggregated waveforms are available in Lab dataset

and are named with the label of the corresponding loads, we do not have the possibility of determine a priori where turn-on and off events are located and whose load they correspond to. However, in case of a single load waveform we can simply use the detection algorithm and, if something is detected, we will know for sure that the load is the one the waveform refers to. This consideration leads to the decision of building a training features dataset using just single load waveforms, which ended in a smaller amount of information that classifiers could use to generalize over tests. This influenced considerably the final results.

5.4.1 Building a new feature dataset for the optimization of classifiers

To start, all the 5 groups of 10 waveforms related to the 5 analyzed appliances where fed into the HAND algorithm, followed by the features extrapolation process. In particular, 21 features were extrapolated, i.e.: peak of power signal, active, reactive and apparent power, ict, ar, lpa, asy, r, sc, dpb, angp, L, md, magnitude of 50Hz, 100Hz, 150Hz, 200Hz and 250Hz harmonics, TED and TOD (please refer to section 3.2 for the meaning of features' labels). Note that in this case, as suggested in Hassan *et al.* (2014), we just use the first 5 harmonics instead of using a more extended content as in the case of COOLL and LIT dataset. This was done because in previous results the SBS process always filtered out higher harmonics, therefore it was decided to group harmonics with order greater than 5 in TED and TOD features, respectively total even and odd harmonic distortion. This allowed to reduce the features dataset dimension without completely get rid of high harmonics' information content.

The resulting features dataset is thus composed by 432 entries, which is sufficient for classifiers to learn how discriminate among 5 different appliances: earlier results shows that LIT features dataset, with its 4112 entries, was enough to learn how to classify among 26 different loads as well as COOLL features dataset, with 840 entries to distinguish 12 devices. Table 7 shows the results obtained for classification task during the model optimization procedure.

Table 7 – Lab Dataset Accuracies in Optimization process (%)

Classifier	Preliminary test	Grid Search	SBS	Grid Search
LR	93.8	93.8	96.2	92.3
SVM	91.5	96.9	96.9	97.7
KNN	90.8	91.5	93.8	91.5
RF	92.3	92.3	95.4	93.8

Source: Own Authorship

In particular, after SBS the dataset dimension was reduce from 21 features to:

- 19 for LR;
- 13 for SVM;
- 13 for KNN;

- 12 for RF.

Once the training and optimization process was finished, the classifiers model were saved and plug inside the NILM algorithm for some tests on aggregated waveforms. This was done to further analyze classifiers ability to generalize over new datas, since the feature training dataset was based just on single appliance measurements, thus no errors due to disaggregation appeared.

In this preliminary test 28 aggregated waveforms of 2 appliances were analyzed. At the end, the detection algorithm produce a total of 144 measurements, each of them was then disaggregated to calculate the features needed for classification. The results obtained follow in Table 8.

Table 8 – Lab Dataset Preliminary Test results

	LR	SVM	KNN	RF
Accuracy (%)	91.5	97.7	92.3	94.6
Number of measurements	144	144	144	144
Errors	36	55	49	51
Percentage of errors	25 %	38 %	34 %	35 %

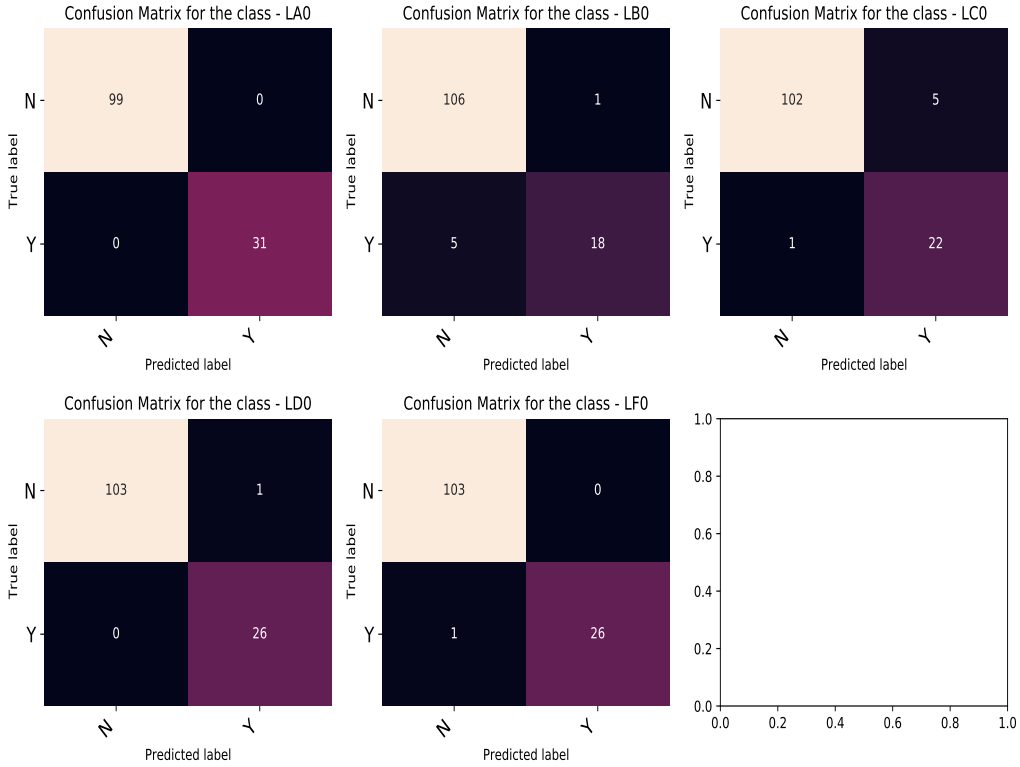
Source: Own Authorship

Table 8 shows that the results obtained are not reliable at all since, the best that we can obtain are, in the case of Logistic Regression classifier, 36 errors over 144 measurements, which is too much. This indicates that there were something wrong during the training and optimization process, classifiers might over-fit training datas leadining to poor generalization capabilities. However, to avoid this, validation and learning curve has already been used to check whether the models were over-fitting data or not (as showed in section 5.2).

Therefore we checked at confusion matrix of the different classifiers as reported in Figures 52 – 55. Note that every time the number of false positives and false negatives for label LB0 and LC0 are the same, but mirrored. This fact indicates that classifiers have some difficulties in distinguish among those 2 classes: this happens because they can be though as the same class actually. As a matter of fact, in section 4.2.3, we saw that LB0 measurements were realized for the charge of a PC while it was OFF; LC0 measurements, instead, were realized for the same PC in charge, this time while it was ON. It turned out, by looking at their respective waveforms, that those two classes share the same behavior and the only difference is that when PC is ON it consumes a little bit more.

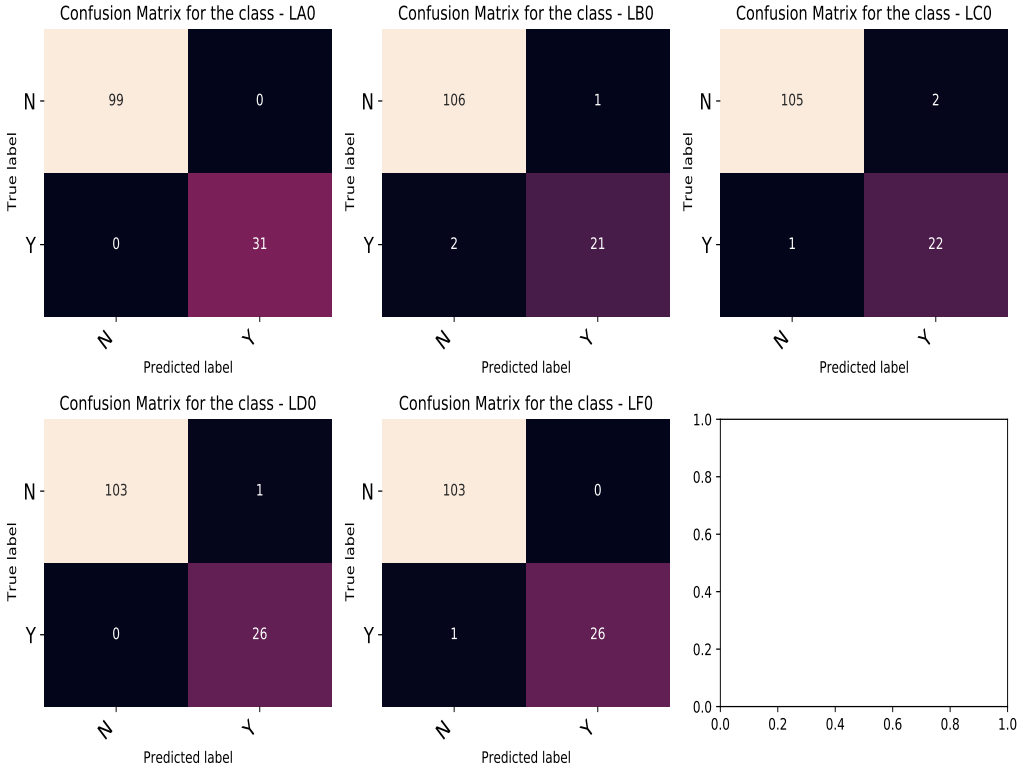
This fact is interesting, because if instead of considering the classes PC On/Off Charge we group them into a unique class (it can be PC Charge for instance) we expect a reduction in the number of errors. Moreover, to reduce wrong classifications, it was decided to increase the amount of information in the dataset by manually calculating features from waveforms. In particular, manually means that features were calculated from parts of the waveforms chosen without using HAND, but targeting some specific intervals of interest. This allowed to introduce more good-quality information to possibly separate more the distinction among the classes.

Figure 52 – Logistic Regression Confusion Matrix.



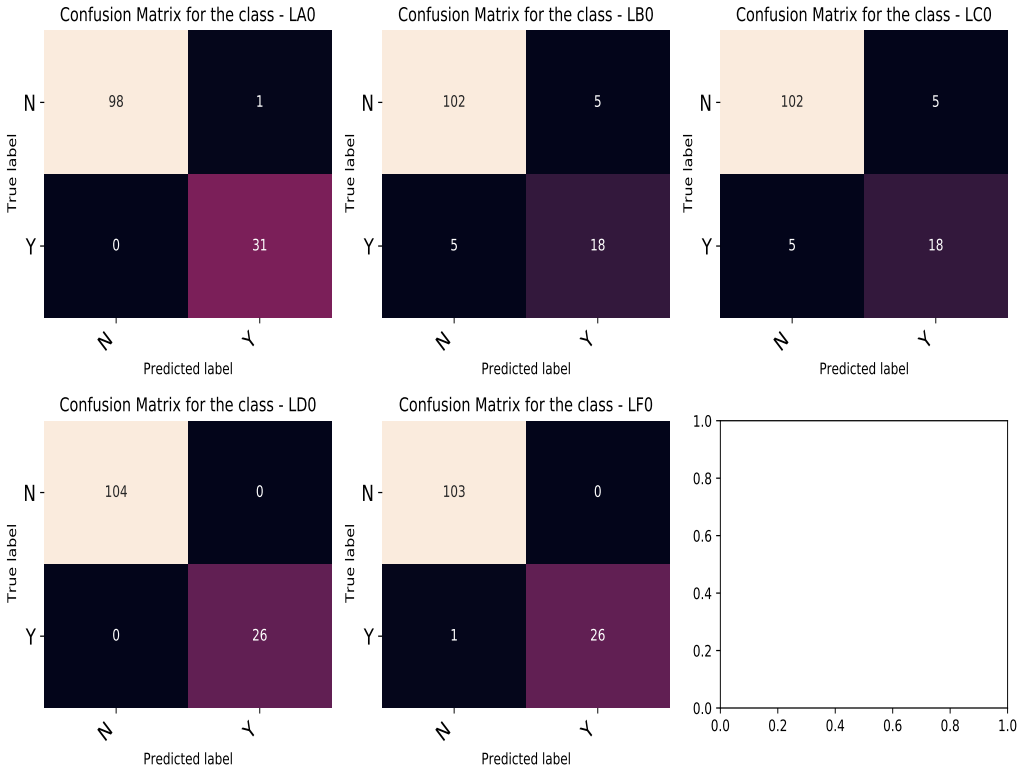
Source: Own Authorship

Figure 53 – Support Vector Machine Confusion Matrix.



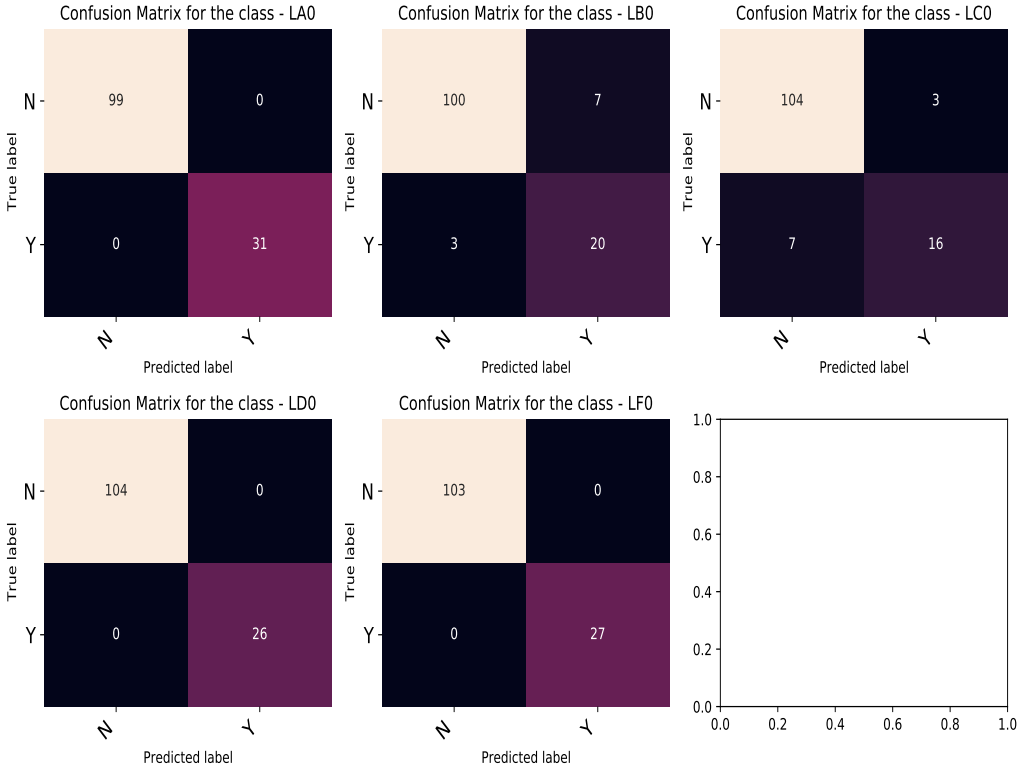
Source: Own Authorship

Figure 54 – K-Neighbors Confusion Matrix.



Source: Own Authorship

Figure 55 – Random Forest Confusion Matrix.



Source: Own Authorship

5.4.2 Validation of NILM algorithm on test dataset

After the considerations of the section above, the dataset was manually modified adding some new features to augment its information content. Moreover, we now neglect all the mistakes due to a swap between LB0 and LC0, or vice-versa.

Once those changes were include, the NILM algorithm was finally validated over new lab measurements. LA0, LB0, LC0, LD0, and LF0 were recorded singularly and simultaneously in aggregated waveforms with two appliances, for a total of 12 different groups, each one with 4 waveforms. Then, every waveform was analyzed and fed into the NILM algorithm where HAND realize a total of 551 event detected and whose features were sent to LR, SVM, KNN, and RF classifiers, in order to further evaluate their performances. The results achieved are shown in Table 9.

Table 9 – Lab Dataset Final Test results

	LR	SVM	KNN	RF
Accuracy (%)	94.6	96.9	91.5	92.3
Number of measurements	551	511	551	551
Errors	169	53	83	43
Percentage of errors	31 %	10 %	15 %	8 %

Source: Own Authorship

All the measurements were included in an excel page and reported at the end of the appendix to show the result achieved (please see Appendix .32). During experiments, we consider correct all the predictions whose label belongs to its waveform group, except for LB0 and LC0 that could be swap. In the excel page, all errors were marked in red and summed at the end of each group measurement.

Table 9 shows some relevant changes in the results. If we compare with Table 8, apart from LR, all other classifiers significantly improve their own prediction accuracy. In particular, SVM and RF worked even better than the preliminary tests, where they respectively made 55 and 51 errors over 144 measurements: here, instead, they made 53 and 43 errors over 551 measurements. Thus a reduction of wrong prediction even with an higher number of measurements. Those results, obtained in a lab on real devices, validate what was presented in previous sections with LIT and COOLL datasets, still pointing out that SVM and RF (and in particular the latter) best fit our NILM algorithm purposes.

However, we can appreciate in the results shown in Appendix .32, that if we take into account, for instance, one line of one group, the predictions of the four classifiers might not be the same, even though they are considered correct and that's because there were no information about where loads' events were located inside the waveform. What was known is that, since a waveform belonged to a certain group of loads, an appliance from that group was expected, no matter where inside the signal. This discrepancy, for sure introduced some errors that were not take into account, since that means check one by one all detected signals' intervals and see if the predicted label actually match the measured load, and this would

take too much time.

Nevertheless, further analysis on waveforms shown that, in the case of LD0 and LF0 some errors from classifiers could be neglected. As a matter of fact, those two classes come from the same load (i.e. a soldering station) in two different states. The respective signals don't have similar shapes as in the case of LB0 and LC0, but once LF0 was recorded always some spurs of LD0 appeared, likely because waveforms in Lab Dataset were recorded over an interval of 10 seconds, giving the possibility to the soldering station to change from state LF0 to LD0. Therefore, a further analysis on that could reduce the number of errors registered in Table 9, giving more credits to the robustness of the whole algorithm.

6 CONCLUSIONS AND FUTURE WORKS

In these days, it is almost impossible to find some devices not using an electrical power supply. Moreover, the increasing demand of energy caused by a growth of the world population claims for renewable energy sources, since fossil fuels are running out of stock and strongly impact our climate.

As well as the need of new sources, save energy, to reduce its waste and increase its use efficiency, is what we might want to do, either to save costs or to be more responsible . NILM aims to find a solution for power saving, trying to measure the aggregate power signal of a system and thus displaying the amount of power which is being used, targeting each of the loads consuming energy. Lot of projects are nowadays studying the best approach to complete this task and a wide variety of research literature is available, showing that there is a big interest moving towards a NILM solution.

In this work, a possible real implementation of a NILM system has been presented. Exploiting two loads datasets, it was shown that we can actually have an algorithm that recognizes the appliances connected to a target system that we want to monitor. Moreover, the results obtained from simulations were finally applied on loads measurements performed in a real scenario, giving a chance to further validate the NILM algorithm proposed in this work.

In particular, we saw from chapter 5 that the algorithm could be loaded on an embedded system, i.e., an NVIDIA TX1 Developer kit in our case. The latter, together with the developed algorithm, demonstrates its ability in recognizing loads from waveforms, after a detection, disaggregation and features extrapolation process, with a particular tendency to reduce the number of errors in the case where Random Forest Classifier is used. With a test accuracy of 99.4% in the case of COOLL and 98.1% for LIT dataset, we saw that RF classifier totalized 43 wrong predictions over 551 measurements during lab experiments, once again confirming that the proposed NILM algorithm works and RF is more suitable for its purposes.

However, during the project some problems were outlined. First of all, even though the accuracy reached from the classifiers are very good (>98%), sometimes some misclassifications appear, making the system not fully reliable if used for loads out of the analyzed datasets. By the way, the features datasets built for this project are composed of just a few transient state, steady-state and VI trajectory features.

In Zoha *et al.* (2012) and Du *et al.* (2010), different types of features are discussed. The latter for example presents two different methods, one based on Eigenvalue Analysis and SVD (Singular Value Decomposition) and one on the wavelet transform of current waveform, that could be exploited to build a new set of features, thus improving the amount of information available for the classifiers.

The other problem we faced is the lack of a robust and reliable detector. Although the core part of the recognition task is the classifier, as long as a proper algorithm for detection is not found, some important error components will prevent features to be extracted correctly, therefore yieldings errors in

classifiers' predictions. HAND algorithm has a very intuitive way of detecting turn-on/off events, even though its methodology is fully based on the shape of waveforms, rather than the informative content that we want to extrapolate from them. As written in section 5.1, transients could have different shapes that should be addressed in different ways in order to get the necessary content needed from classifiers. For example, a transient characterized by multiple ups and downs could cause multiple detection, thus the disaggregation algorithm might subtract one interval from another coming from the transient of the same load.

In Chapter 2 some of the state-of-the-art detection algorithms have been presented: in particular those algorithms turn out to be very precise for certain cases, but not that helpful in others. An interesting solution is provided in Renaux *et al.* (2018b), were the selectivity of detectors is combined together in an ensemble detector, able to chose the best interval that has been target from its sub-units.

Detection is probably where the proposed algorithm weakens, and that calls for new solutions. In particular, in the case of supervised monitoring, the nowadays available technologies in terms of learning algorithms are good enough for NILM purposes: what it is actually missing is a robust structure that helps extrapolating correctly signal features. Moreover, the wide variety of electrical devices suggest that supervised approach is likely to move towards custom solution for specific loads monitoring, otherwise too much effort will be spent on building appliances dataset that helps in generalize over a very large scenario. However, we proved that classifiers could generalize once they face appliances with the same nature of the ones used in the training process. That was the case of the experiments on Lab dataset, where the classes LB0 and LC0 merges together because they were different states of the same appliance.

Some other methods, instead, might want to use an unsupervised learning approach to get rid (or at least reduce) the dependence of classifiers models on datasets. This could save all the efforts made to realize tons of measurements in order to provide more data during the training process of learning algorithms. Moreover, those algorithms have an higher ability to generalize over new data, since the way data are clustered changes as long as we realize more measurements. However this could be also a drawback, because correct class separation won't be successful until we provide a considerable amount of data to those algorithms.

At the end of lab simulations, as presented in section 5.4, satisfactory results were achieved despite some improvements could be included in the methodology proposed in chapter 4. Here, in particular, follows the list of future works that could be applied to this project:

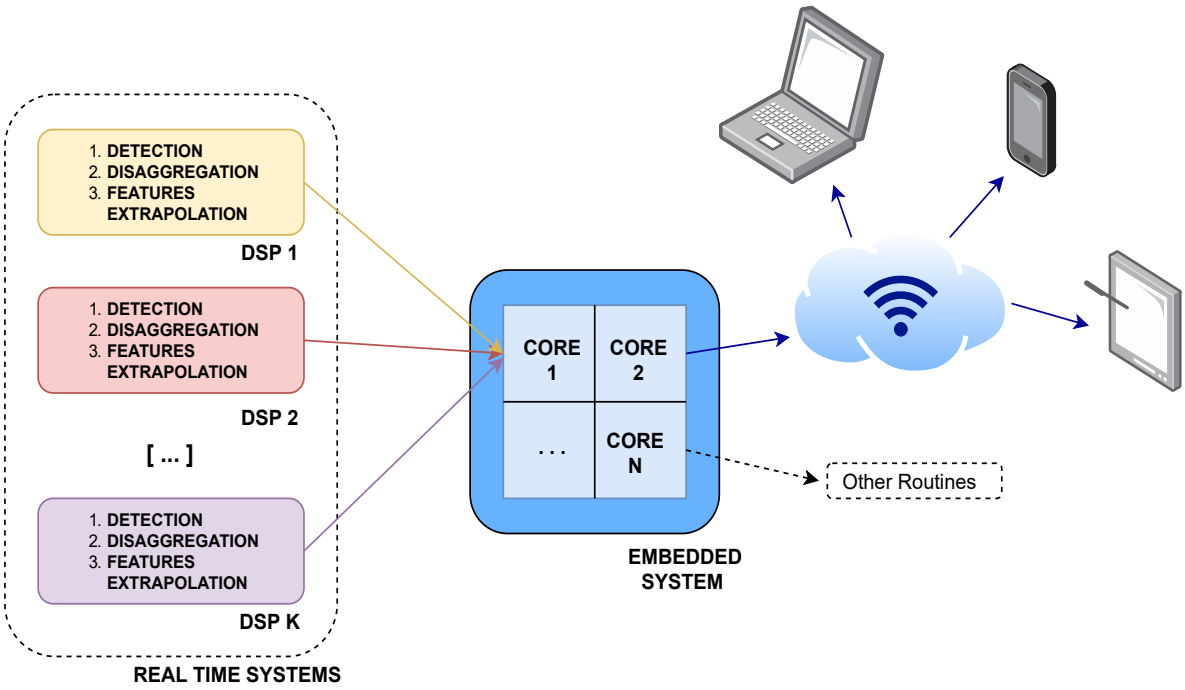
1. Improve detector performance, maybe using ensemble of detectors, to have a more robust and generalized system;
2. Build a new features dataset, trying to use new features in a way to increase the amount of information for classification purposes;

3. Exploit GPU capabilities for higher speed and computation efficiency;
4. Find a way to let the board analyze multiples signals in parallel.

While for point 1 we previously said that the approach mentioned in Renaux *et al.* (2018b) could be a further improvement, what can be done for point 2 is, apart from finding new features, try to realize more measurements of aggregated waveforms. The latter, must always include the use of a disaggregation process and this will enhance the information content of datasets, since we will even have data from appliance on their disaggregated version. This could be achieved producing new datasets where is fundamental to use an approach similar to LIT dataset, where turn-on and off events were always recorded and labeled with the load they belongs to. For example, Lab dataset was built just with single appliance signals, since there where no way of recognizing a priori where each appliance was located inside the recorded waveforms.

Moreover, simulations over the NVIDIA TX1 board gave some hints about a possible implementation of a NILM module accordingly to future works mentioned in point 3 and 4, since we proved that the proposed algorithm runs smoothly without considerable computational requirements. As a matter of fact, multiple embedded systems as the ones provided by NVIDIA, Raspberry or STM microelectronics are already provided with optimized machine learning modules and multiple cores architectures, all with affordable costs. Therefore, we might think about a structure were one core is dedicated for the machine learning processes (whose execution could be optimized through a GPU) while others could manage other routines, as Wi-fi or Bluetooth communications, if our aim is to communicate with a phone application. Then another core could be dedicated to the communication with different real time systems, such as digital signal processors (DSP), whose assignment could be to realize all the signal processing required for the detection, disaggregation and feature extrapolation as shown in Figure 56. In this way, we could design a centralize system able to manage alone multiple external units, each of them in charge of processing power information of the assigned monitored environment.

Figure 56 – Scheme of the possible implementation of a NILM module: the cores of an embedded device, with its own operative system, could be partitioned to work with different tasks, as the communication with some DSPs or the exchange of loads and energy consumption information with other devices.



Source: Own Authorship

APPENDIX

APPENDIX A – PYTHON CODES

.1 DETECTOR

```
import numpy as np

def detector(signal, nsamples, window=4):

   temp = 0
   ts = [] # turn-on instants
   te = [] # turn-off instants

   # create envelope of signal
   peak, _ = find_peaks( signal, distance = nsamples )
   env   = np.array( signal[peak] )

   signal_std = np.zeros( len(env) ) # variance vector
   signal_mean = np.zeros( len(env) ) # mean vector

   # initialize variance and mean vector
   signal_std [0:window] = np.std( env[0:window] )
   signal_mean[0:window] = np.mean( env[0:window] )

   for i in np.arange( window, len(env)-window , window ):
      for j in np.arange( 0, window ):
         signal_mean[i+j] = signal_mean[i+j-1] +
            (env[i+j]-env[i+j-window])/window
         signal_std [i+j] = np.sqrt( (1/window**2) *
            signal_std[i+j-1]**2 + 1/(window-1) *
            (env[i+j]-signal_mean[i+j])**2)

   std_grad = np.gradient( signal_std )
   env_grad = np.gradient( env )
```

```
tmin = 2.0 # empirical low threshold
tmax = 6.0 # empirical high threshold

for i in np.arange( 0,len(env) ):

   if ( signal_std[i] > tmax and std_grad[i] > 0 and env_grad[i]
      > 0 and temp == 0 ) :
      temp = 1
      ts.append(peak[i])
   if ( signal_std[i] < tmin and std_grad[i] < 0 and temp == 1 ) :
      te.append(peak[i])
      temp = 0

ts = np.array( ts )
te = np.array( te )

return ts, te, env, signal_std
```

Note that, in the first *for* cycle, the variance and mean vector ($signal\_std$ and $signal\_mean$, respectively) are defined following eq.(1)-(2) as suggested in (Nait Meziane *et al.*, 2017). *tmin* and *tmax* represent two empirical thresholds defined through tests of HAND algorithm over load signals. The reason why two thresholds were exploited instead of one is that this allow to reduce the detection interval: while *tmin* is used to turn-on the detector, *tmax* is used for its turn-off enabling detection of adjacent transients. As a matter of fact, tests show that when two transients are close to each other the resulting variance sometimes remains over *tmin*, which end up in two transients being detected together. With the help of *tmax* we prematurely switch off HAND algorithm ensuring correct detection.

In section 5.1 we discussed about HAND problems and the use of a modified detector for LIT Dataset, which uses the appliances turn on instances already provided by the dataset manufacturers. From a practical point of view, this means basically substitute those instances in the algorithm instead of using *tmin*.

.2 MOVING AVERAGE FILTER

```
import pandas as pd
```

```
def moving_average(signal,window_size):
    numbers_series = pd.Series(signal)
    windows = numbers_series.rolling(window_size)
    moving_averages = windows.mean()

    moving_averages_list = moving_averages.tolist()
    without_nans = moving_averages_list[window_size - 1:]
    return without_nans
```

While the moving average filter is very easy to understand, during the process of algorithm validation we implemented even another type of low pass filter called Butter Low Pass filter, which was useful since it allow to choose the order and the cutoff frequency of the filter, thus the slope of its stop-band and the amount of harmonic content that should be removed. The code is the following:

```
from scipy.signal import butter,filtfilt

def butter_lowpass_filter(data, cutoff, fs, order):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    # Get the filter coefficients
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    y = filtfilt(b, a, data)
    return y
```

.3 PEAK

```
import numpy as np

def peak(power):
    peak_ = np.max(power)
    return peak_
```

.4 MEDIAN

```
def median(power):
    median_ = np.median(power)
    return median_
```

.5 COVARIANCE

```
def covariance(current,voltage):
    cov_ = np.cov(current,voltage) #returns a matrix 2x2
    cov_ = cov_[0,1]
    return cov_
```

The reason why a value in the right diagonal was taken is because the function $np.cov$ returns a matrix 2x2 where the left diagonal represents the square variance of the relative element .

.6 STEADY STATE

```
import numpy as np
from scipy.signal import find_peaks

def P_envelope( power, nsamples ):
    a      = power
    peak , _ = find_peaks( a, distance = nsamples )
    power_p_env = a[ peak ]
    return power_p_env

def Steadystate_mean( power, nsamples ):
    env    = P_envelope( power, nsamples )
    steady_ = np.mean( env )
    return steady_
```

.7 FFT

```
def Harmonics( signal, fs, f0 ):
    N    = len( signal )
```

```
fstep = fs/N; #freq interval
f    = np.linspace( 0, ( N-1 )*fstep, N )

X    = np.fft.fft( signal )
X_mag = np.abs( X )/N

f_plot      = f[ 0:int( N/2+1 ) ]
X_mag_plot  = 2*X_mag[ 0:int( N/2+1 ) ]
X_mag_plot[ 0 ] = X_mag_plot[ 0 ]/2

mag_harm = np.zeros( int( round( f_plot[ -1 ] / f0 ) ) ) # array
   of harmonics magnitudes
harmonics_index = np.array( [ round( i*f0*len( f_plot )/f_plot[
   -1 ] ) for i in np.arange( round( f_plot[ -1 ] / f0 ) ) ] )

dist = 2 #round( 10/fstep )
if dist == 0:
   dist = 2

for ind, j in enumerate( harmonics_index ):
   if ind == 0:
      mag_harm[ ind ] = X_mag_plot[ 0 ]
   else:
      mag_harm[ ind ] = np.max( X_mag_plot[ int( j-dist ):int(
         j+dist ) ] )


TED = np.sum( mag_harm[ 2::2 ] ) # Even Harmonics Distortion
TOD = np.sum( mag_harm[ 3::2 ] ) # Odd Harmonics Distortion
MAG = mag_harm[ 1:6, ]     # Take the first 5 harmonics

return MAG, TED, TOD
```

The FFT function provided by the numpy library of Python takes into account the sampling frequency of the signals which is provided by fs in the algorithm. On the other hand, $f0$ is the fundamental

frequency, which was always 50Hz in our case.

Note that the final algorithm was intended to work for the latest experiments on real loads signals' waveforms: in those case just the first fifth harmonics were used, while the remaining harmonic content was grouped in two features called *TED* and *TOD* in the upper function.

.8 ACTIVE POWER

```
def active_p(power):
    P = np.mean(power)
    return P
```

.9 REACTIVE POWER

```
def reactive_p(current,voltage,power):
    P = active_p(power)
    S = apparent_p(current,voltage)
    theta = np.arccos(P/S)
    Q = S * np.sin(theta)
    return Q
```

.10 APPARENT POWER

```
def apparent_p(current,voltage):
    Irms = np.sqrt(np.mean(current**2))
    Vrms = np.sqrt(np.mean(voltage**2))
    S    = Irms*Vrms
    return S
```

.11 EXTRAPOLATE CYCLE / CURRENT SPAN

```
import numpy as np
from scipy.signal import find_peaks
```

```
def build_VI_cycle( c_signal, v_signal, nsamples, fs, f0):
   N_cycles = int( len( c_signal )/nsamples )
   d      = np.diag( np.ones( nsamples ) )

   curr = c_signal[ 0:nsamples*N_cycles ]
   volt = v_signal[ 0:nsamples*N_cycles ]

   Av_matrix = d
   for i in np.arange( N_cycles-1 ):
      Av_matrix = np.concatenate(( Av_matrix, d ), axis=1)


   c_cycle = np.matmul( Av_matrix, curr )/( N_cycles )
   v_cycle = np.matmul( Av_matrix, volt )/( N_cycles*np.max( volt ) )

   itc = np.max( c_cycle )-np.min( c_cycle )
   c_cycle = c_cycle/np.max( c_cycle )

   return c_cycle, v_cycle, itc
```

The algorithm, once it knows the length *nsamples* of one cycle of the signal, elaborate an average I and V cycle over the N_CYCLES available on the current (c_signal) and voltage (v_signal) signals passed to the function. In particular, from the average I cycle (c_cycle), the algorithm calculates the current span ITC.

.12 ANGLE / DISTANCE BETWEEN MAXIMUM AND MINIMUM POINT

```
import numpy as np

def Distance_angle_maxminpoint( c_cycle, v_cycle ):
   m = np.argmin( c_cycle )
   M = np.argmax( c_cycle )
```

```
    dpb = ( ( c_cycle[ M ] - c_cycle[ m ] )**2 + ( v_cycle[ M ] -
        v_cycle[ m ] )**2 )**0.5
    angp = np.arctan( ( c_cycle[ M ] - c_cycle[ m ] )/( v_cycle[ M ]
        - v_cycle[ m ] ) )
    return dpb, angp
```

.13 AREA WITH LOOP DIRECTION

```
import numpy as np

def lpa(current_cycle,voltage_cycle):

    lpa_ = np.sum(0.5*(np.array(voltage_cycle[1:-1])
                -np.array(voltage_cycle[0:-2]))*
               (np.array(current_cycle[1:-1])-np.array(current_cycle[0:-2])))

    return lpa_
```

.14 ASYMMETRY

```
from scipy.spatial.distance import directed_hausdorff

def Asymmetry( c_cycle, v_cycle ):
    c_cycle180 = -c_cycle
    v_cycle180 = -v_cycle

    u = np.array( [ c_cycle, v_cycle ] ).transpose()
    v = np.array( [ c_cycle180, v_cycle180 ] ).transpose()

    asy = max(directed_hausdorff(u, v)[ 0 ], directed_hausdorff(v,
        u)[ 0 ])

    return asy
```

.15 LENGTH

```
def Length( c_cycle, v_cycle ):
   lenght = 0
   for i in np.arange( len( c_cycle )-1 ):
      x = v_cycle[ i+1 ] - v_cycle[ i ]
      y = c_cycle[ i+1 ] - c_cycle[ i ]

      lenght += np.linalg.norm( [ x, y ] )

   return lenght
```

.16 MAXIMUM DISTANCE

```
def Maximum_distance( c_cycle, v_cycle ):
   c2 = c_cycle**2
   v2 = v_cycle**2

   md = np.max( ( c2+v2 )**0.5 )

   return md
```

.17 CURVATURE OF MEAN LINE

In the next three features the arguments of the functions will always be (cA, cB, vA, vB).

```
def Curvature_of_mean_line( cA, cB, vA, vB ):
  mean_line = np.zeros( ( len( cA ), 2 ) )

   for i in np.arange( len( cA ) ):
      ip = np.argmin( vB-vA[ i ] )
      xm = 0.5*( vA[ i ]+vB[ ip ] )
      ym = 0.5*( cA[ i ]+cB[ ip ] )
```

```
      mean_line[ i, 0 ] = xm
      mean_line[ i, 1 ] = ym

   r = np.correlate( mean_line[ :, 0 ], mean_line[ :, 1 ] )[ 0 ]

   return r
```

.18 NUMBER OF SELF INTERSECTIONS

```
def Self_intersection( cA, cB, vA, vB ):
   sc = 0
   for i in np.arange( len( cA )-1 ):
      j = i+1
      ip = np.argmin( vB-vA[ i ] )
      jp = np.argmin( vB-vA[ j ] )

      ij = Versor( Vector2D( [ vA[ i ], cA[ i ] ], [ vA[ j ], cA[ j
         ] ] ) )
      iip = Versor( Vector2D( [ vA[ i ], cA[ i ] ], [ vB[ ip ], cB[
         ip ] ] ) )
      ijp = Versor( Vector2D( [ vA[ i ], cA[ i ] ], [ vB[ jp ], cB[
         jp ] ] ) )

      if( np.dot( np.cross( ij, iip ), np.cross( ij, ijp ) ) ):
         sc += 1

   return sc
```

Where the two functions *Vector2D* and *Versor* are respectively:

```
def Vector2D( a, b ):
   x = a[ 0 ] - b[ 0 ]
   y = a[ 1 ] - b[ 1 ]
   vector = [ x, y ]
```

```
    return vector
```

```
def Versor( vector ):
    y = vector/np.linalg.norm( vector )
    return y
```

.19 AREA

```
def Area( cA, cB, vA, vB):
    ar = 0
    for i in np.arange( len( cA )-1 ):
        j = i+1
        ip = np.argmin( vB-vA[ i ] )
        jp = np.argmin( vB-vA[ j ] )

        vdiff = 0.5*np.abs( vA[ j ]-vA[ i ] )
        ar  += vdiff*( np.abs( cB[ ip ] - cA[ i ] ) + np.abs( cB[ jp ]
            - cA[ j ] ) )

    return ar
```

.20 FFT-BASED THRESHOLD

```
import numpy as np

fft = np.fft.fft(variance)
threshold = np.mean(np.abs(fft)/len(variance)*2)
```

.21 LOCAL PEAK-BASED THRESHOLD

```
import numpy as np
from scipy.signal import argrelextrema
```

```
c = argrelextrema(variance,np.greater) # gives the index of local
    maxima
threshold = np.mean(variance[c])
```

.22 CONVERSION FLAC FILES (COOLL)

Here follows the algorithm for correctly extracting the files, from a $.flac$ format to a numpy array in python. Code :

```
import numpy as np
import soundfile as sf

def
    loadfiles(current_audiofiles,voltage_audiofiles,c_scalef,v_scalef):

    current=[]
    voltage=[]

    for i in np.arange(len(current_audiofiles)): #np.arange(20):
        audio, freq = sf.read(current_audiofiles[i])
        current.append([audio])

    for i in np.arange(len(voltage_audiofiles)):
        audio, freq = sf.read(voltage_audiofiles[i])
        voltage.append([audio])

    current=np.array(current)
    voltage=np.array(voltage)

    #scale factor processing
    for i in np.arange(len(current_audiofiles)):
        current[i,0]*=c_scalef[i]
        voltage[i,0]*=v_scalef[i]
```

```
    time=(np.arange(0,len(audio))/freq)
    current = current.reshape(840,-1)
    voltage = voltage.reshape(840,-1)

    return current,voltage,time,freq
```

This algorithm extrapolates all the $.flac$ signals giving $current$ and $voltage$ which are two vectors of dimension [840,599996], a time vector that contains the signals instants of dimension [1,599996] and $freq = 100000$ that gives the sampling frequency .

.23 CONVERSION MAT FILES (LIT)

```
import scipy.io as sp

mat = sp.loadmat('your/address/to/the/waveform.mat')
current = mat['iShunt'].reshape(1,-1)
current = curr[0,:]
voltage = mat['vGrid'].reshape(1,-1)
voltage = volt[0,:]

start_instants = []
for i,el in enumerate(mat['events_r']):
    if el == 1 :
        start_instants.append(i)
```

.24 SEQUENTIAL BACKWARD SELECTION

Here follows the python code of the function (RASCHKA; MIRJALILI, 2019) :

```
from sklearn.base import clone
from itertools import combinations

class SBS():
    def __init__(self, estimator, k_features, scoring=accuracy_score,
```

```
            test_size=0.20, random_state=1):
        self.scoring = scoring
        self.estimator = clone(estimator)
        self.k_features = k_features
        self.test_size = test_size
        self.random_state = random_state

    def fit(self, X, y):

        X_train, X_test, y_train, y_test = train_test_split(X, y,
            test_size=self.test_size,
                        random_state=self.random_state)

        dim = X_train.shape[1]
        self.indices_ = tuple(range(dim))
        self.subsets_ = [self.indices_]
        score = self._calc_score(X_train, y_train,
                           X_test, y_test, self.indices_)
        self.scores_ = [score]

        while dim > self.k_features:
            scores = []
            subsets = []

            for p in combinations(self.indices_, r=dim - 1):
                score = self._calc_score(X_train, y_train,
                                         X_test, y_test, p)
                scores.append(score)
                subsets.append(p)

            best = np.argmax(scores)
            self.indices_ = subsets[best]
            self.subsets_.append(self.indices_)
            dim -= 1
```

```
            self.scores_.append(scores[best])
        self.k_score_ = self.scores_[-1]

        return self

    def transform(self, X):
        return X[:, self.indices_]

    def _calc_score(self, X_train, y_train, X_test, y_test, indices):
        self.estimator.fit(X_train[:, indices], y_train)
        y_pred = self.estimator.predict(X_test[:, indices])
        score = self.scoring(y_test, y_pred)
        return score
```

.25 PRINCIPAL COMPONENTS ANALYSIS

Despite what we see for SBS, here fortunately the PCA class is already present in the scikit-learn python's library, thus its implementation will be really easy by using the following code :

```
from sklearn.decomposition import PCA
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA

for i in np.arange(6,len(df.columns)-1):
    pipe = make_pipeline(StandardScaler(),PCA(n_components=i), knn )
    pipe.fit(X_train, y_train)
    print(i, 'Test accuracy %.3f' % pipe.score(X_test,y_test))
```

In the previous code, an example with the K-Neighbors classifier was made. As you can see, the model evaluation here was done by simply reduce the number of principal components, in a way to observe if we can obtain better classification accuracy by reducing the dimension of the new subspace,

therefore operating features extraction.

.26 GRID SEARCH

```
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

svm = SVC(kernel='linear', C=10000.0, random_state=0)
pipe_svm = make_pipeline(StandardScaler(),svm)

param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0,
   1000.0,10000.0]

param_grid = [{'svc__C': param_range,
           'svc__kernel': ['linear'],
           'svc__random_state':[0]},
          {'svc__C': param_range,
           'svc__gamma': param_range,
           'svc__kernel': ['rbf'],
           'svc__random_state':[1]}]

gs = GridSearchCV(estimator=pipe_svm,
             param_grid=param_grid,
             scoring='accuracy',
             refit=True,
             cv=10,
             n_jobs=-1)

gs = gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_params_)
clf = gs.best_estimator_
```

```
print('Test accuracy: %.3f' % clf.score(X_test, y_test))
```

In the previous code the SVM classifier was evaluated. As you can see here the classifier was evaluated over three parameters : the regulation parameter C, the type of kernel used and the value of the random state (RASCHKA; MIRJALILI, 2019).

.27 DATASET PREPROCESSING

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split

#Substitution of NaN
imr = SimpleImputer(missing_values=np.nan,strategy='mean')
imr=imr.fit(df.values)
imputed_data=imr.transform(df.values)
df=pd.DataFrame(data=imputed_data)

#Label encoding
class_mapping= {label : idx for idx,label in
   enumerate(np.unique(df['Appliance']))}
df['Appliance']=df['Appliance'].map(class_mapping)

#Dataset splitting
X,y = df.iloc[:,1:].values, df.iloc[:,0].values
X_train, X_test, y_train, y_test
   =train_test_split(X,y,test_size=0.20,random_state=0)
```

In the code above df is the dataset being processed. In particular, when df is split into the training and the test dataset $test_size = 0.20$ means that the 20% of the dataset will be used for the test procedure; with the $random\_state$ parameter, instead, we provide a fixed random seed for the internal pseudo-random number generator that is used for shuffling the dataset before splitting.

.28 CLASSIFIER MODEL SAVING AND LOADING

```
from sklearn.ensemble import RandomForestClassifier
import joblib

forest = RandomForestClassifier( n_components=22, n_jobs=-1,
   random_state=0 )
forest.fit(X_train,y_train)
#save model
joblib.dump(forest, 'random_forest_classifier') # (classifier,
   file_name)

#import model
forest = joblib.load('your/path/to/random_forest_classifier')
```

.29 COOLL DATASET TEST CODE

```
import joblib
import get_file as gf

lr = joblib.load('logistic_regression')
std = joblib.load('std_scaler')
test = gf.x
class_labels = gf.class_mapping

prediction = lr.predict(std.transform(test))

for label,number in class_labels.items():
  if prediction == number :
    print("Appliance name: ", label)
```

where get_file is a function that elaborates the $.flac$ file of the waveform the system has been fed, and calculates its features saving them into a vector x; this function also provide a dictionary called

class_mapping for giving as an output the label of the detected load. Notice that, since the example refers to logistic regression, even the model of the standard scaler has been imported, after being previously fitted with training data and saved .

.30 LIT DATASET TEST CODE

```
import scipy.io as sp
import function as fn
import align_detect as adb
import numpy as np
import pandas as pd
import joblib

forest = joblib.load('random_forest')

class_labels = {'A0':0,'B0':1,'C0':2,'D0':3,'E0':4,'F0':5,'G0':6,
    'H0':7,'I0':8,'J0':9,'K0':10,'L0':11,'M0':12,'N0':13,'O0':14,
    'P0':15,'Q0':16,'R0':17,'S0':18,'T0':19,'U0':20,'V0':21,
    'W0':22,'X0':23,'Y0':24,'Z0':25}

def test(file) :

   mat = sp.loadmat(file)
   curr = mat['iShunt'].reshape(1,-1)
   curr = curr[0,:]
   volt = mat['vGrid'].reshape(1,-1)
   volt = volt[0,:]
   instants = []
   for i,el in enumerate(mat['events_r']):
      if el == 1 :
         instants.append(i)

   te,_,_ = adb.mod_detector(curr,instants)
```

```
for j in np.arange(len(instants)):

    if j == 0 :
        if te[j,1] - instants[j] < 1024 or
            (te[j,1]-instants[j])>instants[j] :
            It,Vt = curr[instants[j]:instants[j]+20*256] ,
                volt[instants[j]:instants[j]+20*256]
            Is,Vs = curr[te[j,1]+22*256:te[j,1]+42*256] ,
                volt[te[j,1]+22*256:te[j,1]+42*256]

        else :
            It,Vt = curr[instants[j]:te[j,1]],
                volt[instants[j]:te[j,1]]
            Is,Vs = curr[te[j,1]+2*256:te[j,1] + 22*256],
                volt[te[j,1]+2*256:te[j,1] + 22*256]

    else :
        if te[j,1] - instants[j] < 1024 or
            (te[j,1]-instants[j])>instants[j]:
            It,Vt,_ =
                adb.buffer(instants[j],instants[j]+20*256,curr,volt,
                    part='t')
            Is,Vs,_ =
                adb.buffer(instants[j]+22*256,instants[j]+42*256,curr,volt,
                    part='s')

        else :
            It,Vt,_ =
                adb.buffer(instants[j],te[j,1],curr,volt,part='t')
            Is,Vs,_ =
                adb.buffer(instants[j],te[j,1],curr,volt,part='s')

    test_s = fn.features_steady(Is,Vs,Is*Vs)
```

```
    test_t = fn.features_transient(It,Vt,It*Vt)

    test = np.concatenate((test_t,test_s), axis=1 )
    prediction = forest.predict(test)

    for label,number in class_labels.items():
       if prediction == number :
          print("Appliance", j+1, " name: ", label)
```

The function $test$ gets the $.mat$ file as an input and extrapolate the current, the voltage signals and, also, the turn-on instants for using the modified detector, this one imported from the $.py$ file called $align\_detect$.
Then the transient detection is performed: note that if the detected transient is too short or too long, the algorithm decides to take an interval of 20 cycles of the signal. Then, after detection, disaggregation is made (in a sequence of loads, not needed for the first one) always calling a function from the $align\_detect$ file.
Finally features are calculated with the help of $function.py$, where all the features' algorithms explained in Chapter 3 are contained, and then the RF model elaborates its prediction.

.31 DISAGGREGATION

```
def buffer(t_start,t_end,c_signal,v_signal, part, n_cycles = 10):

   phi = (t_start - t_end)%256

  if part == 's' :

    temp_c = c_signal[t_start - (n_cycles+2)*256 : t_start -2*256]
    curr_ = c_signal[ t_end +2*256 + phi: t_end+ (n_cycles+2)*256 +
       phi ]

    x = np.argmax(np.correlate(curr_,temp_c,mode = 'full'))%256
    curr_ = c_signal[ t_end +2*256 + x + phi : t_end+
       (n_cycles+2)*256 + x + phi ] - temp_c
```

```
    volt_ = v_signal[ t_end +2*256 + x + phi :
        t_end+(n_cycles+2)*256 + x + phi ]

  if part == 't' :

    diff = t_end - t_start
    temp_c = c_signal[t_start - (2 + int(diff/256))*256 : t_start -
        (2 + int(diff/256))*256 + diff ]
    curr_ = c_signal[ t_start : t_end ] , c_signal[ t_start : t_end
        ]

    x = np.argmax(np.correlate(curr_,temp_c,mode = 'full'))%256
    curr_ = c_signal[ t_start + x + phi : t_end + x + phi ] - temp_c
    volt_ = v_signal[ t_start + x + phi : t_end + x + phi]

  curr_ = np.array(moving_average(curr_,4))
  volt_ = np.array(moving_average(volt_,4))

  return curr_,volt_
```

The function use to make disaggregation is called $buffer$ and takes as parameters the start and end instants of the transient, the whole current and voltage signals that has to be analyzed, a parameter called $part \in [s,t]$ to decide if isolate the transient or the steady state of the signal, and n_cycles, which value correspond to the length of the steady state interval to extrapolate.

The most important part of the algorithm, to be sure that isolation has been made correctly, is review $temp$ and the current signal phase, to subtract one from each other correctly . Note that no subtraction will be made over the voltage signal, since the voltage line is very robust and its signal only slightly changes when some appliances turned on.

To ensure phase alignment, first of all the variable phi is calculated. This variable takes into account the number of samples per cycles and ensures a correct alignment from a sample point view, by granting that the start instants of both $temp$ and $curr\_$ have the same initial phase.

This should be enough for alignment, but actually is not. Infact appliances with different power factors can introduce, when they turn on, a phase component in the signal. To compensate this additional phase a further shift component x has been added.

The variable x has been calculated as the index who maximizes the correlation between $curr\_$ and $temp$.

Correlation is infact a tool that we can use to look for a specific shape of, for example, a signal inside another signal. In this process the two signals will be overlapped, shifted sample by sample and the correlation will tracks qualitatively how much the two signals line up.

All the source codes has been made available on the GitHub online repository :

https://github.com/giobraglia/Signal-Processing-Features

.32 FINAL LAB TESTS RESULTS

In the next pages the final results obtained for the evaluation of the proposed NILM algorithm in MElting Lab of Unimore are presented. For each group of waveform all the predicted labels for each classifier were recorded. Each group is marked by a "Waveform number", to retrieve the waveform in the Lab Dataset, and the "GROUP"number, which indicates the appliances recorded in the waveform. In particular, the cells highlighted in red represents the errors; at the end of each columns the number of errors is resumed.

CLASSIFICATION TEST

GROUP :	1LA0		
Waveform number:	10001		
LR	SVM	KNN	RF
LA0	LA0	LA0	LA0
LA0	LA0	LA0	LA0
LA0	LA0	LA0	LA0
0	0	0	0

GROUP :	1LA0		
Waveform number:	10002		
LR	SVM	KNN	RF
LA0	LA0	LA0	LA0
LA0	LA0	LA0	LA0
LA0	LA0	LA0	LA0
LA0	LA0	LA0	LA0
0	0	0	0

GROUP :	1LA0		
Waveform number:	10003		
LR	SVM	KNN	RF
LA0	LD0	LD0	LD0
LA0	LA0	LA0	LA0
LA0	LA0	LA0	LA0
LA0	LA0	LA0	LA0
0	1	1	1

GROUP :	1LA0		
Waveform number:	10004		
LR	SVM	KNN	RF
LA0	LA0	LA0	LA0
LA0	LA0	LA0	LA0
0	0	0	0

GROUP :	1LB0		
Waveform number:	10005		
LR	SVM	KNN	RF
LB0	LB0	LB0	LB0
LB0	LC0	LB0	LB0
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LA0	LC0	LB0	LB0
LB0	LB0	LB0	LB0
LB0	LB0	LB0	LC0
LB0	LC0	LB0	LB0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LD0	LC0	LB0	LB0
LA0	LC0	LB0	LC0
LB0	LB0	LB0	LB0
6	0	0	0

GROUP :	1LB0		
Waveform number:	10006		
LR	SVM	KNN	RF
LB0	LC0	LB0	LB0
LB0	LC0	LB0	LB0
LA0	LB0	LB0	LB0
LB0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LB0	LC0	LB0	LC0
LB0	LB0	LB0	LB0
LB0	LB0	LB0	LB0
LB0	LC0	LB0	LB0
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LB0	LC0	LB0	LB0
LB0	LB0	LB0	LB0
LB0	LB0	LB0	LB0
3	0	0	0

GROUP :	1LB0		
Waveform number:	10007		
LR	SVM	KNN	RF
LB0	LB0	LB0	LB0
LB0	LC0	LB0	LB0
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LB0	LC0	LB0	LB0
LB0	LC0	LB0	LB0
LB0	LB0	LB0	LB0
LB0	LB0	LB0	LB0
LB0	LB0	LB0	LB0
LB0	LC0	LB0	LB0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LB0
LF0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
4	0	0	0

GROUP :	1LB0		
Waveform number:	10008		
LR	SVM	KNN	RF
LB0	LC0	LB0	LB0
LA0	LC0	LB0	LB0
LA0	LC0	LB0	LC0
LA0	LB0	LB0	LB0
LB0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LA0	LB0	LB0	LC0
LA0	LC0	LB0	LB0
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LB0
LB0	LB0	LB0	LB0
LB0	LB0	LB0	LB0
5	0	0	0

GROUP :	1LC0		
Waveform number:	10009		
LR	SVM	KNN	RF
LB0	LC0	LB0	LB0
LB0	LC0	LB0	LB0
LB0	LC0	LB0	LC0
LB0	LC0	LD0	LB0
LB0	LB0	LB0	LB0
LB0	LC0	LC0	LC0
LF0	LC0	LC0	LB0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LB0
LA0	LC0	LD0	LB0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LB0	LC0	LB0	LB0
LA0	LB0	LD0	LB0
LB0	LB0	LB0	LB0
8	0	3	0

GROUP :	1LC0		
Waveform number:	10010		
LR	SVM	KNN	RF
LB0	LC0	LB0	LB0
LB0	LC0	LB0	LB0
LA0	LC0	LB0	LB0
LA0	LC0	LB0	LB0
LA0	LC0	LB0	LB0
LD0	LC0	LB0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LF0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LB0	LB0	LB0	LB0
LB0	LB0	LB0	LB0
LB0	LC0	LB0	LB0
LB0	LC0	LB0	LB0
9	0	0	0

GROUP :	1LC0		
Waveform number:	10011		
LR	SVM	KNN	RF
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LB0	LC0	LB0	LB0
LB0	LC0	LB0	LB0
LB0	LC0	LB0	LB0
LB0	LB0	LB0	LB0
LB0	LC0	LB0	LC0
LB0	LC0	LB0	LC0
LB0	LB0	LB0	LB0
LB0	LB0	LB0	LB0
LB0	LB0	LB0	LB0
LB0	LB0	LB0	LB0
LB0	LC0	LB0	LB0
LA0	LC0	LB0	LB0
LB0	LC0	LC0	LC0
1	0	0	0

GROUP :	1LC0		
Waveform number:	10012		
LR	SVM	KNN	RF
LD0	LC0	LB0	LC0
LB0	LB0	LB0	LB0
LB0	LC0	LB0	LB0
LB0	LC0	LB0	LB0
LB0	LC0	LB0	LB0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LF0
LA0	LC0	LD0	LC0
LF0	LC0	LC0	LF0
LF0	LC0	LC0	LC0
LD0	LC0	LB0	LB0
LD0	LC0	LB0	LB0
LA0	LC0	LB0	LB0
LB0	LC0	LB0	LB0
9	0	0	2

GROUP :	1LD0		
Waveform number:	10013		
LR	SVM	KNN	RF
LD0	LD0	LD0	LD0
LA0	LD0	LD0	LD0
LA0	LA0	LD0	LD0
LA0	LD0	LD0	LD0
LD0	LD0	LD0	LD0
LD0	LD0	LD0	LD0
LA0	LD0	LD0	LD0
LA0	LA0	LD0	LD0
5	2	0	0

GROUP :	1LD0		
Waveform number:	10014		
LR	SVM	KNN	RF
LA0	LA0	LD0	LD0
LA0	LA0	LD0	LD0
LA0	LA0	LD0	LD0
LA0	LA0	LD0	LD0
LA0	LA0	LD0	LD0
LA0	LA0	LD0	LD0
6	6	0	0

GROUP :	1LD0		
Waveform number:	10015		
LR	SVM	KNN	RF
LD0	LD0	LD0	LD0
LD0	LD0	LD0	LD0
LA0	LD0	LD0	LD0
LD0	LD0	LD0	LD0
LA0	LD0	LD0	LD0
LA0	LA0	LD0	LD0
LD0	LA0	LD0	LD0
LD0	LA0	LD0	LD0
LA0	LA0	LD0	LD0
4	4	0	0

GROUP :	1LD0		
Waveform number:	10016		
LR	SVM	KNN	RF
LD0	LD0	LD0	LD0
LD0	LD0	LD0	LD0
LD0	LD0	LD0	LD0
LD0	LD0	LD0	LD0
LD0	LD0	LD0	LD0
LD0	LD0	LD0	LD0
LD0	LD0	LD0	LD0
LA0	LD0	LD0	LD0
LD0	LD0	LD0	LD0
LA0	LA0	LD0	LD0
LD0	LD0	LD0	LD0
LD0	LD0	LD0	LD0
2	1	0	0

GROUP :	1LF0		
Waveform number:	10017		
LR	SVM	KNN	RF
LA0	LA0	LD0	LD0
LA0	LA0	LD0	LD0
LD0	LD0	LD0	LD0
LA0	LF0	LF0	LF0
4	3	3	3

GROUP :	1LF0		
Waveform number:	10018		
LR	SVM	KNN	RF
LA0	LF0	LF0	LF0
LA0	LD0	LD0	LD0
LD0	LD0	LD0	LD0
LA0	LA0	LD0	LD0
4	3	3	3

GROUP :	1LF0		
Waveform number:	10019		
LR	SVM	KNN	RF
LA0	LF0	LF0	LF0
LD0	LD0	LD0	LD0
LA0	LA0	LD0	LD0
3	2	2	2

GROUP :	1LF0		
Waveform number:	10020		
LR	SVM	KNN	RF
LA0	LF0	LF0	LF0
1	0	0	0

GROUP :	2LA0LB0		
Waveform number:	20001		
LR	SVM	KNN	RF
LA0	LA0	LA0	LA0
LA0	LD0	LD0	LB0
LA0	LA0	LA0	LA0
LA0	LD0	LD0	LD0
LA0	LA0	LD0	LB0
LA0	LD0	LD0	LB0
LA0	LA0	LA0	LA0
LA0	LC0	LD0	LB0
LA0	LA0	LD0	LB0
LA0	LA0	LA0	LA0
LA0	LA0	LA0	LA0
0	3	4	1

GROUP :	2LA0LB0		
Waveform number:	20002		
LR	SVM	KNN	RF
LA0	LC0	LD0	LB0
LA0	LC0	LD0	LB0
LA0	LA0	LA0	LA0
LA0	LD0	LD0	LB0
LA0	LD0	LD0	LB0
LA0	LA0	LD0	LB0
LA0	LA0	LD0	LB0
LA0	LA0	LD0	LB0
LA0	LC0	LD0	LB0
LB0	LB0	LA0	LA0
LA0	LA0	LA0	LA0
0	2	8	0

GROUP :	2LA0LB0		
Waveform number:	20003		
LR	SVM	KNN	RF
LB0	LB0	LA0	LA0
LA0	LA0	LA0	LA0
LA0	LD0	LD0	LD0
LA0	LC0	LD0	LB0
LA0	LC0	LD0	LB0
LA0	LA0	LA0	LA0
LA0	LA0	LA0	LA0
LA0	LC0	LC0	LB0
LA0	LC0	LD0	LB0
LA0	LA0	LD0	LB0
0	1	5	1

GROUP :	2LA0LB0		
Waveform number:	20004		
LR	SVM	KNN	RF
LA0	LD0	LD0	LB0
LA0	LA0	LA0	LA0
LA0	LA0	LD0	LB0
LB0	LB0	LD0	LD0
LA0	LC0	LB0	LB0
LA0	LA0	LD0	LB0
LB0	LC0	LA0	LA0
LB0	LC0	LD0	LB0
LA0	LD0	LD0	LC0
0	2	3	1

GROUP :	2LA0LC0		
Waveform number:	20005		
LR	SVM	KNN	RF
LA0	LC0	LD0	LB0
LA0	LA0	LD0	LB0
LA0	LC0	LC0	LB0
LA0	LC0	LC0	LB0
LA0	LC0	LC0	LC0
LA0	LA0	LA0	LA0
LA0	LD0	LD0	LB0
LA0	LC0	LD0	LB0
LB0	LB0	LD0	LD0
LA0	LC0	LD0	LB0
LA0	LA0	LA0	LA0
0	1	6	1

GROUP :	2LA0LC0		
Waveform number:	20006		
LR	SVM	KNN	RF
LA0	LC0	LC0	LB0
LA0	LA0	LA0	LD0
LA0	LA0	LA0	LA0
LB0	LB0	LD0	LA0
LB0	LC0	LD0	LB0
LA0	LC0	LD0	LB0
LA0	LA0	LA0	LA0
LB0	LB0	LA0	LA0
LA0	LB0	LD0	LB0
LA0	LC0	LD0	LB0
0	0	5	1

GROUP :	2LA0LC0		
Waveform number:	20007		
LR	SVM	KNN	RF
LA0	LC0	LD0	LC0
LA0	LA0	LD0	LC0
LA0	LC0	LC0	LB0
LA0	LA0	LA0	LA0
LA0	LA0	LA0	LA0
LB0	LB0	LD0	LA0
LA0	LC0	LD0	LB0
LA0	LC0	LD0	LB0
LA0	LA0	LA0	LA0
LA0	LC0	LD0	LB0
LA0	LB0	LD0	LB0
0	0	7	0

GROUP :	2LA0LC0		
Waveform number:	20008		
LR	SVM	KNN	RF
LA0	LC0	LD0	LD0
LA0	LA0	LD0	LD0
LA0	LA0	LD0	LB0
LA0	LA0	LA0	LB0
LA0	LC0	LD0	LB0
LA0	LA0	LD0	LB0
LA0	LC0	LD0	LB0
LA0	LC0	LD0	LB0
LA0	LA0	LD0	LB0
LA0	LC0	LD0	LA0
0	0	9	2

GROUP :	2LA0LD0		
Waveform number:	20009		
LR	SVM	KNN	RF
LB0	LB0	LD0	LD0
LB0	LB0	LD0	LD0
LA0	LD0	LD0	LB0
LA0	LD0	LD0	LB0
LA0	LA0	LD0	LB0
LA0	LA0	LA0	LA0
LA0	LB0	LD0	LB0
LA0	LA0	LA0	LA0
2	3	0	4

GROUP :	2LA0LD0		
Waveform number:	20010		
LR	SVM	KNN	RF
LA0	LA0	LA0	LA0
LA0	LA0	LA0	LA0
LB0	LC0	LA0	LA0
LA0	LA0	LA0	LA0
LB0	LB0	LA0	LA0
LA0	LD0	LD0	LB0
2	2	0	1

GROUP :	2LA0LD0		
Waveform number:	20011		
LR	SVM	KNN	RF
LA0	LA0	LD0	LB0
LA0	LA0	LA0	LA0
LA0	LD0	LD0	LB0
LB0	LB0	LD0	LD0
LA0	LD0	LD0	LB0
1	1	0	2

GROUP :	2LA0LD0		
Waveform number:	20012		
LR	SVM	KNN	RF
LA0	LD0	LD0	LB0
LA0	LA0	LA0	LA0
LA0	LA0	LD0	LB0
LA0	LD0	LD0	LB0
0	0	0	3

GROUP :	2LB0LD0		
Waveform number:	20013		
LR	SVM	KNN	RF
LB0	LC0	LB0	LB0
LB0	LC0	LB0	LC0
LA0	LC0	LB0	LB0
LB0	LB0	LB0	LB0
LF0	LF0	LA0	LD0
LF0	LC0	LC0	LD0
LF0	LC0	LC0	LF0
LF0	LC0	LC0	LF0
LB0	LB0	LD0	LD0
LB0	LC0	LB0	LB0
LD0	LC0	LB0	LD0
LF0	LD0	LD0	LD0
LF0	LC0	LC0	LF0
LF0	LC0	LC0	LC0
LB0	LC0	LB0	LB0
LA0	LD0	LD0	LD0
LB0	LC0	LC0	LB0
8	1	1	2

GROUP :	2LB0LD0		
Waveform number:	20014		
LR	SVM	KNN	RF
LB0	LB0	LB0	LB0
LD0	LC0	LB0	LB0
LD0	LD0	LD0	LD0
LF0	LC0	LC0	LD0
LB0	LC0	LB0	LB0
LD0	LD0	LD0	LD0
LB0	LC0	LB0	LB0
LD0	LD0	LD0	LD0
LF0	LC0	LC0	LD0
LF0	LF0	LC0	LC0
LA0	LC0	LB0	LB0
LD0	LD0	LD0	LD0
LB0	LC0	LB0	LB0
LB0	LC0	LC0	LC0
LF0	LF0	LC0	LC0
LD0	LC0	LB0	LC0
LD0	LB0	LD0	LD0
5	2	0	0

GROUP :	2LB0LD0		
Waveform number:	20015		
LR	SVM	KNN	RF
LB0	LB0	LC0	LC0
LB0	LC0	LB0	LB0
LF0	LF0	LC0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LF0
LF0	LC0	LC0	LC0
LF0	LC0	LB0	LC0
LF0	LC0	LD0	LD0
LF0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LB0	LC0	LB0	LC0
LB0	LB0	LB0	LB0
LD0	LD0	LB0	LD0
LF0	LD0	LD0	LD0
LB0	LB0	LB0	LB0
10	1	0	1

GROUP :	2LB0LD0		
Waveform number:	20016		
LR	SVM	KNN	RF
LB0	LC0	LB0	LB0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LD0
LF0	LC0	LC0	LC0
LF0	LF0	LB0	LD0
LF0	LC0	LC0	LC0
LB0	LC0	LB0	LB0
LB0	LC0	LB0	LB0
LD0	LD0	LD0	LD0
LF0	LC0	LC0	LC0
LD0	LD0	LD0	LD0
LB0	LC0	LB0	LB0
LF0	LD0	LD0	LD0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
11	1	0	0

GROUP :	2LB0LF0		
Waveform number:	20017		
LR	SVM	KNN	RF
LB0	LB0	LB0	LB0
LA0	LD0	LD0	LD0
LB0	LB0	LD0	LB0
LA0	LC0	LB0	LB0
LD0	LD0	LD0	LC0
LF0	LF0	LF0	LF0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LF0
LF0	LC0	LC0	LF0
LB0	LC0	LC0	LC0
LB0	LC0	LB0	LB0
LB0	LB0	LB0	LB0
LA0	LC0	LB0	LB0
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LB0	LB0	LB0	LB0
4	2	3	1

GROUP :	2LB0LF0		
Waveform number:	20018		
LR	SVM	KNN	RF
LB0	LC0	LB0	LB0
LA0	LC0	LB0	LB0
LA0	LC0	LB0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LB0	LC0	LB0	LB0
LB0	LB0	LB0	LB0
LF0	LF0	LF0	LF0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LF0
LB0	LC0	LC0	LC0
LB0	LC0	LB0	LB0
LB0	LC0	LB0	LC0
LA0	LC0	LB0	LB0
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
3	0	0	0

GROUP :	2LB0LF0		
Waveform number:	20019		
LR	SVM	KNN	RF
LA0	LC0	LC0	LC0
LD0	LC0	LB0	LC0
LF0	LC0	LC0	LC0
LB0	LC0	LB0	LC0
LA0	LC0	LB0	LB0
LA0	LF0	LF0	LF0
LF0	LC0	LC0	LC0
LB0	LB0	LB0	LC0
LA0	LC0	LB0	LB0
LA0	LC0	LB0	LC0
LB0	LC0	LC0	LC0
LB0	LC0	LB0	LB0
LB0	LC0	LB0	LB0
LA0	LC0	LB0	LB0
LB0	LC0	LC0	LB0
LB0	LC0	LC0	LC0
7	0	0	0

GROUP :	2LB0LF0		
Waveform number:	20020		
LR	SVM	KNN	RF
LA0	LC0	LB0	LB0
LB0	LC0	LB0	LB0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LF0	LF0	LF0	LF0
LB0	LC0	LC0	LC0
LB0	LB0	LB0	LB0
LB0	LB0	LB0	LB0
LA0	LC0	LB0	LB0
LB0	LC0	LB0	LB0
LB0	LC0	LB0	LB0
LB0	LB0	LB0	LB0
LB0	LC0	LB0	LB0
LA0	LC0	LB0	LB0
LB0	LC0	LC0	LB0
LA0	LC0	LD0	LB0
7	0	1	0

GROUP :	2LC0LD0		
Waveform number:	20021		
LR	SVM	KNN	RF
LD0	LB0	LD0	LB0
LB0	LC0	LB0	LB0
LD0	LD0	LD0	LD0
LB0	LC0	LB0	LC0
LB0	LC0	LB0	LB0
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LB0
LF0	LC0	LC0	LC0
LF0	LF0	LB0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LB0	LC0
LD0	LD0	LB0	LB0
LA0	LD0	LD0	LB0
LD0	LD0	LD0	LB0
5	1	0	0

GROUP :	2LC0LD0		
Waveform number:	20022		
LR	SVM	KNN	RF
LB0	LC0	LC0	LB0
LD0	LD0	LD0	LD0
LB0	LC0	LB0	LB0
LA0	LC0	LB0	LB0
LA0	LD0	LD0	LB0
LD0	LC0	LD0	LB0
LD0	LD0	LD0	LD0
LB0	LC0	LC0	LD0
LB0	LC0	LB0	LC0
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LF0	LD0	LC0	LD0
LD0	LC0	LD0	LB0
LD0	LC0	LD0	LB0
LF0	LC0	LC0	LF0
4	0	0	1

GROUP :	2LC0LD0		
Waveform number:	20023		
LR	SVM	KNN	RF
LB0	LC0	LB0	LB0
LF0	LF0	LC0	LD0
LD0	LD0	LB0	LD0
LD0	LD0	LC0	LC0
LB0	LC0	LB0	LB0
LD0	LD0	LD0	LD0
LF0	LD0	LD0	LD0
LA0	LC0	LB0	LB0
LB0	LB0	LD0	LB0
LD0	LD0	LB0	LC0
LD0	LD0	LB0	LD0
LA0	LC0	LB0	LB0
LB0	LC0	LB0	LB0
LF0	LC0	LB0	LD0
5	1	0	0

GROUP :	2LC0LD0		
Waveform number:	20024		
LR	SVM	KNN	RF
LF0	LF0	LC0	LD0
LF0	LF0	LC0	LC0
LF0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LD0	LD0	LD0	LB0
LD0	LD0	LD0	LD0
LB0	LC0	LB0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LB0	LC0	LB0	LD0
LD0	LC0	LC0	LD0
LD0	LD0	LD0	LD0
LD0	LD0	LD0	LD0
LB0	LB0	LC0	LC0
6	2	0	0

GROUP :	2LC0LF0		
Waveform number:	20026		
LR	SVM	KNN	RF
LA0	LC0	LB0	LB0
LB0	LC0	LB0	LD0
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LD0	LD0	LB0	LB0
LD0	LC0	LD0	LB0
LB0	LD0	LD0	LB0
LB0	LB0	LD0	LD0
LB0	LB0	LD0	LB0
LB0	LB0	LB0	LB0
LA0	LF0	LF0	LF0
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LB0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
4	2	4	2

GROUP :	2LC0LF0		
Waveform number:	20027		
LR	SVM	KNN	RF
LB0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LA0	LD0	LD0	LD0
LB0	LC0	LD0	LB0
LB0	LC0	LB0	LB0
LF0	LF0	LF0	LF0
LF0	LC0	LB0	LC0
LB0	LC0	LB0	LF0
LA0	LD0	LD0	LD0
LB0	LC0	LC0	LD0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LB0	LC0	LC0	LC0
LB0	LB0	LD0	LB0
LA0	LD0	LD0	LD0
LB0	LC0	LC0	LD0
LF0	LC0	LC0	LF0
3	3	5	5

GROUP :	2LC0LF0		
Waveform number:	20028		
LR	SVM	KNN	RF
LF0	LC0	LC0	LF0
LF0	LC0	LC0	LF0
LF0	LC0	LC0	LC0
LD0	LB0	LD0	LB0
LF0	LC0	LC0	LB0
LB0	LB0	LD0	LB0
LD0	LB0	LD0	LB0
LC0	LB0	LB0	LB0
LA0	LB0	LD0	LB0
LC0	LB0	LB0	LB0
LB0	LC0	LD0	LB0
LB0	LC0	LD0	LB0
LD0	LB0	LD0	LB0
LB0	LB0	LB0	LB0
LB0	LB0	LD0	LB0
4	0	8	0

GROUP :	2LC0LF0		
Waveform number:	20029		
LR	SVM	KNN	RF
LB0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LC0	LC0
LB0	LC0	LB0	LD0
LB0	LC0	LB0	LC0
LA0	LC0	LB0	LC0
LB0	LB0	LD0	LB0
LD0	LC0	LD0	LB0
LF0	LF0	LF0	LF0
LB0	LB0	LC0	LC0
LD0	LC0	LB0	LC0
LF0	LC0	LC0	LC0
LF0	LC0	LB0	LD0
LF0	LC0	LC0	LB0
LA0	LB0	LD0	LD0
LB0	LB0	LD0	LB0
4	0	4	3

REFERENCES

ABADI, Martín; AGARWAL, Ashish; BARHAM, Paul; BREVDO, Eugene; CHEN, Zhifeng; CITRO, Craig; CORRADO, Greg S.; DAVIS, Andy; DEAN, Jeffrey; DEVIN, Matthieu; GHEMAWAT, Sanjay; GOODFELLOW, Ian; HARP, Andrew; IRVING, Geoffrey; ISARD, Michael; JIA, Yangqing; JOZEFOWICZ, Rafal; KAISER, Lukasz; KUDLUR, Manjunath; LEVENBERG, Josh; MANé, Dan; MONGA, Rajat; MOORE, Sherry; MURRAY, Derek; OLAH, Chris; SCHUSTER, Mike; SHLENS, Jonathon; STEINER, Benoit; SUTSKEVER, Ilya; TALWAR, Kunal; TUCKER, Paul; VANHOUCKE, Vincent; VASUDEVAN, Vijay; VIéGAS, Fernanda; VINYALS, Oriol; WARDEN, Pete; WATTENBERG, Martin; WICKE, Martin; YU, Yuan; ZHENG, Xiaoqiang. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. 2015. Software available from tensorflow.org. Disponível em: http://tensorflow.org/.

ANCELMO, Hellen; MULINARI, Bruna Machado; POTTKER, Fabiana; LAZZARETTI, André; BAZZO, Thiago de Paula Machado; OROSKI, Elder; RENAUX, Douglas; LIMA, Carlos Erig; LINHARES, Robson. A new simulated database for classification comparison in power signature analysis. *In*: **20th International Conference on Intelligent Systems Applications to Power Systems**. *[S.l.: s.n.]*, 2019. p. 1 – 7.

ANCELMO, Hellen Cristina; GRANDO, Flavio Lori; COSTA, Clayton Hilgemberg; MULINARI, Bruna Machado; OROSKI, Elder; LAZZARETTI, Andre Eugenio; POTTKER, Fabiana; RENAUX, Douglas Paulo Bertrand. Automatic power signature analysis using prony method and machine learning-based classifiers. *In*: **2nd European Conference on Electrical Engineering and Computer Science**. *[S.l.: s.n.]*, 2018.

BOLLEN, M.H.J.; GU, I.Y.H. **Signal Processing of Power Quality Disturbances**. *[S.l.]*: John Wiley & Sons, 2006.

BOUHOURAS, Aggelos S.; GKAIDATZIS, Paschalis A.; CHATZISAVVAS, Konstantinos C.; PANAGIOTOU, Evangelos; POULAKIS, Nikolaos; CHRISTOFORIDIS, Georgios C. Load signature formulation for non-intrusive load monitoring based on current measurements. **Energies**, v. 10, n. 4, 2017. ISSN 1996-1073. Disponível em: https://www.mdpi.com/1996-1073/10/4/538.

CHANG, Hsueh-Hsien. Non-intrusive demand monitoring and load identification for energy management systems based on transient feature analyses. **Energies**, v. 5, n. 11, p. 4569–4589, 2012.

DANTAS, Pierre; SABINO, Waldir; BATALHA, Maryana. Energy disaggregation via data mining. *In*: **Proceedings of the 4th Brazilian Technology Symposium**. *[S.l.: s.n.]*, 2019. p. 541–546.

DU, Yi; DU, Liang; LU, Bin; HARLEY, R.G.; HABETLER, Thomas. A review of identification and monitoring methods for electric loads in commercial and residential buildings. *In*: . *[S.l.: s.n.]*, 2010. p. 4527 – 4533.

FAWCETT, Tom. Introduction to roc analysis. **Pattern Recognition Letters**, v. 27, p. 861–874, 06 2006.

FIGUEIREDO, Marisa; ALMEIDA, Ana; RIBEIRO, Bernardete. Home electrical signal disaggregation for non-intrusive load monitoring systems. **Neurocomputing**, v. 96, p. 66–73, 2012. ISSN 0925-2312.

HART, G. W. Nonintrusive appliance load monitoring. **Proceedings of the IEEE**, v. 80, p. 1870 – 1891, 04 1992.

HASSAN, Taha; JAVED, Fahad; ARSHAD, Naveed. An empirical investigation of V-I trajectory based load signatures for non-intrusive load monitoring. **IEEE Transactions on Smart Grid**, v. 5, n. 2, p. 870–878, 2014.

LIANG, J.; NG, S. K. K.; KENDALL G.AND CHENG, J. W. M. Load signature study –part i: Basic concept, structure and methodology. **IEEE Transactions on Power Delivery**, v. 25, n. 2, p. 870–878, 2010.

Linhares, R. R.; Lima, C. R. E.; Renaux, D. P. B.; Pottker, F.; Oroski, E.; Lazzaretti, A. E.; Mulinari, B. M.; Ancelmo, H. C.; Gamba, A.; Bernardi, L. A.; Lima, L. T. One-millisecond low-cost synchronization of wireless sensor network. *In*: **VIII Brazilian Symposium on Computing Systems Engineering**. *[S.l.: s.n.]*, 2018.

LLC, Google. **Google Colab**. 2014. Disponível em: https://colab.research.google.com/notebooks/welcome.ipynb?hl=it.

Mulinari, B. M.; de Campos, D. P.; da Costa, C. H.; Ancelmo, H. C.; Lazzaretti, A. E.; Oroski, E.; Lima, C. R. E.; Renaux, D. P. B.; Pottker, F.; Linhares, R. R. A new set of steady-state and transient features for power signature analysis based on v-i trajectory. *In*: **2019 IEEE PES Innovative Smart Grid Technologies Conference - Latin America (ISGT Latin America)**. *[S.l.: s.n.]*, 2019. p. 1–6.

MULINARI, Bruna Machado; LINHARES, Robson; CAMPOS, Daniel; COSTA, Clayton; ANCELMO, Hellen; LAZZARETTI, André; OROSKI, Elder; LIMA, Carlos Erig; RENAUX, Douglas; POTTKER, Fabiana. A new set of steady-state and transient features for power signature analysis based on v-i trajectory. *In*: **IEEE PES Innovative Smart Grid Technologies Conference - Latin America**. *[S.l.: s.n.]*, 2019. p. 1–6.

NAIT-MEZIANE, Mohamed; RAVIER, Philippe; ABED-MERAIM, Karim; LAMARQUE, Guy; BUNETEL, Jean-Charles; RAINGEAUD, Yves. Electrical transient modeling for appliance characterization. **EURASIP Journal on Advances in Signal Processing**, v. 55, p. 1–19, 12 2019.

Nait Meziane, M.; Ravier, P.; Lamarque, G.; Le Bunetel, J.; Raingeaud, Y. High accuracy event detection for non-intrusive load monitoring. *In*: **2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**. *[S.l.: s.n.]*, 2017. p. 2452–2456.

NAIT-MEZIANE, Mohamed; RAVIER, Philippe; LAMARQUE, Guy; BUNETEL, Jean-Charles Le; RAINGEAUD, Yves. High accuracy event detection for non-intrusive load monitoring. *In*: **International Conference on Acoustics, Speech, and Signal Processing**. *[S.l.: s.n.]*, 2017.

NASREEN, Shamila. A survey of feature selection and feature extraction techniques in machine learning,sai,2014. *In*: . *[S.l.: s.n.]*, 2014.

PAL, M. Random forest classifier for remote sensing classification. **International Journal of Remote Sensing**, Taylor and Francis, v. 26, n. 1, p. 217–222, 2005. Disponível em: https://doi.org/10.1080/01431160412331269698.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PICON, Thomas; MEZIANE, Mohamed Nait; RAVIER, Philippe; LAMARQUE, Guy; NOVELLO, Clarisse; BUNETEL, Jean-Charles Le; RAINGEAUD, Yves. COOLL: Controlled on/off loads library, a public dataset of high-sampled electrical signals for appliance identification. **arXiv preprint arXiv:1611.05803 [cs.OH]**, 2016.

Pöttker, F.; Lazzaretti, A. E.; Renaux, D. P. B.; Linhares, R. R.; Lima, C. R. E.; Ancelmo, H. C.; Mulinari, B. M. Non-intrusive load monitoring: A multi-agent architecture and results. *In*: **2018 2nd European Conference on Electrical Engineering and Computer Science (EECS)**. *[S.l.: s.n.]*, 2018. p. 328–334.

RASCHKA, Sebastian; MIRJALILI, Vahid. **Python Machine Learning, 3rd Ed.** 3. ed. Birmingham, UK: Packt Publishing, 2019. ISBN 978-1789955750.

RAYBAUT, Pierre. Spyder-documentation. **Available online at: pythonhosted. org**, 2009.

Renaux, D.; Linhares, R.; Pottker, F.; Lazzaretti, A.; Lima, C.; Coelho Neto, A.; Campaner, M. Designing a novel dataset for non-intrusive load monitoring. *In*: **2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC)**. *[S.l.: s.n.]*, 2018. p. 243–249.

RENAUX, Douglas P. B.; LIMA, Carlos R. Erig; POTTKER, Fabiana; OROSKI, Elder; LAZZARETTI, Andre E.; LINHARES, Robson R.; ALMEIDA, Andressa R.; COELHO, Adil O.; HERCULES, Mateus C. Non-Intrusive Load Monitoring: an Architecture and its evaluation for Power Electronics loads. *In*: **IEEE International Power Electronics and Application Conference and Exposition (PEAC)**. *[s.n.]*, 2018. p. 1–6. Disponível em: https://ieeexplore.ieee.org/document/8590472/.

RENAUX, Douglas Paulo Bertrand; LINHARES, Robson Ribeiro; POTTKER, Fabiana; LAZZARETTI, Andre Eugenio; LIMA, Carlos Eduardo Erig de; COELHO-NETO, Adil; CAMPANER, Mateus Hercules. Designing a novel dataset for non-intrusive load monitoring. *In*: **VIII Brazilian Symposium on Computing Systems Engineering**. *[S.l.: s.n.]*, 2018.

RENAUX, Douglas Paulo Bertrand; LINHARES, Robson Ribeiro; POTTKER, Fabiana; LAZZARETTI, Andre Eugenio; LIMA, Carlos Eduardo Erig de; COELHO-NETO, Adil; CAMPANER, Mateus Hercules. Designing a novel dataset for non-intrusive load monitoring. *In*: **VIII Brazilian Symposium on Computing Systems Engineering**. *[S.l.: s.n.]*, 2018.

RUANO, Antonio; HERNANDEZ, Alvaro; UREñA, Jesús; RUANO, Maria; GARCíA, Juan. Nilm techniques for intelligent home energy management and ambient assisted living: A review. **Energies**, v. 12, p. 2203, 06 2019.

Scholkopf, B.; Kah-Kay Sung; Burges, C. J. C.; Girosi, F.; Niyogi, P.; Poggio, T.; Vapnik, V. Comparing support vector machines with gaussian kernels to radial basis function classifiers. **IEEE Transactions on Signal Processing**, v. 45, n. 11, p. 2758–2765, 1997.

UKIL, Abhisek; ŽIVANOVIć, Rastko. Adjusted haar wavelet for application in the power systems disturbance analysis. **Digital Signal Processing**, v. 18, n. 2, p. 103 – 115, 2008.

WANG, A.; CHEN, B.; WANG, C.; HUA, D.D. Non-intrusive load monitoring algorithm based on features of v–i trajectory. **Electric Power Systems Research**, v. 157, p. 134–144, 04 2018.

WANG, A. Longjun; CHEN, B. Xiaomin; WANG, C. Gang; HUA, D. D. Non-intrusive load monitoring algorithm based on features of V–I trajectory. **Electric Power Systems Research**, v. 157, p. 134–144, 2018.

XU, Yun; GOODACRE, Royston. On splitting training and validation set: A comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning. **Journal of Analysis and Testing**, v. 2, 10 2018.

ZOHA, Ahmed; GLUHAK, Alexander; IMRAN, Muhammad; RAJASEGARAR, Sutharshan. Non-intrusive load monitoring approaches for disaggregated energy sensing: A survey. **Sensors (Basel, Switzerland)**, v. 12, p. 16838–16866, 12 2012.