EXPLOITING COMPRESSED SENSING IN DISTRIBUTED MACHINE LEARNING

By

STAMATAKIS M. EMMANUEL

M.Sc in Advanced Manufacturing Systems, Automation and Robotics, HMU,2020

B.Sc in Informatics University of Piraeus, 1998

A THESIS

Submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SCHOOL OF ENGINEERING

HELLENIC MEDITERRANEAN UNIVERSITY

2023

Approved by:

Associate Professor

Panagiotakis Spyridon

This page has intentionally been left blank.

Abstract

A range of contemporary applications, such as remote monitoring of crucial measurements, mechanical fault recognition, remote detection of structural strain in constructions, and many others, have become possible today thanks to the Internet of Things (IoT). Due to the fog-based design of these systems, devices are placed at the extreme edge, requiring data transmission to a central node. At the same time, the performance of these devices is limited both by power requirements for wireless transmission via the network and by their limited computational capabilities and features.

Ideally, there would be a way to compress the data to a great extent so that, upon receipt by the receiving node, it could be accurately reconstructed. This need is addressed by a new technique called compressive sensing. Its operation is based on the sparsity characteristic of most natural signals when represented in a specific basis. This technique allows reconstruction with far fewer points than traditional sampling techniques, such as Nyquist.

The intersection of the compressive sensing (CS) and machine learning (ML) domains has garnered significant research interest, combining the fundamental principles from both areas, and is the focus of the present master's thesis. Through this study, the use of CS as a compression tool before transmission is explored, and the limits of its application in distributed ML systems are determined. More specifically, the impact of CS on the recognition capability of reconstructed signals by a trained ML model is examined. The study aims to achieve a high recognition rate for reconstructed signals at the network's edge and extreme edge.

This page has intentionally been left blank.

Table of Contents

List of Figures

Acknowledgments

With the completion of my diploma thesis as well as the end of my postgraduate studies, I would like to thank the supervising professor Mr. Panagiotakis Spyridon of the Department of Electrical and Computer Engineering of EL.ME.PA, for his undivided help and guidance in my studies career as well as for the trust he showed me for this particular job. All my professors for their effort and the knowledge they gave us. I would especially like to thank my professor Dr. Papadakis Stylianos for his continuous support and valuable help in the machine learning part during the development of this paper. Finally, I must not forget to thank my family, as well as everyone who supported me spiritually, helping me to complete the work assigned to me.

This page has intentionally been left blank.

Dedication

dedicated to

To all who strive to the end.

This page has intentionally been left blank.

Preface

For many contemporary applications and particularly those that use micro-controllers, the capacity to gather and analyze signals effectively is crucial. Two significant and quickly developing fields of research, compressive sensing and distributed machine learning, have the potential to completely change how we gather and use data in these systems. By dividing the work across several computers, distributed machine learning enables the training of large and complicated machine learning models, while compressive sensing provides the efficient collection of data that are sparse in particular domains.

The intersection of these two fields is the area of our interest in this master thesis while we focus on the use of micro-controllers at the edge and extreme edge for signal sampling, compressed sensing and machine learning classification. We present a method developing a trained model for classifying signals from an IMU that represents human gestures. This method takes advantage of the properties of sparse signals and the capabilities of distributed systems to enable more efficient and effective machine learning, particularly in the context of micro controller based systems.

The research presented in this thesis is motivated by the need to address the challenges of dealing with large and complex data sets in modern machine learning applications that involve micro-controllers. By leveraging the power of compressive sensing and distributed machine learning, we aim to provide new solutions that can enable more effective and efficient data processing and machine learning in such systems.

This work has the potential to have a significant impact in a wide range of areas that rely on micro-controllers placed on the edge and even on extreme edge, including but not limited to Internet of Things (IoT) systems, and embedded systems. We hope that it will contribute to a better understanding of the capabilities and limitations of compressive sensing and distributed machine learning in the context of micro-controller-based systems, and inspire further research in these areas

This page has intentionally been left blank.

Synopsis

We live in an era where the spread of IoT is an indisputable fact. A variety of sensors monitors and records data and sends them to central processing and storage facilities. Sensors are part of a wide variety of systems with a wide range of applications, from medical imaging to audio and visual monitoring.

In a typical IoT scenario, many self-powered constrained devices capture real-world data and communicate with each other and with the cloud, normally, over low-power wireless connections to exchange information and provide specific services. However, the power consumption associated with wireless transmission, the scarce network resources, and the constrained capabilities of such IoT devices, limit their performance [1]. Thus, a compression technique that would drastically reduce the size of the data to be sent, while ensuring the reconstruction of the original data in the destination node, is highly welcome. In this context, compressed sensing (CS) technique is a promising standard to be integrated into IoT systems design. CS is a new compression theory and signal recovery philosophy that exploits the sparsity behavior of most physical signals when represented in a suitable domain. When a signal is sparse in a specific domain then, CS samples signals much more efficiently than the conventional Shannon-Nyquist sampling method. It is known, it turns out, that many physical signals, such as the sound, image or measurements of an IMU, are sparse in a Fourier domain or in a wavelet domain [2]. CS allows the sampling of sparse signals, with a very small number of samples so that reconstruction can be achieved at the receiver.

The main disadvantage of conventional sampling methods followed by compression is that the sampling steps deal with a huge amount of data that requires efficient sensors and large storage capacities. This as a result, leads to increased processing costs, and therefore to waste of time. CS solves this problem. Instead of compressing data after sampling, CS performs data sampling in a compressed manner [2]. CS is a new example of signal sampling, which allows signal retrieval using only a few samples much less than one would even expect from Nyquist. On the other hand, machine learning is one of the most modern and powerful trends, which can effectively complement CS [3]. With its help we can implement decision-support systems based on CS-data we have collected, related to decision making.

Today there has been a lot of interest in intersection of compressed sensing and machine learning meet. The goal of this area, called "Compressive Learning," is to combine the two disciplines in order to create cutting-edge methods for data analysis and signal processing [4]. Reconstruction of compressed signals, using neural networks can be considered a subject within the area of compressive learning. It is important to continue researching the fast developing topic of compressed learning because it has the potential to significantly improve the state of the art in signal processing and judgment.

The purpose of this master thesis is on the one hand to study the use of CS technique as a compression tool and on the other to investigate the limits of its application in distributed compressed learning scenarios. To this end, in our system, an IoT node (leaf node) performing CS will sample a signal at rates much lower than those specified by Nyquist. The compressed signal will then be sent for reconstruction to a receiver node (sink node). At the same time, a machine learning classification model located in both the leaf and the sink node will be trained to recognize specific patterns using the compressed signal. Our goal is to achieve the closest possible prediction rate between recognitions at the edge and the extreme edge of our network, under various CS conditions.

Study points of the present master thesis will be the differences in the performance of the ML model if we represent the original signal in different domains (wavelet, DCT) and with different sampling rates. Another point of interest will be tuning critical values (hyper parameters) for critical CS points. More specifically, we will try to answer the following research questions: a) what is the most appropriate compressed sampling table (Bernoulli, Random, Gauss etc)? b) What is the minimum number of compressed coefficients that allow successful reconstruction and recognition at the edge? Finally, a comparison will take place between two methods of feature selection as far as the correlation between them and the correlation of the features with the output of the machine learning model.

1 Introduction

1.1 General

Modern signal processing techniques like compressed sensing make it possible to acquire large amounts of data with fewer measurements than previously necessary. This is accomplished by exploiting the data's built-in structure, such as sparsity or low-rankness, to lower the required number of measurements. Numerous industries of different fields use compressed sensing, including wireless communication, medical imaging, and image and video compression [1].

In recent years, there has been a growing interest in applying compressed sensing to distributed machine learning. This application involves training machine learning models on distributed systems, such as edge devices or Internet of Things (IoT) devices [5]. This approach also involves capturing data on extreme edge devices (leaf nodes) such as micro-controllers (esp, Arduino etc) and send them for classification to a trained sink node in the network. Distributed machine learning has the advantage of being able to process data closer to the source, reducing the need for data transmission and storage. This is especially important at the edge and extreme edge, where resources are often limited and latency is a concern [6] However, there are DML topologies where, in some IoT nodes, it is not possible or desirable to implement ML on. This means that the data will have to be sent for identification to a central node. This will be desirable either for central management, or design, or for any other reason. In this case it is desirable to send as little data as possible for reasons of speed, security and many others that will be analyzed later. At this point, compressed sensing comes to cover this need.

By applying CS, it became possible to reduce the amount of data that needs to be transmitted and processed at the edge, improving the efficiency and performance of the system [7]. Especially at leaf nodes, in distributed machine learning (DML) systems, it can provide significant benefits in terms also at a point of privacy protection. By reducing the amount of data that needs to be transmitted to the sink node, CS can help to minimize the risk of sensitive personal information being exposed to external parties. This is especially important in scenarios where data is collected from a large number of devices, as the amount of data that needs to be transmitted and stored (in a central sink node) can be significantly reduced and also privacy concerns are a concern as we will see next.

In addition, using CS at the leaf nodes allows for sensitive data to be processed and analyzed locally, rather than being transmitted to a centralized server. The CS processed data are then transmitted over the network to known sink nodes that will perform a ML task, as the designed DML model defines. What happens if someone steals our data is a major concern. There is where CS comes to increase security [8]. In order to reconstruct the initial signal from the compressed data, the receiver must have the representation matrix and other parameters that are known only to the leaf node and to authorized reconstructors. Unauthorized parties would not have access to the necessary information to reconstruct the data.

Another potential application of compressed sensing in distributed machine learning is in the field of wireless communication. CS is used to reduce the amount of data transmitted over a wireless network improving the efficiency, the power consumption and finally the performance of the system [10]. There are several challenges that need to be addressed in order to fully realize the potential of compressed sensing in distributed machine learning. One challenge is the development of efficient algorithms for compressed sensing in distributed systems. Another challenge is the design of systems that can effectively integrate compressed sensing into distributed machine learning scenarios [11].

1.2 The basic idea

We know that distributed machine learning (DML) is a way to process and analyze data from distributed systems, such as edge and extreme edge devices of Internet of Things (IoT) devices. One of the main challenges of distributed machine learning (DML) is the need to transmit and store large amounts of data, which can be resource-intensive and may have negative impacts on privacy. In order to address this challenge, compressed sensing (CS) has been proposed as a way to acquire data with a minimal number of measurements. We know about the impact about the privacy and security but what about the potential loss of information that may occur during the compression process? For example, how does the use of compressed sensing affect the accuracy of machine learning trained models? How does it compare to traditional approaches that do not use compression?

It is important to consider the tradeoffs associated with the use of CS, as the information may be lost during the compression process. This may have an impact on the accuracy of machine learning models trained on the data. In order to assess the benefits and tradeoffs of using CS in DML, it is important to examine the impact of CS on model accuracy and compare it to traditional approaches that do not use compression [12].

1.3 Contribution

In this context, the objective of this thesis is to explore the use of CS in DML and assess the benefits and trade-offs of this approach. To achieve this goal, we will conduct experiments to evaluate the impact of CS on model accuracy in signals captured of an IMU embedded in a micro-controller (Arduino nano33 ble sense).

First we will create a dataset from the imu, forming 3 different moves (a triangle, a circle and the letter M). Next feature selection will be performed on that dataset with different approaches, to limit the numbers of features. Using traditional machine learning a classifier will be trained on that data. As we can easily understand the model hasn't seen a reconstructed from compressed sensing signal. The trained model should be installed both at sink and leaf node (edge and extreme edge).

In the context of this thesis idea, the scenario is as follows: one or more extreme edge devices, such as Arduino Nano 33 BLE Sense micro-controllers, capture IMU signals while performing various movements. These signals will be compressed sensing and transmitted to a sink node, such as a Raspberry Pi, where they are reconstructed and classified using our trained classifier. The goal of this research is to explore the trade-offs between the benefits of CS in this scenario, as far as the potential loss of information that may occur during the compression process is concerned.

To answer this research question, a number of experiments need to be conducted. Some questions that need to be addressed through these experiments include:

- How does the use of CS at the leaf nodes affect the accuracy of the machine learning classifier at the sink node?
- How does the use of CS at the leaf nodes compare to traditional approaches that do not use compression in terms of model accuracy and resource efficiency?
- What is the optimal compression ratio for the specific application and requirements of the DML system that ensures that the reconstructed signal will be classified correctly?

This study provides an integrated system that can work in a DML. More specifically, a method of creating a machine learning model will be presented which will work effectively in leaf and sink nodes of a DML system. This model will take into account all trade-offs when using compressed sensing in a DML environment. In this environment a leaf node at the extreme edge will receive IMU signals which will be sent to a sink node where they will be recognized.

It is wanted this research to contribute to a better understanding of the capabilities and limitations of compressive sensing and distributed machine learning in the context of micro-controller-based systems, and inspire further research in these areas

1.4 Structure

In this section, the structure of the report of this thesis is analyzed, by presenting the chapters that follow. The construction was done in a systematic and comprehensive manner in order to present in a clear and comprehensible manner the purpose of the research, the problems we faced as well as the findings that emerged after the exhaustive study of the object.

In the Background and related work chapter, there are two large sections, background and related work. In the background section, there is a systematic presentation of the basic principles of the areas that we are interested in this work. Initially a clarification is made of the space of IoT and extreme edge in which we work, presenting its characteristics and particularities. Then a reference is made to the part of machine learning that interests us, presenting basic principles and characteristics of our concern and especially SVM. Special reference is also made to the explanation of the operation mode of the basic ML methods that will be used in terms of feature selection, evaluation mechanism etc.

In this section, extensive reference is made to compressed sensing method in terms of its mathematical background at the level we are interested in, as well as its basic operating principle. Comparisons are made between different representation bases, and points of special interest are clarified such as the role of sample arrays, representation and transformation basis as well as in which cases it is applicable and in which it is not.

In the related work section of the chapter under consideration, a study is conducted in the area of compressed sensing, where works related to our own study are presented, which make use of the CS technique in combination with ML. In addition to presenting the works, a relative comparison is made regarding the approach and the way the technique is used in relation to our own study.

The chapter "Design of the system" follows, in which a presentation of the hardware and software of the system we developed is made in order to carry out the experiments of the present study. Reference is made to the design and topology of the system we used, and the role of CS is presented through the system's design.

In the chapter "Implementation of the System," we present the way we designed and developed the software for the study, which combined multiple areas. The design of the end-to-end ML system is analyzed, as well as the method developed to port the ML model and make it executable on a microcontroller. The development of the CS system is also analyzed, how it was implemented in Python, our choices in its basic parameters, and the method we followed.

In the next chapter, "Evaluation," the main chapter of the present study, the implementation of both the ML and CS systems is carried out. We present the results of our choices in the basic parameters presented during the development of the systems and how our choices, through the design of appropriate experiments, led to the conclusions that are also the answers to the questions faced by the present thesis. Different ML models were created, and we study their behaviors; we also extensively and thoroughly examine the solver that our study has indicated as the most suitable. An extensive study is also conducted on the role played by the sampling matrix, and an algorithm is presented, with the help of which we answer our main questions.

In the last chapter of the present work, the conclusions drawn during the study are presented, and by summarizing them, a comprehensive picture of the relationship between CS and distributed ML is provided. The chapter concludes with questions that arise after the completion of the study, which serve as suggestions for future research topics. The area we are studying is new and represents a very dynamic field with strong interest due to its extensive application.

2 Related technologies and Related work

2.1 IoT

The technological term "Internet of Things" is becoming more and more well-known. If we were try to give a simplified interpretation of the term, we would say that it is a technology that turns simple devices into smart and interconnected ones.

The Internet of Things (IoT) includes a software and hardware infrastructure that connects the physical world to the Internet. Due to the explosive growth of user interest, the number of IoT devices has increased dramatically in recent years. It is estimated that by 2025, more than 75 billion devices will be connected to the Internet [13], with what this implies for the economic impact on the global market. IoT devices usually have limited computing power, small memories but at the same time generate large amounts of data.

2.1.1 IoT on the Edge

In our everyday life, low-power systems to which sensors are mainly adapted are used in homes, vehicles and workplaces. The huge reported growth of the IoT sector has been achieved mainly thanks to the so-called "cloud" (cloud computing) which, however, has several disadvantages, the most important of which is the delay it brings to the network

The delay is caused by the transfer of data from the IoT data collection devices (IoT edge devices) to the central point where they are processed (cloud) and of course the realization of all the required actions [14]. It is obvious that this disadvantage is unacceptable in very important services such as e.g. monitoring the user's health.

With the increasing use of IoT devices, the "only cloud computation model" is becoming impractical. This is because it increases latency, reduces data network bandwidth, and raises privacy and reliability issues at the same time. The solution will be to bring the computing process while making the relevant decisions as close as possible to the point of data collection (edge computing).

The ability for a device to make a decision based on the data it collects is possible using machine learning. A technology that makes it possible to create a new term, the Internet of

Conscious Things. Unfortunately, however, limitations in computing capabilities and the lack of resources in these devices either limit or prohibit the application of complex machine learning (ML) algorithms.

The need to be able to delegate data analysis and decision-making tasks to edge devices is so great that the topic "Edge Computing" has emerged as one of the most prevalent in searches and keywords of scientific articles and papers (and not only). This is illustrated in Figure 1.

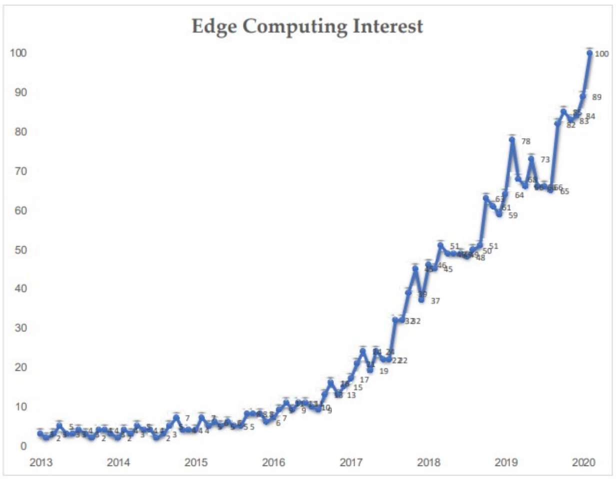


Figure 1: Edge Computing Interest (source : Google Trends)

2.1.2 Machine Learning on the Edge

In recent years, we have, wittingly or unwittingly, become dependent on machine learning and deep learning technology. These technologies include everyday applications such as photo classification, face and speech recognition, but also in more important cases, such as self-driving cars and medical diagnosis. For the most part these models are very large and require a lot of computing power, which makes them very difficult to run on any of the billions of microcontrollers in operation today.

ML developers focus on redesigning existing successful models with a reduced or even modified number of parameters in order to create the lightest model. In this way, the demand for memory and computing power is reduced while at the same time maintaining its accuracy in predictions. So the compression of the model is what allows us to integrate it even into very small micro-controllers, like for example the Arduino Uno, which has 2K RAM, and which can, as it turns out, perform ML functions [15].

The trend is so great that major ML libraries such as TensorFlow released a micro-controller-specific version, much lighter (TensorFlow Lite) but incorporating all its core features. This has been a springboard for tiny devices that can be used for IoT needs to incorporate machine learning.

DML is a recently raised term, actually is a subset of machine learning that trains and run models at different and multiple devices (distributed). This approach is used when the amount of data is too large to process or when the computation that is required for the training of a model is too intensive for a single machine to handle. DML is used in a variety of applications such as natural language processing, computer vision, data analysis, gesture classifications etc. It has also become increasingly important for the development of artificial intelligence and machine learning systems that can operate in real-time environments [16].

Extreme edge computing and distributed machine learning are closely linked concepts. Instead of using a centralized server for processing, edge computing uses devices that are close to the data source. This method can help a system respond more quickly and with less latency. This idea is expanded upon by extreme edge computing, which uses embedded technology like smart devices or sensors to carry out processing. Edge and extreme edge computing is used in the context of machine learning to carry out computation on the devices that create the data. Applications that need real-time decision-making or have restricted connection can benefit from this method.

2.2 Machine Learning

At a higher level, a machine learning problem can be divided into three types of tasks (Figure 2):

- Data tasks (data collection, data cleaning and feature configuration).
- Training tasks (building machine learning models using data features).
- Assessment tasks (evaluation of the model).

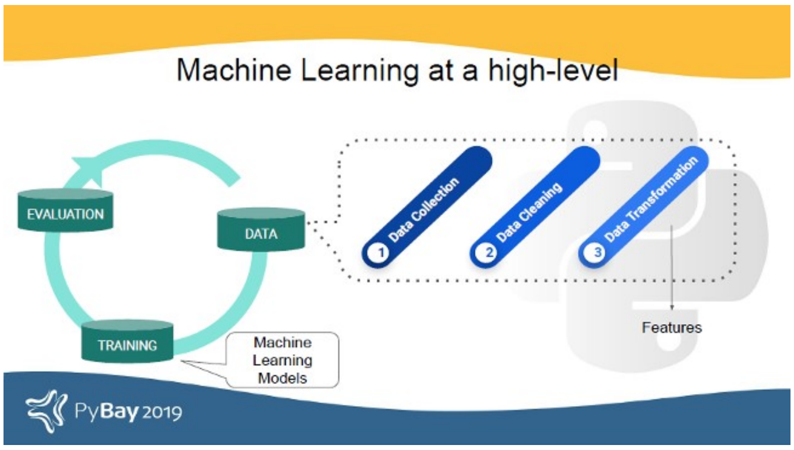


Figure 2: Machine learning at high level, source: *PyBay2019*

Different types of data use different processing techniques. For example, an image looks like one thing to the human eye, but a machine sees it differently after it is converted to numerical features derived from the pixel values of the image using different filters (depending on the application). So there are processing techniques for the IMU sensor data type, which is essentially pure time series data.

If we wanted to look at a more abstract level what machine learning is we would say that it is a subset of artificial intelligence (AI) that can solve tasks that are infeasible or too difficult to handle with more traditional programming languages. In 1959, Arthur Samuel defined machine learning as "The field of study which gives computers the ability to learn, without being explicitly programmed" [17]. Despite the variety and effectiveness of machine learning algorithms, it is becoming clear that data is undeniably more important than any chosen algorithm.

There are several categories of machine learning, which can be broadly classified (based on the feedback of the system under development), into three main types: supervised, unsupervised and reinforcement learning. Based on the result produced by the model, machine learning algorithms are divided into 3 categories: Classification, Regression and Clustering Algorithms.

In this thesis we are going to implement among others, a system that recognizes human gestures. For a system like that, a type of "supervised learning" would be appropriate. In supervised learning, we train a model on labeled training data, which consists of input data and the corresponding correct output labels. The goal is to learn a function that can map the input data to the correct output labels.

Such a system would also be able to perform a classification task. In classification, the goal is to predict the class or category that a given input belongs to. In this case, the input would be the data from the IMU sensor, and the classes would be the different gestures or movements that the system is able to recognize.

After a thorough study, as will be presented in a documented and detailed manner, SVM was chosen as the classicization algorithm

Support vector machines (SVMs) are a type of supervised machine learning algorithm that can be used for classification and regression tasks. SVMs are not directly based on a distance metric, but they do use a distance-based approach to find the hyper plane in a high-dimensional feature space that maximally separates the classes in the training data [18].

In the case of a linear SVM, this hyper plane is simply a line that separates the classes. In the case of a non-linear SVM, like ours, the classes are separated by a "margin" around the hyper plane, which is achieved by using a kernel function to map the data into a higher-dimensional space. The distance between the hyper plane and the closest points in the training data (called the "support vectors") determines the width of the margin.

SVMs are known for their good generalization performance and ability to handle high-dimensional data. They can also be effective in cases where the number of features is much greater than the number of samples. However, they can be more computationally intensive to train than some other algorithms, and they may not be the best choice for very large datasets.

2.2.1 SVM algorithm

Support Vector Machine (SVM) is a supervised machine learning algorithm that can be used for either classification or regression purposes. It is most often used in classification problems, as in the case of this thesis.

The basic idea of SVM is to find a hyper plane that best divides a data set into two classes (binary classification) [19].

As we can see in Figure 3, at its left part, a simple classification problem appears with three different possible separation lines. All three of the lines drawn separate the two categories, with the feeling that all 3 are 'correct'. However, if we had to choose one of the lines as a classifier for a test data set, we would have to ask ourselves by what criteria to choose it and also which of the lines is better than the others.

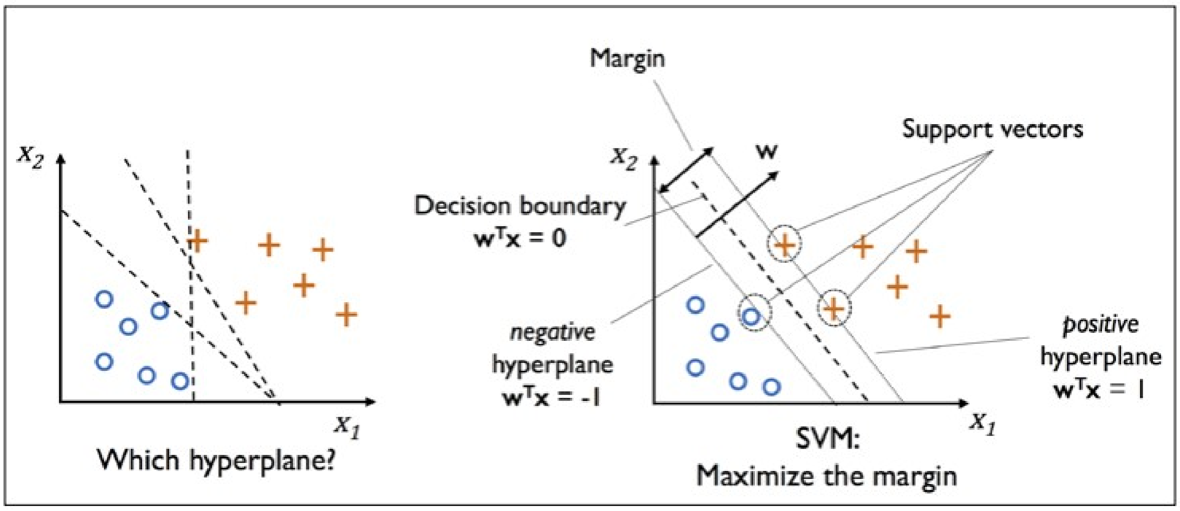


Figure 3: SVM basic concept

If we select the lines that are really close in either the left or the right subset of the data then there is a chance that an input point will be classified in the wrong class. This is because we have fitted the line tightly over some of the data points we have in the training set. Thus, a point that enters will be very close, for example, to the left category, but the line places it to the right of it, so it belongs to the right category.

The line in the middle does not have this problem. So we need to select the line in the middle as the separator. This is the basic idea in SVMs. At this point we should define some basic concepts.

- The line in the middle of the separation is called the decision boundary

- The area defined by the decision line, if we travel to the left and right of it until we encounter data, is called the margin. The margin is symmetrical around the decision line.
- The data that are closest and actually define the dividing lines of the margin are called support vectors.

What we want is to have the maximum possible margin [20], so that our algorithm can better generalize to the data that is going to come, as a result it will be a better classifier.

An important aspect of SVMs is that support vectors are the most useful data points. After the training we can throw away all the data except the support vectors, which will be used to create the model that will perform the categorization. This is a very important feature useful for saving on data storage, which is valuable in the very limited storage capabilities of the small embedded systems we are studying.

Continuing, the basic concept and operation of SVM will be presented and also how the maximum possible Margin can be calculated from the data.

First of all, the line that separates the 2 classes (decision boundary) in its most general form is: y=w•x+b where w is the coefficient of the line and b is the bias from the origin of the axes. So if we have an e.g. 2-dimensional space, then the above equation will be of the form y=w1•x1+w2•x2+w0. Essentially it is the inner product of the two vectors w and x. Therefore, it can also be written as y=wT•x+w0. This line defined from that equation, divides the plane into two half-planes and therefore can be a linear classifier. A special feature of this separator is that we are not interested in the value of the equation but the sign, y=sign(w0+(w,x)). That is, if a new element comes, the sign of the equation is of our interest. If it is positive it belongs to one class if it is negative it belongs to the other class [21].

Straight lines, however, that will separate the two classes will possibly be found plenty, as presented previously. We wish to find the line that leaves the maximum possible margin between the classes and this line is exactly the linear separator of SVMs.

As is known from geometry, the distance of a point x0 from a line w0+wT•x=0 is:

$$d(p,H)=\frac{|w_0+(w,x_0)|}{\|w\|}$$

Equation (1) distance of a point from a straight line

In eq(1), $\| w \|$ is the Lp norm of the vector w (lets say of dimension m) whose value is calculated from eq(2)for p=1 (Manhattan norm). When p=2 I have the Euclidean norm:

$$L_p = \| w \|_p = (\sum_{i=1}^{m} | x |^p)^{\frac{1}{p}}, p \geq 1$$

Equation (2) Lp Norm calculation

The SVM algorithm considers it to have 2 classes 1 and -1. So the equations of the lines to the left and right of the decision boundary will be as shown in Figure 3. Given eq(1), it is found that the distance of the class 1 line, from the decision line is

$$d(x_s, H) = \frac{1}{\| w \|}$$

the same as the distance of the class -1 boundary line. So the distance between these two lines defined by the margin is

$$m \arg in = \frac{2}{\| w \|}$$

It is therefore obvious that the margin is maximized when w is minimized [21].

So summarizing all of the above, we can say that the function of SVM focuses on finding the minimum w (which maximizes the margin). In addition to this, the decision function should give value ≥1 if it belongs to one class and ≤-1 if it belongs to the other. In general, we can write the above in the form of equations as follows:

- Minimization of $\frac{1}{2} \bullet \| w \|_2^2$
- $y_i \bullet (w_0+(w,x))-1 \geq 0,\ y_i \in \{-1,+1\}$

We have one constraint for each data, so in N data we have N constraints [21]

Given that:

$$\| w \|_p = \left(\sum_{k=1}^{m} | w_i |^p\right)^{\frac{1}{p}}$$

If we have p=1: L1 SVM Linear Programming, If we have p=2: L2 SVM Quadratic Programming

$$\| w \|_2^2 = (w, w)$$

2.2.2 Evaluation of Classification models.

The process of evaluating the machine learning model we create is one of the most important phases. Using evaluation we want to see how the model will behave, i.e. how well it will classify an input data into the correct class. The evaluation is done in two ways. By using the evaluation indicators and by using the evaluation mechanisms presented in the next section.

2.2.2.1 Confusion matrix

Diagonal values of the confusion matrix are correct predictions, while off-diagonal values are incorrect predictions.

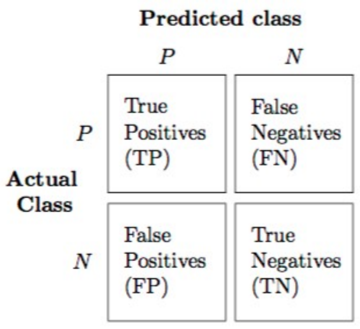


Figure 4: Confusion Matrix

2.2.2.2 Accuracy

This measure enables us to estimate the accuracy with which an input data will be classified into the correct class [22]. As model accuracy we define the amount of sample data correctly categorized to the total number of sample items.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Equation (3) Accuracy calculation formula

Only the accuracy as a measurement does not accurately reflect the true situation when using a data set that does not contain an equal number of elements from each class which must be taken seriously. Also, other indicators such as precision and recall are equally important.

2.2.2.3 Precision.

Precision tries to answer the following question:

What percentages of positive identifications were actually correct? And mathematically it is defined based on the following formula:

$$\Pr ecision = \frac{TP}{TP + FP}$$

Equation (4) precision formula

From eq(4) it can be seen that: A model that produces no false positives has an accuracy of 1.0.

2.2.2.4 Recall.

Recall attempts to answer the question, what percentage of true positives were correctly identified as positive? Mathematically it is defined by eq(5):

$$\mathrm{Re}\, call = \frac{TP}{TP + F\mathrm{N}}$$

Equation (5) Recall formula

It can be seen that a model that produces no false negatives has a recall value of 1.0

2.2.2.5 F1 Score

To fully assess the effectiveness of a model we have just built, both precision and recall must be considered, which are unfortunately at odds. More specifically, improving precision usually reduces recall and vice versa.

The F1-score combines both precision and recall in order to calculate a more harmonious mean between these indices. The formula that calculates the F1-score is given by the following equation:

$$F1 - score = 2 * \frac{\Pr ecicion\ * \mathrm{Re}\, call}{\Pr ecsion\ + \mathrm{Re}\, call}$$

Equation (6) F1 score formula

2.2.3 Evaluation mechanisms

The evaluation metrics show the behavior of the model, when it is applied to a set of data of which we know the class to which each one belongs (known output to known input). However, if we want to see what will be the behavior of the trained model in data it has not encountered, then these metrics cannot help us. At this point it is appropriate to mention two basic terms, those of under-fitting and over-fitting[21].

Over-fitting occurs when a model is applied and perfectly fits the data in the training set. This implies that there is excellent index performance (accuracy) but very poor on new input data. Essentially such a model is useless.

Under-fitting occurs when essentially a model cannot perceive the internal information of the data set, that is, when the model does not apply well to the data. This is usually a result of simplistically designed models, improper data pre-processing resulting in outliers and other reasons.

How do we deal with these two problems? With the process of Cross Validation [21]. There are various evaluation mechanisms with the most important being Hold-out validation and Cross-Validation.

2.2.3.1 Hold-Out Validation

This mechanism is extremely simple. We randomly split our data into two different size chunks one large and one small. We train the model on the bulk of the data and evaluate the validation metrics on the smaller set. Computationally speaking, the process is simple to program and characterized by fast execution. The downside is that it is not statistically robust.

The validation results are from a small subset of the data, hence its estimate of the generalization error is less reliable.

2.2.3.2 K-Fold Cross-Validation

Cross-Validation is a technique for evaluating machine learning models on a limited sample of data figure 5. The procedure has a single parameter called k which refers to the number of groups into which a given sample of data should be divided. Hence, the process is often called k-fold cross-validation. When a specific value for k is chosen (let k= 10) it becomes 10-fold valid.

Cross-validation is mainly used in applied machine learning to estimate the behavior of a machine learning model on data it has not yet seen. It is a popular method because it is simple to understand and because it generally leads to a less biased or less optimistic estimate of model skill than other methods, such as a simple hold out validation. In general, the process is as follows [21]:

1. The dataset is shuffled.

2. The set of data is divided into k groups

3. For each of the k groups:

    3.1. The group becomes a test set

    3.2. The other k-1 groups become training sets (train set).

    3.3. I train the model on the train set

    3.4. Evaluate the trained model on the test set

    3.5. I get and store the evaluation score

3.6. I repeat process 3 until all groups have finally been a test set

4. I find the average of K evaluations and this is the evaluation index of the model (cross validation score).

Important note: each data in the dataset is assigned to a single group and remains in that group throughout the process. This means that each sample has a chance to be used in the test set 1 time and used to train the model k-1 times.

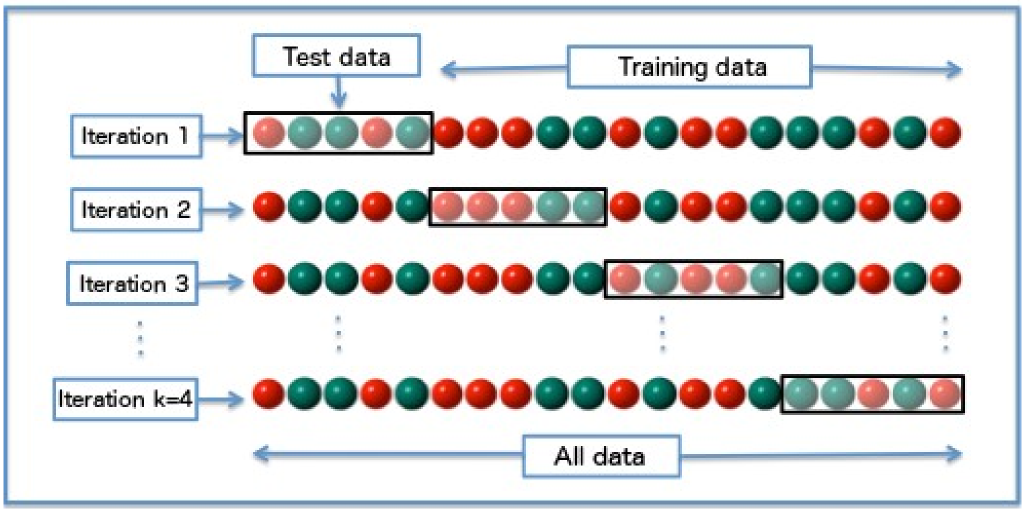


Figure 5: K-Fold cross validation, source: Wikipedia

Selecting the best k is a very important matter. An incorrectly chosen value can lead to either a large variation in the score or a bias of the model. Studies have shown that k=10 is a generally good value. The cross-validation technique would be the strongest validation technique but it has some weaknesses. More specifically, it has been found that it can cause a so-called shift of the data set. I have data shift when equal distribution of each class in the training and test sets is not ensured. To solve this problem, a variant of the k-fold cross validation method was developed which was called Stratified k-fold Cross Validation [21] and which essentially ensures the equal distribution of the classes as shown in the Figure 6.

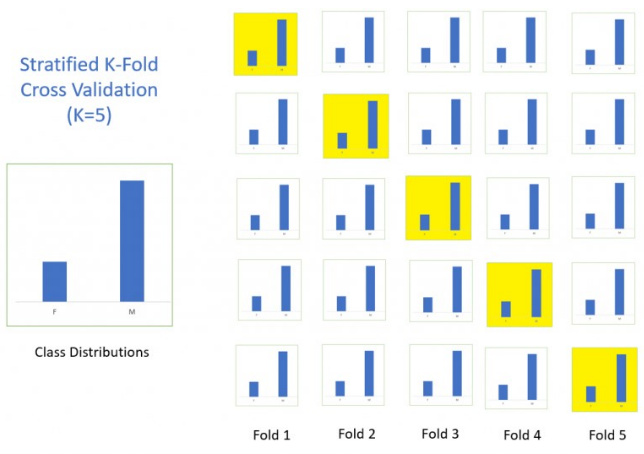


Figure 6: Stratified k-Fold Cross Validation

2.2.4 Feature Selection

The data we use to train a model has a huge impact on its effectiveness; in fact our model is our data. Regardless of the system output, even partially related data to output, can negatively affect the performance of the model. Therefore feature selection and data cleaning should be the first and most important step of its design.

The feature selection technique aims to reduce the features in the data set in order to reduce the complexity of the resulting model. Of course, with the reduction, the predictive accuracy of the model is not degraded at all; it is degraded to a tolerable level [23]. It is one of the initial stages in the machine learning process. The main reasons for using feature selection are:

- Allows the machine learning algorithm to train faster.
- It reduces the complexity of a model and makes it easier to convert it into executable code
- Improves the accuracy of a model if the right subset of data set is selected.
- Reduces over-fitting.
- Reduces training time: fewer data points reduce algorithm complexity and algorithms are trained faster.

2.2.4.1 Filter Method

Filter methods are generally used as a pre-processing step (Figure 7). Feature selection is independent of any machine learning algorithm. Instead, features are selected based on their scores in various statistical tests in which we study their association with the system output [25] .

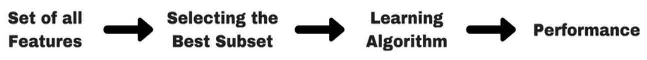


Figure 7: Filter Method for feature selection

2.2.4.2 Wrapper Method

In this method figure 8, we try to use a subset of features and train a model. Based on the conclusions we get, we decide to add or remove features from our subset. The problem is essentially reduced to a search problem. These methods usually require a long calculation time. [26]

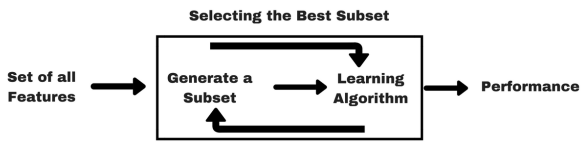


Figure 8: Wrapper Method

The wraparound method has 3 approaches:

- **Forward Selection**: Forward selection is an iterative method in which we start with no features in the model. In each iteration, we keep adding a feature that improves the performance of our model, until adding a new one stops improving it.
- **Backward Selection**: In this approach, we start with all features and remove the least important feature at each iteration that improves model performance. We repeat this until no improvement is observed when removing a feature.
- **Recursive elimination** of features: It is a greedy optimization algorithm that aims to find the subset of functions with the best performance. It repeatedly builds models and saves the best or worst performance each time. Builds the next model with the

remaining features until all are used up. It then ranks the features in order of their elimination.

2.2.5 Difference between filter and wrapper methods

The main differences between filter and wrapper methods in the feature selection process [27] are:

- Filter methods measure the correlation of with the dependent variable (output of the system), while wrapper methods measure the result of the trained model from using a subset of features.
- Filter methods are much faster compared to wrapper methods as they do not involve training the models. In addition to this, convolution methods also have a very high computational cost.
- Filter methods use statistical methods to evaluate a subset of features, while wrapper methods use validation methods.
- Filter methods may not find the best subset of features in many cases, but wrapper methods can always provide the best subset of features.
- Wrapper methods are more prone to over fitting models compared to filter methods.

2.3 Compressed Sensing

Compressed sensing, also known as compressive sampling, is a powerful technique for efficiently acquiring and reconstructing signals that are sparse or compressible in some domain [28]. By using a small number of non-uniformly spaced samples, it is possible to accurately reconstruct a signal that would otherwise require dense and uniformly spaced samples using traditional methods [29]. This is possible because compressed sensing leverages the sparsity of the signal, which means that it can be represented using a relatively small number of non-zero coefficients in some basis [30]. By reducing the number of required measurements, compressed sensing can significantly improve efficiency and robustness to noise [31]. In this section, we will delve into the mathematical foundations of compressed sensing, including the linear algebraic formulation and the concept of sparsity (Elad, 2010). We will also discuss various algorithms for compressive sensing, such as orthogonal matching pursuit and basis pursuit [32], and examine their performance and computational complexity [33].

2.3.1 Beating Nyquist

With compressed sensing, it is possible to significantly decrease the number of necessary measurements, which is much lower compared to the number specified by the Nyquist-Shannon theorem. This conventional theorem states that a signal must be sampled at a rate no less than twice the highest frequency present in the signal to prevent signal aliasing [34]. On the other hand, CS leverages the sparsity of signals in a specific basis or domain to enable accurate reconstruction with only a limited number of linear measurements, even at a lower sampling rate than what is prescribed by the Nyquist-Shannon theorem. This approach of signal acquisition makes CS a suitable solution for various applications where data reduction is a crucial factor.

We will proceed to see that by using compressed sensing, we can indeed sample at rates below Nyquist and then successfully reconstruct the original signal. Let's assume we have a high-resolution signal with duration of one second, consisting of 4096 points. Suppose the signal is a composition of 2 frequencies, 200 and 1000Hz. Based on the Nyquist criterion, we would want double the points defined by the maximum frequency contained in the signal, i.e., 2000 points. However, we see in Figure 9 that with significantly fewer points and without any optimization, we achieve a very good reconstruction.

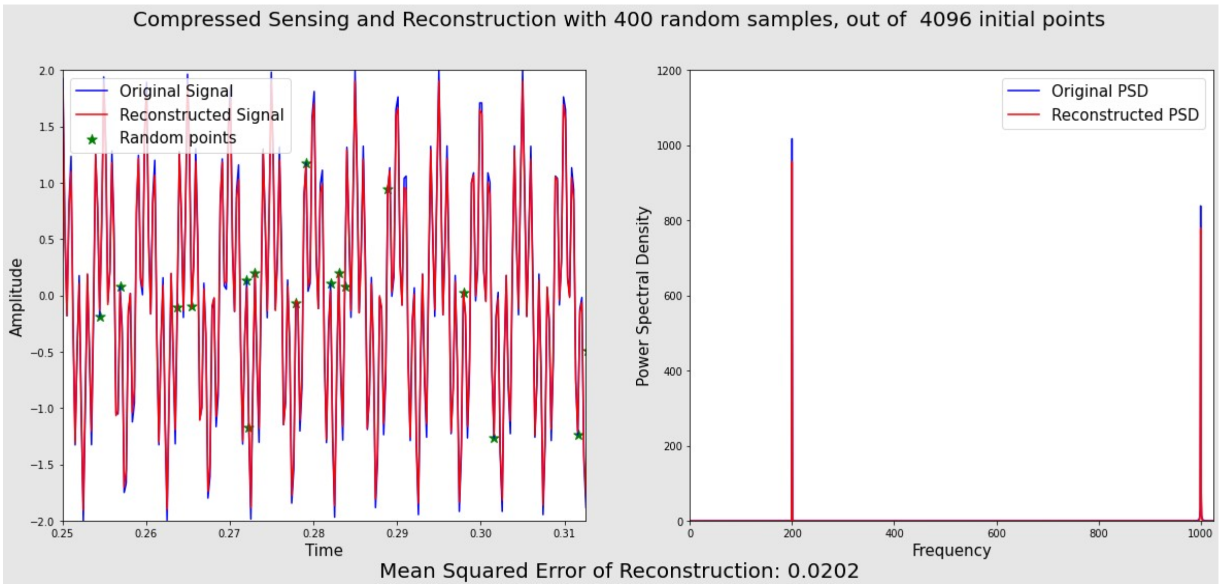


Figure 9: CS beating Nyquist

2.3.2 Random vs Non-Random measurements

A good observation is that the points are chosen from initial signal in a random way [34] and that is a very important issue in CS at the phase of signal acquisition. The reason for this is that, a random measurement selection provides high probability of achieving a good signal reconstruction with very good accuracy, even with limited (subNyquist) measurements. This is because random measurements are unlikely to be highly correlated with each other, so we have minimum coherence between the measurement matrix and the sparsity basis of the signal, as it will be discussed later in details. Random measurements selection also, helps to reduce the computational complexity of signal reconstruction algorithms and ensure their stability and robustness to noise.

Below we present the same signal with the same number of reached points but this time not in a random way but equally spaced points. The result is shown in **Figure 10**. It now becomes obvious that choosing the points in a random way is a necessary condition for successful signal reconstruction. At the left of the figure the signal is presented in the time domain and at the right is presented at the frequency domain. Especially at the frequency domain we can observe that the acquisition system is totally unable to perceive the frequency components of the signal.

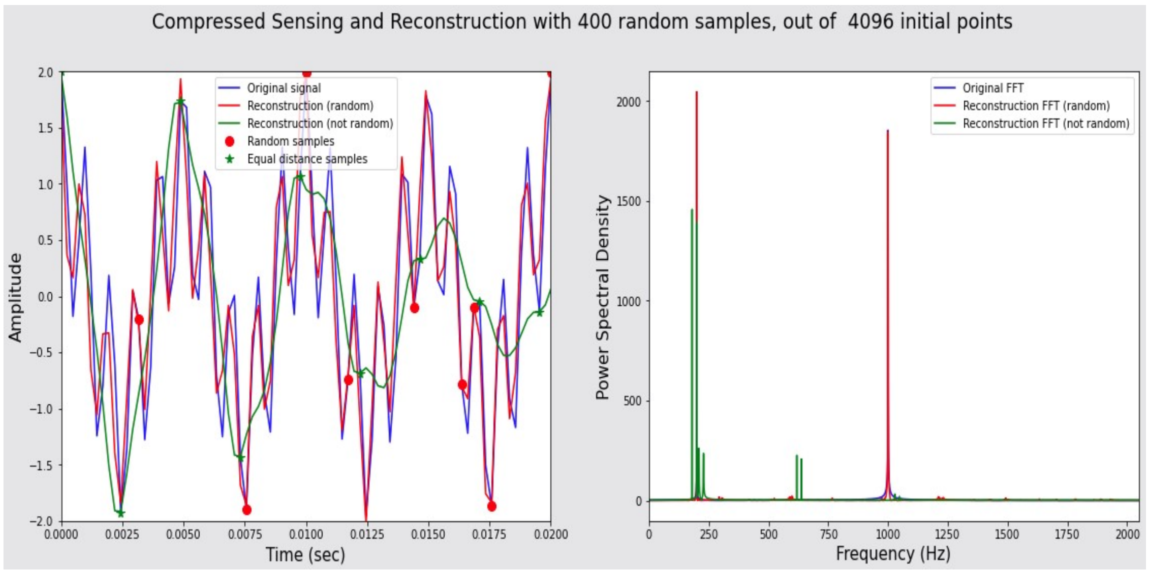


Figure 10: Random vs. not random sampling

2.3.3 Signal Domain and Transformation basis

Two key concepts that we need to clarify because they are very important in CS are signal domain and signal transformation/representation basis. Even these concepts belong to reconstruction subsystem we need to present them here because it will help us better understand the role of some basic points in the measuring subsystem.

When we talk about signal domain we are referring to the set of values that the signal can take on. To make it clearer we can measure a signal in the time domain that means that each sample of the signal corresponds to a specific point in time. A signal can also be defined in the frequency domain, in which case each sample of the signal corresponds to a specific frequency component [35].

On the other hand, a signal representation/transformation basis is a set of functions that can be used to represent/transform the signal in a particular domain [36] .For example, the Fourier basis is a common choice for representing signals in the frequency domain, while the wavelet basis is often used for representing signals in the time-frequency domain. To achieve transformation we need a transformation basis array usually called psi (Ψ) that transforms a signal to a specific base [36]. We will see more about that in details, in a following section of this report, because psi is a key point in the context of CS.

In the Figure 11 we can see a signal we got from the accelerometer of our micro-controller. The signal corresponds to the formation of a circle and is presented in time and frequency domain, in Fourier (DCT) and wavelet (DWT) representation basis in original and k-sparse form after thresholding.

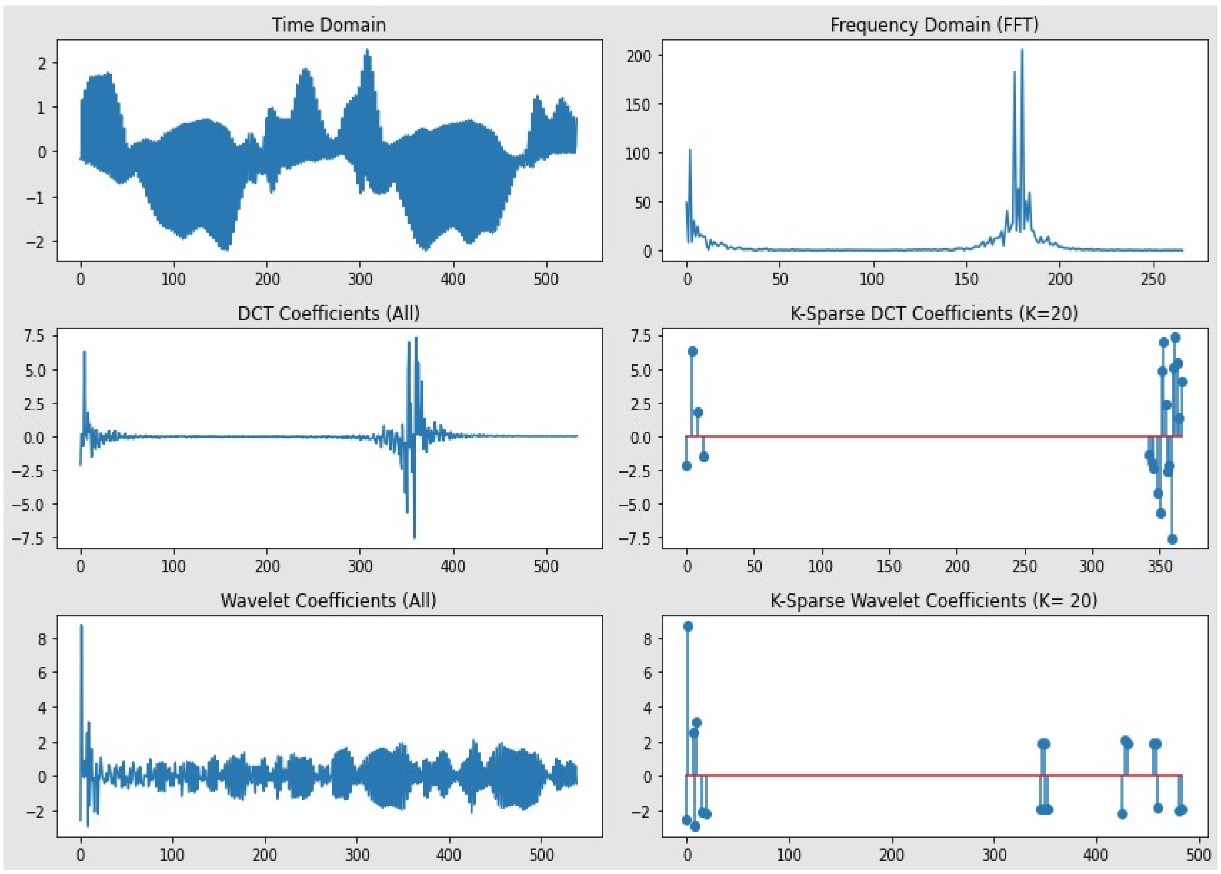


Figure 11: analysis of a signal x in time and frequency domain, DCT and wavelet basis

In the first line we notice that the representation in the frequency domain has far fewer significant values than our signal in its original form in the time domain. This leads us to assume that a frequency domain representation basis is a good choice. The first subplot of the second (DCT transformation coefficients) and third line (DWT transformation coefficients) should also confirm this hypothesis. We see that the values approaching zero (the least significant ones) in the DCT representation basis are much more than in DWT. In DCT representation basis it is obvious that my signal is **considered** sparser than in wavelet representation basis.

The second row at the second column of the Figure 11 plots the 20 strongest (also known as K-sparse, K=20) coefficients the DCT representation. Similarly, the 20 strongest coeffs of the wavelet coefficients are shown in the second subplot of the third row. The use of k-sparse representations, as said before, allows us to efficiently capture the essential information of the signal, providing a comprehensive understanding of the signal's properties that is crucial in various signal processing applications. Also by thresholding non important coefficients are turned to zero, and by that we have eliminated noise from the signal, because noise has low value coefficients in the representation.

In general, the choice of signal domain and signal transformation basis will depend on the characteristics of the signal and the goals of the analysis. An IMU signal, (accelerometer values) neither is periodic nor has particular frequency content, so it is not convenient to represent it in the frequency domain using the Fourier basis. On the other hand, the signal has localized features in time or frequency, it is more appropriate to represent it in the time-frequency domain using the wavelet basis. We will study to see which the best representation basis is, because it’s critical in compressive sampling but from a first look at the previous figure it intuitively seems that the DCT representation basis is more suitable.

2.3.4 CS Measuring subsystem

The measurement subsystem is responsible for taking a small number of measurements. The system is trying to capture the essential information about the signal with minimum number of measurements. This is achieved using a well-suited measurement matrix to our signal [37].

One of the structural elements of the compression subsystem is the sampling matrix A. If I have a signal x and a measurement matrix A, it will randomly select a subset of x to create the vector y. The size of A, and consequently the number of points, is proportional to the sparsity of the signal. The whole process can be represented by the equation: $y = A * X$ [38] and visually as follows:

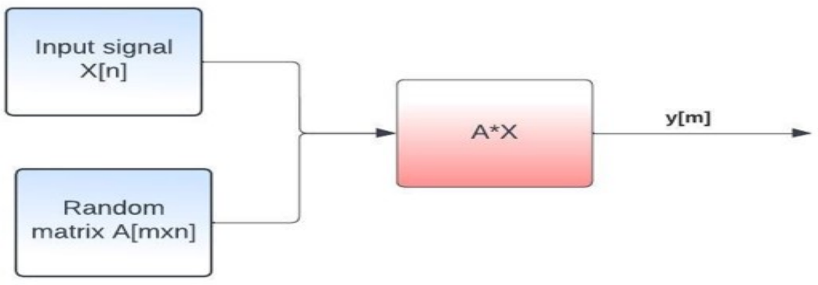


Figure 12: compressive sampling visual representation

Following the above, compressing a signal is a matter of matrix multiplication [39]. More specifically, if I have a vector x ∈ R^n, which contains my original signal (from an IMU sensor) and multiply it with a matrix A ∈ R^{mxn} with m << n, then a vector y ∈ R^m is produced, as shown in the following Figure 13 . This vector represents the compressed version of x. The points I obtain are non-adaptive, meaning they do not contain information about previous points due to their random selection. These fewer data points are very easy to store or transmit over a network to a sink node.

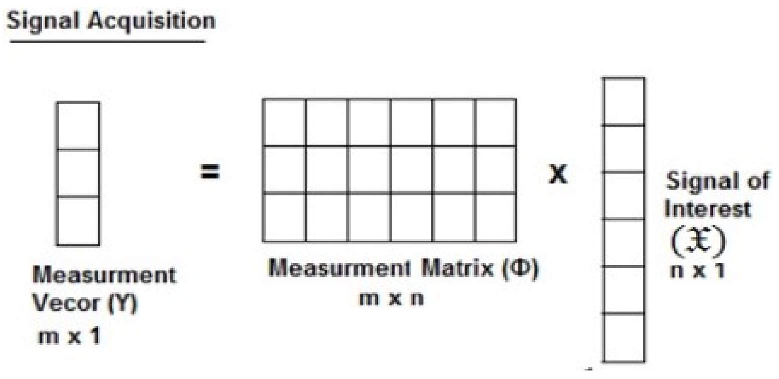


Figure 13: Compressive sampling as array multiplication

2.3.4.1 A good measuring matrix A

H CS can be summarized in the following seemingly simple statement: "a signal, if it is sparse enough in a given basis, can be recovered (with high probability) using significantly fewer measurements than its original, provided there are sufficient measurements and these measurements are sufficiently random" .

Each part of the above statement can be mathematically rigorously represented in a general framework that describes the geometry of sparse vectors and how these vectors are transformed through random measurements. A sampling matrix has certain properties, the values of which define its quality and, consequently, the quality of the reconstruction of the signal it samples [40].

2.3.4.2 Transformation matrix psi

A transformation matrix, in the context of compressed sensing (CS), is a key component. We use it to represent the linear mapping between the high-dimensional signal and its compressed measurement vector. That means that we map the signal from its high-dimensional domain (initial form) to a lower-dimensional measurement domain (transformation form), where the number of important measurements is much smaller than the signal dimension.It is very important and we will show that later, to note that the transformation matrix in CS must be incoherent [3] with the sparsifying domain of the signal, which ensures that the measurements provide a compressed representation of the signal that retains the essential information needed for reconstruction .

The choice not only as far as the coherence is concerned, but also about the type of matrix is crucial for the success of the CS recovery algorithm. A popular choice is a random matrix or a sub-sampled Fourier matrix, which have been shown to provide good performance in practice. Other examples of transformation matrices include wavelet and curvelet transforms, which are commonly used in image processing applications.

2.3.4.3 Coherence of A and Ψ.

A crucial issue in CS is the coherence of the sample matrix and transformation matrix. The degree of resemblance between the rows and columns of the measurement matrix and the sparse representation matrix is known as coherence. A suitable measurement matrix for compressed sensing should have minimal coherence with the representation matrix, which means that its columns should differ as little as possible from those of the sparse representation matrix. This makes it easier for the compressed sensing technique to precisely recover the sparse representation of the original signal and ensures that the measurement matrix is well-conditioned. In compressive sensing, coherence is used to rate the effectiveness of a sensor matrix A. A lower coherence value indicates that the matrix is more suitable for compressive sensing and will lead to better performance. Mathematically, it can be expressed as:

$$\eta(A, \Psi) = \max_{i,j} \frac{|a_i^T \Psi_j|}{||a_i||_2 ||\Psi_j||_2}$$

Equation (7) coherence of two arrays calculation formula [41]

Where a_i is the ith column of the matrix A and ψ_j is the jth column of the matrix ψ. The term $||a_i||$ represents the Euclidean norm of the ith column and the term $||\psi_j||$ represents the Euclidean norm of the jth column. The maximum is taken over all pairs of columns (i, j)

2.3.4.4 mutual coherence of A

The mutual coherence of the sampling array and the coherence of the sampling matrix and transformation matrix plays important role in determining the performance of the compressed sensing algorithm. The mutual coherence of a sampling array measures the similarity between the columns of the sampling matrix. It is wanted sampling array to have low mutual coherence, in other words, the columns of the sampling matrix are as dissimilar as possible. This ensures that the measurement matrix satisfies the restricted isometry property (RIP), which is a key condition for the success of the compressed sensing algorithm. Mutual coherence of a matrix A is a measure of the maximum absolute inner product between any two columns of A, normalized by their respective norms. Mathematically can be expressed as:

$$\mu(A) = \max_{i \neq j} \frac{|(a_i)^T a_j|}{||a_i||_2 ||a_j||_2}$$

Equation (8) mutual coherence calculation formula [41] where ai is the ith column of the matrix A and aj is the jth column of the matrix A and The term ||ai|| represents the Euclidean norm of the ith column and the term ||aj|| represents the Euclidean norm of the jth column. The maximum is taken over all pairs of columns (i, j)

In order to better understand the concept of coherence between matrices and the mutual coherence of a matrix, a visual representation is provided in the following Figure 14. Let's assume that a DCT transformation basis array psi has 50x50 elements. The sampling matrix A has 10x50 elements and consists only of 0s and 1s, with the constraint of having only one 1 in each row, the so-called random single pixel subsample array. Why this constraint? When A is multiplied by a signal x with 50 points, it will create a signal y containing 10 elements, and each element of y will be an identical element of x rather than a linear combination of some of them. Also, let's assume a sampling matrix A2, which consists of the first 10 rows of psi, an entirely parallel matrix. As we will show, this is a very poor choice and is likely to result in very poor outcomes. The following figure visually presents the A, A2, and Psi matrices.

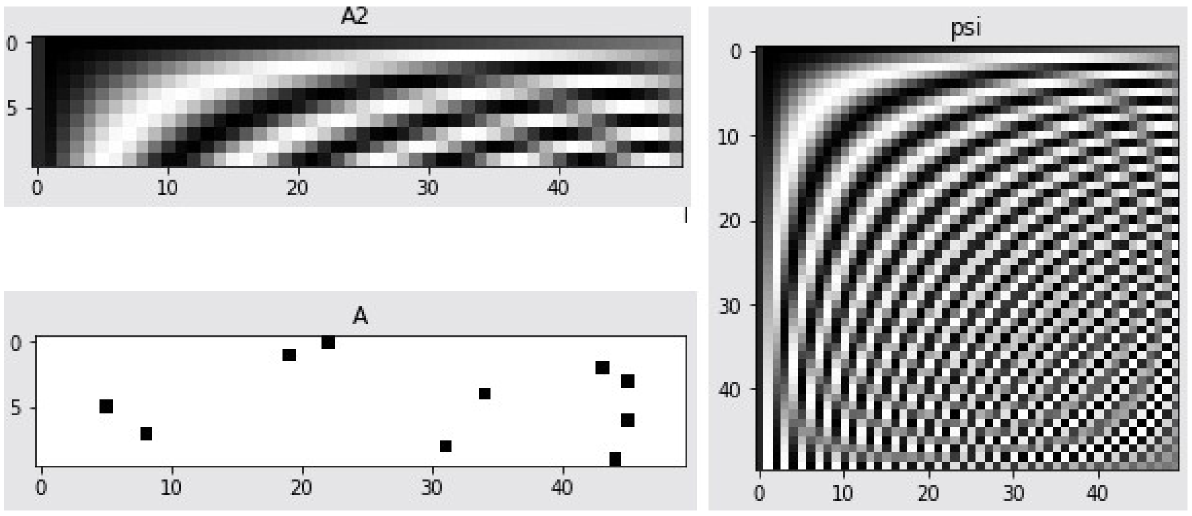


Figure 14: A, A2 sampling matrixes and psi 50x50 transformation matrix

We measure the mutual coherence of each matrix and the coherence between each sampling and transformation matrix. The results are presented in the following comparative

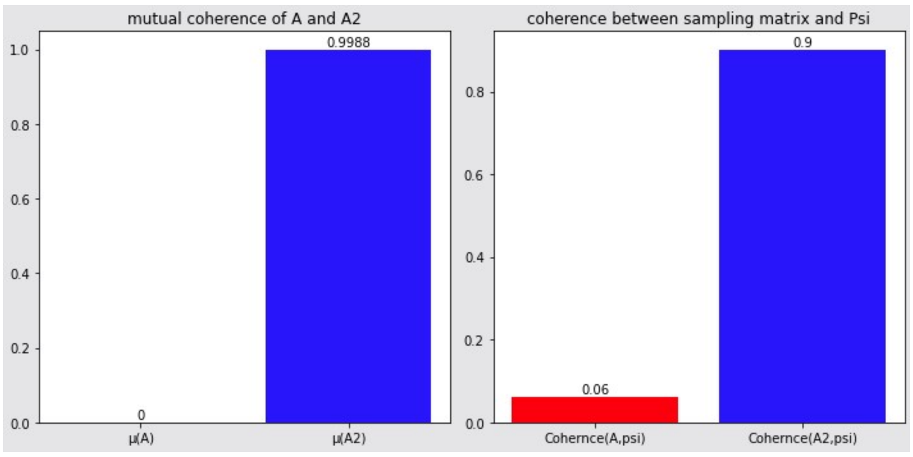


Figure 15: coherences between matrixes

One can easily realize that A is a very good choice; while on the contrary, A2 is a very bad one. It is obvious that A2 is completely parallel to psi, which is logical since it is an inherent part of it. More analysis will be done below in the sampling matrix section.

In Figure 16 are presented four types of good random measurement matrices A that are more frequently used [36].

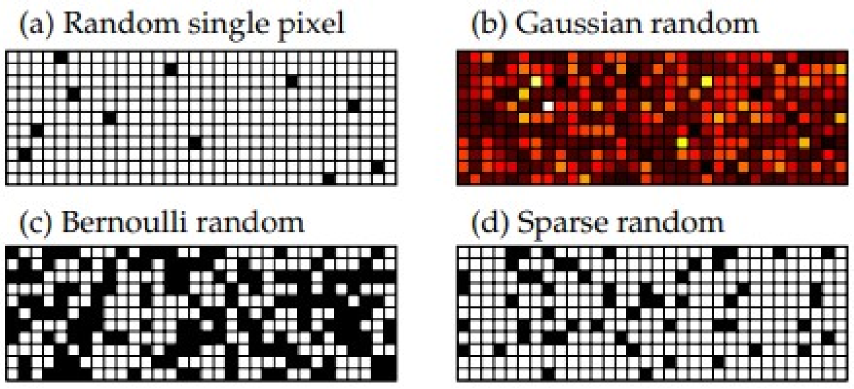


Figure 16: examples of random measurement matrices A

2.3.4.5 Number of measurements m.

The number of measurements that A will produce must be sufficiently large and is given by the following equation, given that x is k-sparse in a representation basis Ψ:

$m = \mu * K * \log(n/K).$

Equation (9). number of sample points to create y [42]

The m is a constant that depends on how incoherent the matrix A and Ψ are.

2.3.4.6 Traditional compression vs. CS compression.

In traditional compression, a signal is captured by an ADC at rates defined by the Shannon-Nyquist rule. However, when sampling at such a high rate, we have two main problems. First, the size of the signal is very large compared to the information contained within it , and second, there are signals (multiband signals) with a very large spectral range, making their sampling extremely difficult even for the best ADCs.

In traditional signal compression, the necessary transformation coefficients in a basis (DCT, Fourier, etc.) are calculated. From these coefficients, after sorting them in ascending order Figure 17, , we only retain the largest and most significant ones, while discarding the smaller ones for storage/transmission reasons. This raises the question, "why take so many samples when most of them have to be discarded during the thresholding process?'.

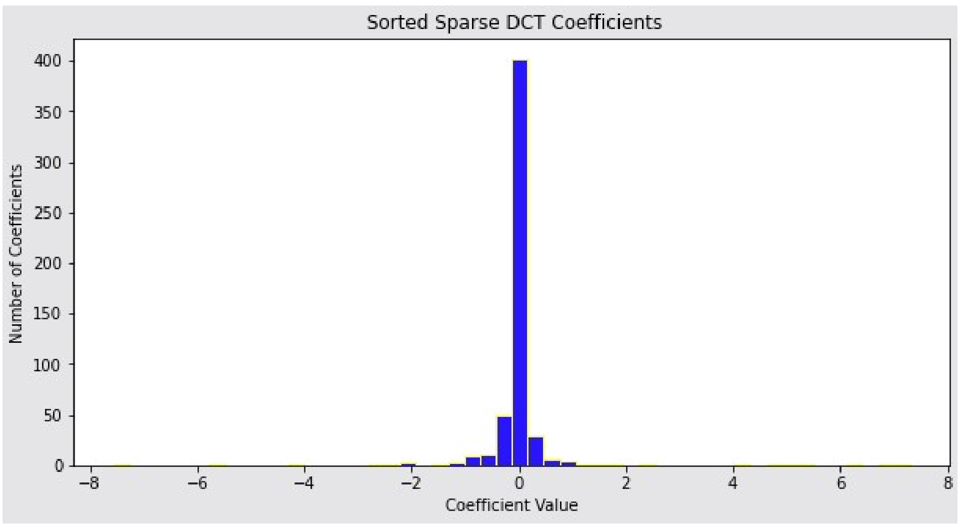


Figure 17: DCT coefficients sorted

Compressed Sensing (CS) is a more efficient method compared to traditional compression, especially when the compressed signal is to be transmitted over a data network [43]. Traditional methods have higher computational complexity and require more computational power and resources, consuming more energy compared to CS. This makes CS an ideal choice for microcontrollers located at the edge and extreme edge. Additionally, its low computational power requirements make it significantly faster. CS is also more tolerant to incoming noise in the signal compared to traditional compression methods. The differences between the two methods are illustrated in Figure 18.

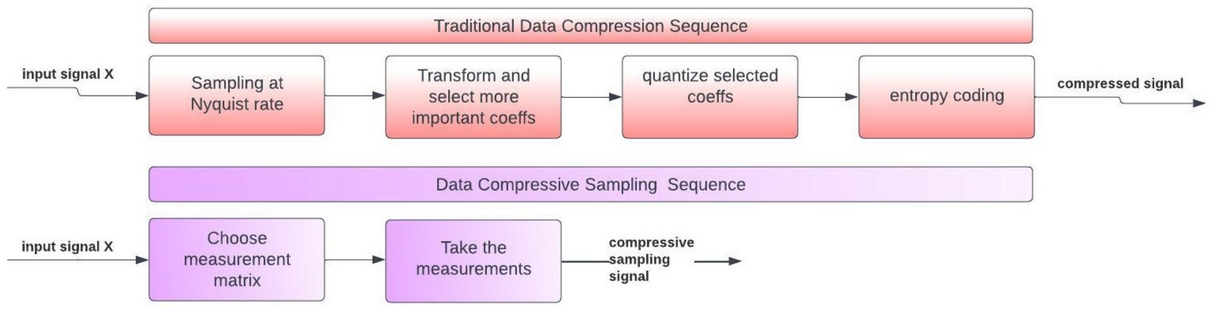

Traditional Data Compression Sequence
input signal X
Sampling at Nyquist rate
Transform and select more important coeffs
quantize selected coeffs
entropy coding
compressed signal
Data Compressive Sampling Sequence
input signal X
Choose measurement matrix
Take the measurements
compressive sampling signal

Figure 18: Traditional compression vs CS Acquisition

2.3.5 CS- Reconstruction Subsystem

The reconstruction subsystem tries to reconstruct the original signal with best accuracy, using only a small number of measurements that have been taken. This is achieved by various ways we will examine in the specific section of our thesis. The recovery subsystem can also incorporate prior information about the signal, such as its sparsity structure or its properties in a particular basis, to further improve the accuracy of the reconstruction.

2.3.5.1 Sparsity as a parameter

In the context of compressed sensing, a signal is **considered** to be sparse if it can be represented in some basis, using a small number of non-zero coefficients [44]. To make things clear, transforming a given signal in a particular basis representation does not make it sparse, but it is **considered** as sparse if most of its coefficients in that representation are close to zero.

The Discrete Cosine Transform (DCT) is often used to represent IMU signals in a sparse form because it of transforms signals into a frequency-domain representation, where most of the energy of the signal is concentrated in a few coefficients. These coefficients represent the main features of the signal and by applying a threshold on these smaller coefficients, it is possible to obtain a sparse representation of the signal. Also this process can be considered as noise removal. The thresholding process discards these small coefficients and only retains the larger ones, which represent the important information in the signal. The choice of the threshold depends on the desired level of accuracy, but typically, the threshold is set such that a small fraction of the total number of coefficients is kept.

2.3.5.2 Reconstruction

When the compressed signal y reaches the sink node, a good question is why we don't simply find the inverse of matrix A, given the equation y=A*x, so that x=A-1*y? The reason is that A is not a square matrix since m<<n, and it doesn't have an inverse. Moreover, the system of equations, as we will see, is indeterminate and requires complex methods to be solved.

As mentioned earlier, natural signals, such as sound or inertial signals, are highly compressible. This means that if they are transformed into an appropriate basis, they can be represented by a set of coefficients where most of them are zero or at least very close to zero (Figure 11). he non-zero coefficients are the ones we are interested in. Mathematically, let's consider a compressible vector x where x ∈ Rn. This vector can be transformed into a coefficient vector S as defined by the following equation:

$$x = \Psi * s$$

Equation (10) signal X transformation

Where Ψ is the transformation matrix (e.g., DCT, Fourier, etc.) that transforms the original signal x into the coefficient vector S. The vector S will have only a few non-zero elements (i.e., it is sparse) that capture the essential information of the signal. The goal of compressed sensing is to recover x (or equivalently, the sparse vector S) from a much smaller set of measurements y, using the knowledge of the sensing matrix A and the sparsity basis Ψ.

Combining the above equations, we have: Y=A*x=A*Ψ*s where we end up with the basic equation of Compressed Sensing.

Y==A*Ψ*s

Equation (11) main cs equation

The matrix $\Theta \in R^{mxn}$ is the product of A * Ψ. As we mentioned in the acquisition model, the Measurement matrix A represents m linear measurements of the original signal x. The system of equations defined by eq(11)has Θ and y as input data, and we are looking to find the vector s. Once we find it, we will substitute it into eq(10) to obtain the reconstructed original signal x. However, eq (11) has n unknowns, which are the coefficients of the vector s, and m equations, meaning that our system is non-deterministic since we have infinite solutions that satisfy our equation. A chosen reconstruction algorithm, also known as a solver, is fed with the matrix Θ and the compressed vector that arrived (in our case, at the sink node). The solver solves the non-deterministic system of equations and finds the sparsest s that solve eq (11). Then, through the inverse transformation process (e.g., IDCT if we had chosen DCT), we obtain the reconstructed vector x_hat. The reconstruction model is presented in the following diagram.

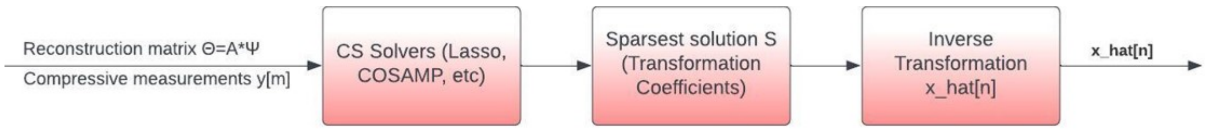

Reconstruction matrix Θ=A*Ψ
Compressive measurements y[m]
CS Solvers (Lasso, COSAMP, etc)
Sparsest solution S (Transformation Coefficients)
Inverse Transformation x_hat[n]
x_hat[n]

Figure 19: CS reconstruction model

2.3.5.3 Reconstruction Algorithms (solvers)

Various reconstruction algorithms have been developed, which rely on different approaches for calculating s. The most significant approaches are:

2.3.5.3.1 Convex-Relaxation.

This group of algorithms is based on solving the convex optimization problem through linear programming to achieve the computation of s and, consequently, the reconstruction of the original signal. In these methods, the number of measurements is small, but the calculations have high computational complexity. Some solvers of this class are BP, BPDN, LASSO, LARS [45]. Their basic philosophy is that we know that we are essentially dealing with an optimization problem in which we are searching for the sparsest s, i.e., the one with the smallest l1 norm that satisfies eq(11).

The l1 norm is given by the formula:

$$\|s\|_1 = \sum_{k=1}^{n} |s_k|$$

Equation (12) l1 norm calculation formula

2.3.5.3.2 Greedy Iterative algorithms

In addition to the l1-minimization implementation, there are also different approaches based on so-called greedy algorithms. In this class of algorithms, the reconstruction of the original signal is done step-by-step through an iterative process [45].

The basic idea is the selection of rows from the matrix Θ in a greedy way. In each iteration, the row found to have the highest correlation with y is selected based on the criterion of minimizing the least squares error. The contribution of the selected row in each iteration concerning y is weighted, and the row is removed from the rows under consideration. This continues until the desired subset of Θ is selected. The criterion by which the algorithm stops varies depending on the implementation.

More commonly used greedy algorithms are MP, OMP, and COSAMP. Greedy algorithms provide fast and accurate reconstructions with relatively low computational complexity. However, they are sensitive to parameter selection and may not always provide the optimal reconstruction.

2.3.5.3.3 Sparsity and lp norm minimization(l1 vs l2 minimization norm)

The choice of the method for solving the non-deterministic system of equations to find the sparsest s, which is also our solution, plays a crucial role. Next, a solution resulting from the resolution of the l1 and l2 minimization norm is depicted in Figure 20. The first row shows the coefficients of the S of a signal, calculated with the l1 method on the left in blue and the l2 minimization norm on the right in red, respectively. The second row displays the values of the coefficients of the vector s in the form of a histogram. It is evident that the vast majorities of the coefficients with the l1-minimization approach are zero or at least very close to zero. This does not happen with the l2-minimization. We easily conclude that the l2 norm approach is not capable of creating a sparse vector s, and therefore the reconstruction of the original signal is impossible. The l1 method, on the other hand, is the one that creates the appropriate s.

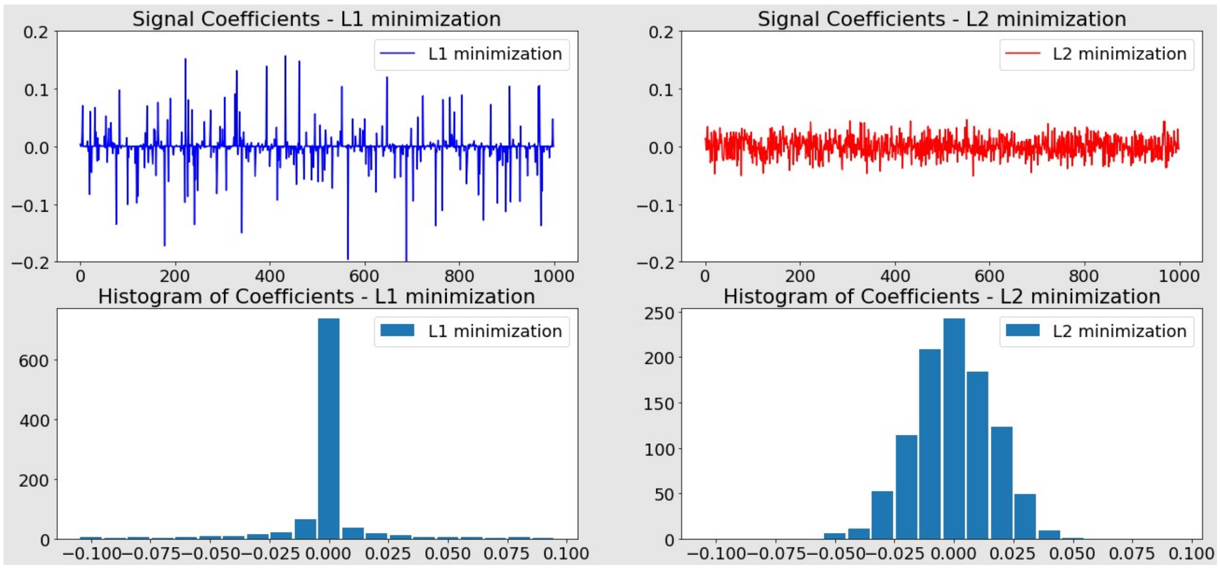


Figure 20: l1 vs l2 minimization norm

2.4 Related Work

Compressed Sensing (CS) and Machine Learning (ML) are two rapidly growing research fields that have shown to offer a lot in a broad spectrum of applications. There is a significant intersection between these two fields, as CS techniques are increasingly used to improve the efficiency and accuracy of machine learning algorithms, while machine learning is utilized to enhance the performance of CS algorithms. In this section, we will explore the relationship between these two fields and provide a comprehensive overview of the research conducted concerning the application of CS in machine learning. Our goal is to synthesize the main findings and contributions of this body of work and demonstrate the potential for further collaboration and integration between CS and machine learning.

In a distributed machine learning system, the process is divided into various nodes within the network [16]. Each node performs a specific task. In this case, we have the signal acquisition node (leaf node) which, for the ML system, is the data collection node. Simultaneously, we also have the leaf node responsible for executing learning and prediction tasks. By separating the system into multiple nodes, we have parallel computation, leading to improved performance and scalability.

In the present study, the impact of CS on the performance of a trained ML model was thoroughly investigated. The model was an SVM classifier, which performed multi-class signal classification. The classifier could recognize incoming signals corresponding to human movements (triangle formation, etc.). Dimensionality reduction of the elements constituting the dataset is a necessary process, and there are several approaches for this. In our approach, a selection of the most significant features with the highest relationship to the system's output was performed. The dataset resulting from the reduction of features was then used to train the model. It should be noted that this reduction occurs at the data collection node and allows for better generalization and overall behavior of the studied classifier. Subsequently, we had a further reduction of elements for transmission by applying CS to the already reduced data. Our approach, leveraging CS in machine learning, has several advantages.

First, using a feature selection method with grid search, we ensure that the most potent features are selected for compression, reducing the volume of data that needs to be transmitted. This not only saves time and transmission costs but also improves the accuracy of the learning algorithm by reducing noise and redundancy in the data.

Secondly, by performing compressed sensing on the selected features, we effectively compress the data before transmission, allowing for more efficient data transfer to the leaf node. This can be particularly useful in scenarios where the data is large, and the transmission bandwidth is limited. With data compression, the time and resources required for transmission are reduced.

In addition to the above, the use of compressed sensing in machine learning can also enhance data privacy and security by reducing the amount of data that needs to be transmitted. In scenarios where the data contains sensitive information, it is essential to minimize the volume of data transmitted to reduce the risk of data breaches.

In the context of our work and through the aforementioned approach, we study the determination of the compression ratio's impact on our SVM model's recognition ability. In other words, we determine the minimum number of measurements that must be taken from an already compressed signal so that the reconstructed signal obtained with the conventional compressed sensing method can be recognized by the SVM. It should be emphasized that the SVM is trained on uncompressed data. Another aspect of the study had to do with whether and to what extent the feature selection method matters before compression. If we had chosen a different feature selection method, e.g., Pearson correlated feature removal, would the minimum number of measurements from the original signal be the same? Would we have a similar impact on the model's performance?

Subsequently, we will present works in the same area as ours. An effort will be made to highlight the different approaches and the way they exploit the two technologies. By comparing these approaches, we can gain insights into the effectiveness of various feature selection and compressed sensing methods in different scenarios and applications. This comparison can help identify the most suitable techniques for specific situations, further improving the performance and efficiency of machine learning models while preserving data privacy and security.

In Zonzini's work [46], an effort is made to address the challenges faced by the Structural Health Monitoring (SHM) of a construction project and to improve the accuracy and quality of the estimated structural integrity of the project from existing machine learning models. To achieve this goal, they propose a framework that incorporates data compression and machine learning. The authors first apply compression to the data using a method called MRAK-CS to reduce their dimensions and minimize the risk of network congestion during transmission. The compressed data, after being received, is used as input in Artificial Neural Network (ANN) and One-Class Convolution Neural Networks (OCCNN) algorithms with the aim of reconstructing the original signal and subsequently performing damage detection on the reconstructed signals using binary classifiers.

The authors evaluate the impact of the data compression ratio on the reconstruction model's performance and how it changes with the variation of this ratio. In addition, they assess the impact of low-cost sensors and the inclusion of temperature data as input to the network. The results show that the combination of data compression and optimized machine learning algorithms can achieve high classification scores, with precision and accuracy greater than 96% and 95%, respectively.

The sequence of feature selection and compression applied in this study is first data compression and then sending the data to the sink node. Once the data arrives, it is reconstructed, and then feature selection is performed, which feeds the neural network for damage detection.

A similar approach to [46] is also presented in work [47], which also focuses on the monitoring of the structural state of construction projects. In this work too, after compressive sampling, transmission and reconstruction were carried out with the help of a neural network, with the difference that the network was transparent and integrated basic parameters of CS (sampling matrix) into a specific layer. By conducting experiments on sensors placed on a metal bridge, they proved that reconstruction using machine learning is more effective than the conventional method of solving the non-deterministic system. It should be noted that the output of the neural network was the reconstructed original signal, which was processed by a machine learning model making predictions. The latter is a common point in all works incorporating machine learning tasks in the sink node and those presented in the current section, including our own.

A new concept of distributed compressive sampling is introduced by Palagni and Deng [48]. The work combines two techniques, distributed CS and deep learning, for sampling and processing signals in a distributed manner. Compressive sensing is a way of acquiring signals using fewer data, which is useful when there is limited storage space or bandwidth for transmission. The method uses multiple nodes for signal acquisition and a deep learning network for reconstructing the signals from the compressed data.

In Zhang's work [49], the authors propose a deep learning approach for CS electrocardiogram (ECG). Traditional compressive sensing methods have limitations regarding reconstruction accuracy. To overcome these limitations, the authors combine CS and ML. The approach proposed in the work, called CSNet, consists of a deep neural network that takes compressed ECG measurements and reconstructs the original ECG signals. The results show that the combination of CS and deep learning leads to better reconstruction quality and computational efficiency compared to traditional CS methods.

In Tran's work [50], the concept of "compressive learning" combines the ideas of compressive sensing and machine learning for the analysis of multidimensional signals such as video and images. The authors propose a new way of compressing signals that takes their structure into account, leading to a more efficient approach. They then use machine learning to analyze the compressed signals and train models to perform tasks such as object classification and face recognition. The authors demonstrate that their approach works better than traditional methods and is more effective for high-dimensional signals. By combining CS and ML, we have an improvement in the way signals are analyzed.

3 Design of the system

Within the framework of creating the studied system, a set of suitably configured and parameterized software and hardware was used. The overall development followed a process of distributed information management in the standards of Fog computing.

Our system is a distributed machine learning system where capturing and recognition take place at different nodes of the network. At extreme edge nodes, often referred to as leaf nodes, capturing occurs, and recognition takes place at a central node (sink nodes) of the implementation. However, some of the leaf nodes can also perform recognition of signals they capture, depending on the needs of the implementation.

With the signal capture, the transmission of the captured information to the sink node is required. The information transmission requires compression before transmission to save energy during sending from the leaf node and to reduce the occupation of the communication sub-network nodes during transmission. For reasons explained in detail in this report, CS outperforms conventional compression, and that is why it is used. The leaf node, in addition to capturing and, in some cases, recognizing the signal, is required to implement compressive sampling before sending the signal to the central node. In the sink node, we have the reception and reconstruction of the original signal before it is given for recognition to the machine learning subsystem it possesses.

In the following section, a presentation of the software and hardware of our system will be made, both at the Leaf and sink node levels. The central design idea will also be presented in detail, along with a comprehensive representation of the topology and mode of operation of the system.

In Figure 21 is a graphical representation of the topology and operation of our system. Through this figure, we can see in detail and in-depth the entire philosophy of the design of the system under study.

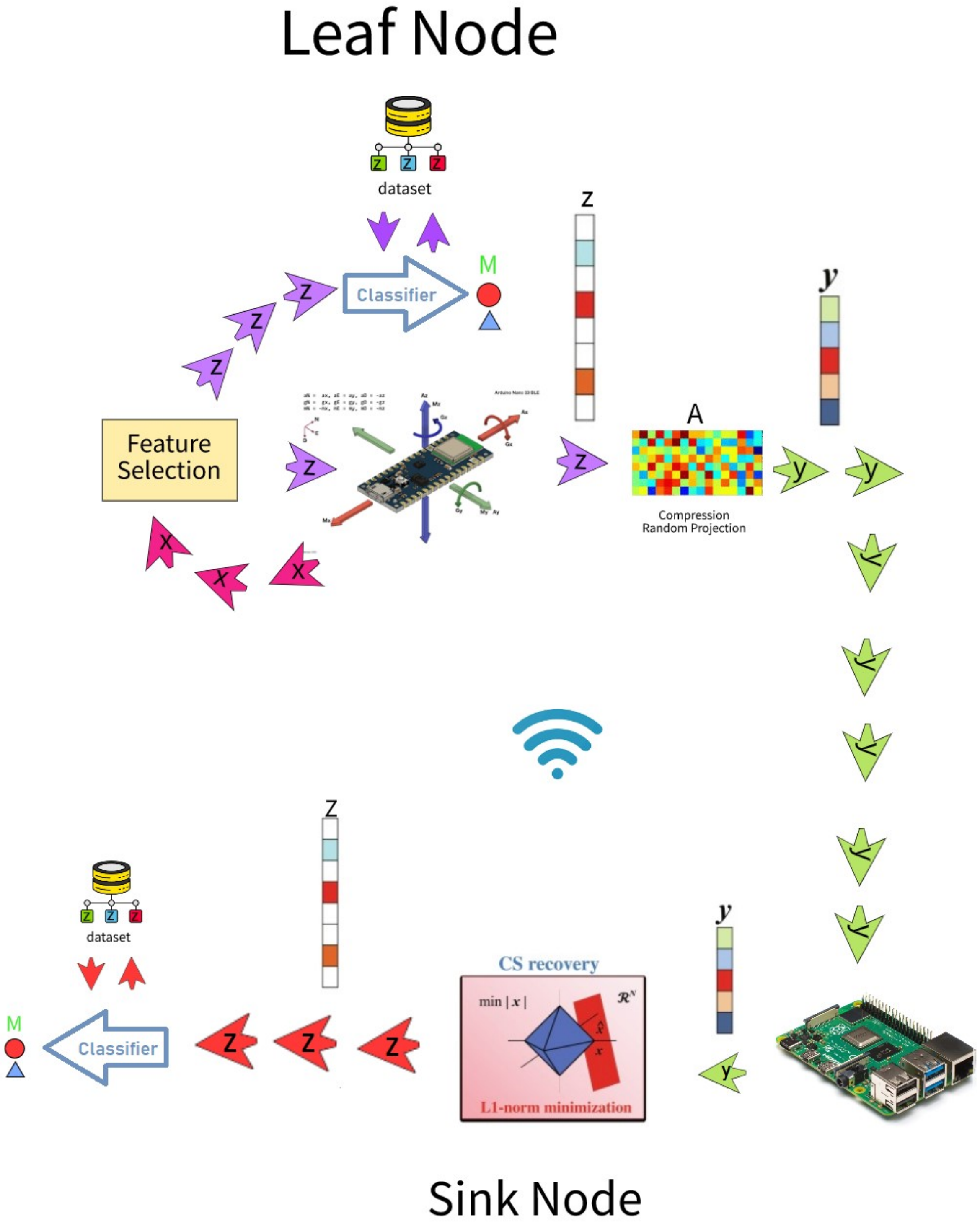


Figure 21: Topology and functionality of the system

3.1 *Sink Node*

3.1.1 Hardware sink node

As mentioned earlier, at the center of our topology is a Raspberry Pi 4 Model B. The purpose of the node is to collect incoming signals from the extreme edge nodes and reconstruct the original signals using CS techniques. It then effectively recognizes them with the help of an SVM trained classifier it has. The Pi has the resources to implement the above requirements, and that's why the final result is a full-functional distributed machine learning implementation.

The remaining features are summarized below:

- CPU: Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- RAM: 2GB, 4GB, or 8GB LPDDR4-3200 SDRAM (depending on the model)
- Connectivity: Gigabit Ethernet, 2.4/5.0 GHz 802.11ac wireless, Bluetooth 5.0, BLE
- Storage: microSD card slot for loading operating system and data storage
- GPIO: 40-pin GPIO header, populated

3.1.2 Software sink node

The sink node, in addition to processing and classifying incoming data using an SVM classifier, must also reconstruct signals using compressed sensing techniques and solve problems using algorithms such as Lasso, OMP, and CoSaMP.

The following software components are required to meet these objectives:

- Operating System: At the basic operating system level, the node has the most recent operating system installed, Raspbian GNU/Linux 10 (buster).
- Python: The primary programming language for implementing the machine learning model and running the SVM classifier, as well as for signal processing and reconstruction tasks.
- NumPy: A numerical computing library for Python, used for data manipulation and computation.
- Pandas: A library for data analysis and manipulation in Python.
- Scikit-learn: A machine learning library for Python, which provides implementation of various machine learning algorithms, including SVM.

- Scipy: A scientific computing library for Python, used for signal processing and optimization.
- SciPy Optimize: A module within Scipy that provides optimization algorithms, including Lasso and OMP.
- CoSaMP: A library for compressed sensing, used for signal reconstruction.
- Matplotlib: A plotting library for Python, used for visualizing the results. (optional)

This software stack provides the Raspberry Pi 4 Model B with the necessary tools to perform the tasks of signal processing, reconstruction, and classification in a distributed machine learning system.

3.2 Leaf Node

3.2.1 IMU signal

A digital signal is a sequence of discrete values in time. These values are a subset of the values of the analog signal which by its nature contains infinite points. It is precisely this characteristic of it that must be eliminated with the help of digitization. The process is called analog to digital signal conversion (DAC) and it rests on two main pillars. Sampling is the measurement of the accelerometer value, in each of the three axes at equal moments of time (sampling period T) [51].

An Inertial Measurement Unit (IMU) is a sensor that measures acceleration and angular velocity. These measurements can be represented as time series data (Figure 24), with the measurements being taken at discrete time intervals.

Time series data is the data that is collected over time, each point represents a measurement taken at a specific point in time. In the context of machine learning, time series data can be used to train models that can make predictions about future events based on past observations. For example, an IMU time series data might be used to train a machine learning model to predict the type of activity being performed (e.g. walking, running, jumping) based on the past measurements of acceleration and/or angular velocity. In this thesis we used the

LSM9DS1 IMU that is embedded in the nano33 ble sense micro-controller. Figure 22 shows the concept of accelerometer and gyroscope measurements.

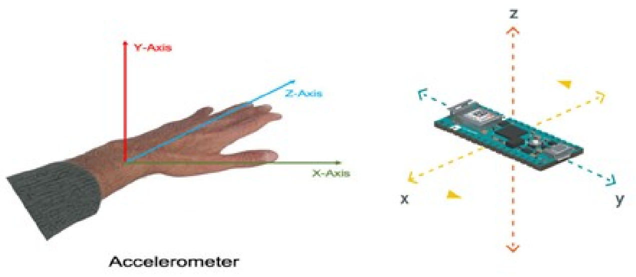


Figure 22: The concept of accelerometer

In Figure 23 we can see the values of the accelerometer sensor as they change during a movement. In the example we see, the letter M is formed with the sensor at a sampling rate of 119Hz. The accelerometer values for each axis are plotted separately, while in the Figure 24 they are presented as a unified time series signal formed as:

[accx1,accy1,accz1,…,accx178,accy178,accz178]

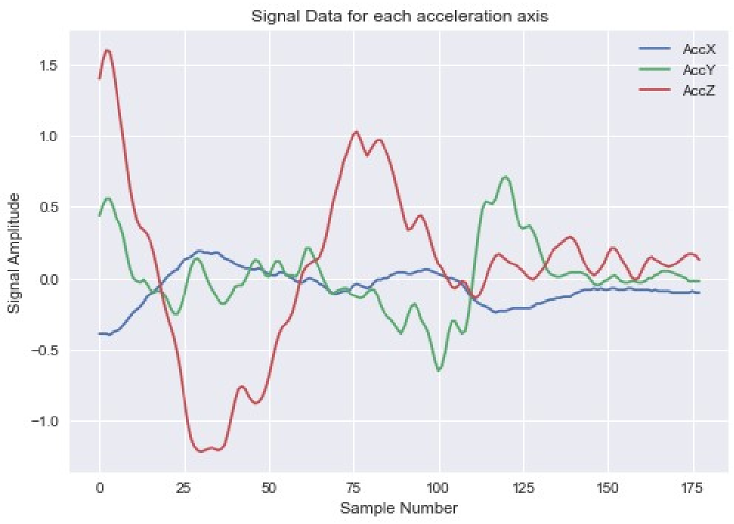


Figure 23: plot of each accelerometer axis

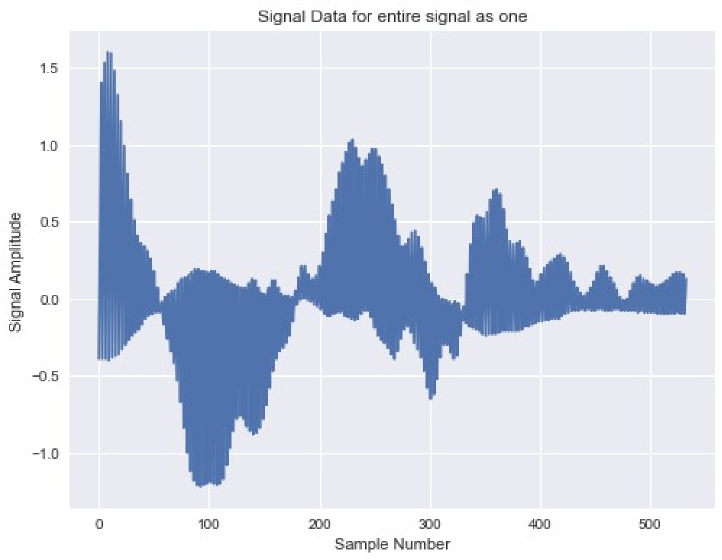


Figure 24: plot of measurements as one time-series signal

In the following figure (Figure 25) the IMU measurements that respond to the three different moves (M-shape, circle, and triangle) that we will study in this research are plotted. They are presented in time and frequency domain representation. Time domain representation shows us the signal's dynamics over time, while frequency domain representation, shows us the frequency components that are forming the signals and are obtained through FFT. By analyzing both representations, we have a better understanding of the signal's properties and behavior [52]. The figure clearly showcases the time and frequency domain of the three IMU signals, enabling easy comparison.

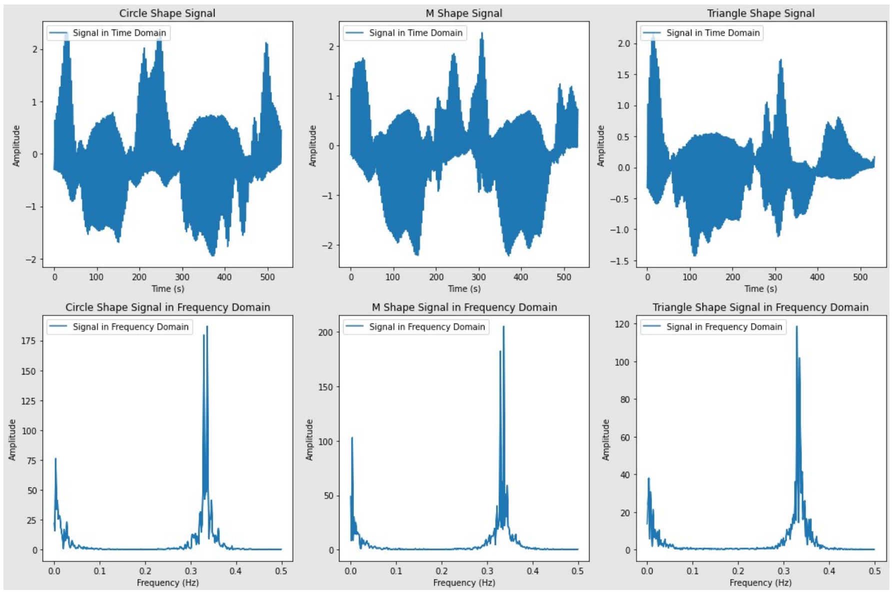


Figure 25: the three movements of our ML model presented in time and frequency domain

3.2.2 Hardware leaf node

The Arduino mini 33Ble sense, a board with a 32-bit ARM Cortex-M4F CPU operating at 64 MHz, is located at the leaf node. Additionally, it features 256 KB of RAM and 1 MB of flash memory. The IMU of the Nano 33 BLE Sense is one of its major components (Inertial Measurement Unit). A 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer make up the IMU. These sensors can be used to detect acceleration, measure angular velocity, and measure the strength of the magnetic field, respectively. In addition to these sensors, the Nano 33 BLE Sense also features pressure, humidity, and temperature sensors. The IMU and these sensors make the Nano 33 BLE Sense a potent tool for projects involving wireless. As about the sensor of our interest the IMU, LSM9DS1 is a system-in-package featuring a 3D digital linear acceleration sensor, a 3D digital angular rate sensor, and a 3D digital magnetic sensor. The LSM9DS1 has a linear acceleration full scale of ±2g/±4g/±8g/±16g, a magnetic field full scale of ±4/±8/±12/±16 gauss and an angular rate of ±245/±500/±2000 dps. Also includes an I2C serial bus interface supporting standard and fast mode (100 kHz and 400 kHz) and an SPI serial standard interface.

3.2.3 Software Leaf node

The leaf node is located at the extreme edge and its role is multifaceted. It is a data collection node from the inertial sensor of the microcontroller, on which it applies CS before sending them to the sink node. At the same time, it should be able to recognize human movements, the formation of a triangle, a circle, and the formation of the letter "M" with the help of a trained machine learning model.

To implement the above functions, the machine learning system and the compressed sampling system had to be developed. These systems, in terms of finding their basic implementation parameters, as well as their mode of operation and how they were implemented, are described in detail in the next section of the implementation.

3.3 Designing, the basic idea

The basic idea of the system under development in this work is the following: We are in a distributed machine learning (DML) operating model, and we are interested in studying a signal which is initially captured at an extreme edge node of the network and undergoes CS. After its reconstruction, we want to determine to what extent it would be recognizable by the ML model of the central node that received it. We can see it in details in Figure 21.

The basic design philosophy for setting up the system that would allow us to get answers to the main questions of this work was simple. In our DML system, the capture and CS should be done at the leaf-node. The initial signal would be captured and stored in a vector X. This vector, in our system, undergoes processing at two levels.

- First, by the machine learning subsystem, where the absolutely necessary features will be selected to make the signal recognizable. This feature selection process is desirable not only to reduce the size of the data but also to eliminate features that convey the same information. This results in increasing the performance of the ML model. We will call Z the vector that contains the feature-selected data. The size of Z is much smaller than that of X, and an extensive study is done on the maximum possible number that can be removed. Our goal is to have the best ML model without over-fitting, so it can generalize well.
- Next, by the CS system, where the vector y will be created with a random selection of a significantly smaller number of points from the vector Z with the feature-selected data,

always based on the principles and philosophy of CS. It should also not remove too many points to reconstruct the signal well (low MSE), which will ultimately be recognizable. The trade-offs are quite significant at both the machine learning level and the CS level.

A challenge we faced, which was within the scope of the contributions of this work, was to make the leaf-node capable of making predictions, as can happen in a DML system. The predictions should be made with an effective model but at the same time capable of running on extreme edge devices, which are characterized by very low computational capabilities. For this purpose, a technique was developed that used traditional machine learning techniques, as will be analyzed in the next part of this report.

The role of the sink node would be to perform the reconstruction of the original signal from the compressed measurements that just arrived and make predictions for the signal that has just been reconstructed.

To study the exploitation of compressive sampling in the performance of a trained machine learning model, the present system was created, a graphical representation of which is presented in Figure 21.

This page has intentionally been left blank.

4 Implementation of the system

As can be understood, the development of an IoT system with edge and extreme computing features that processes and analyzes data from suitable sensors is one of the burning issues of our time. One of the most modern and powerful trends today is that of DML. It is a distributed system for signal capture on some nodes and making predictions on some other nodes of a network. The integration of machine learning models into extreme edge nodes of the DML is of utmost importance.

The transmission of information between nodes in such a distributed system is desirable to be done with the least possible energy consumption and computational power (which is limited anyway) and with the least possible occupation of the nodes of the communication sub-network. The application of CS meets this need.

In this section, we present the way in which the system was developed with the help of which we conducted the study. The presentation includes the machine learning and compressive sampling systems developed for both the leaf node and the sink node of our approach.

4.1 Machine Learning System

In this section, we will analyze how an offline classifier was created, which can distinguish human movements. The selection of movements was made with the criterion of having samples of different frequencies, e.g., the formation of the letter M is faster than that of a circle. We chose movements that cause abrupt changes in accelerometer values and some that do not. At the same time, we have movements with a high degree of similarity, such as the letter M and the triangle.

For the implementation of the ML system, which will be able to operate on microcontrollers with limited capabilities and features at the extreme edge, the following functions were implemented:

- Development of the data collection system and feature extraction for creating datasets (feature extraction subsystem). This subsystem operates at the extreme edge and captures signals from the microcontroller's IMU, proceeds to the appropriate arrangement of the signal values, and then stores them in a CSV file to form the dataset for training our model.

- Training of the machine learning model. This is an end-to-end project that carries out a thorough study and ultimately creates the most capable offline model based on various evaluation metrics. In the end, hyper tuning is performed, and the model parameters are extracted.
- Development of a system for extracting an optimized prediction model from the model parameters that were extracted and the final integration into the microcontroller and conducting related experiments.

The system developed can easily constitute a significant part of a broader distributed computational intelligence system between edge and extreme edge devices.

4.1.1 feature extraction subsystem and data-set creation

In machine learning, data is perhaps the most important parameter, even more so than the model. Statistician George Box has said, "all models are wrong, but some are useful." It is evident that collecting data in the wrong way and inadequate processing ensures that the model to be developed will not be useful at any level.

4.1.1.1 Defining basic parameters, SR and movement duration.

There is no one-size-fits-all answer to the question of what is a good sampling rate for an IMU (inertial measurement unit) to sample human movement, as the optimal sampling rate will depend on the specific requirements and constraints of the application. The key factors weighed in order to decide on a sampling frequency for this thesis were:

- Frequency of human motion: a human move has frequencies up to around 10 Hz, so according to Nyquist, sampling rate of at least 20 Hz is recommended capture the motion without the presence of alias. If we use higher sampling rate we can capture more dynamic or high-frequency movements. For example, a higher sampling rate may be needed to accurately capture fast or complex movements, while a lower sampling rate may be sufficient for slower or simpler movements [51] .

- Accuracy and Precision: When using higher sampling rates we generally result a more accurate and precise measurement, as the IMU to captures more data points and the changes in the measured variables more quickly. Sampling at a higher rate however, requires more processing power and can generate more data, which may be challenging to store and transmit [52].
- Compressive Sensing techniques are more effective when applied to large-dimensional data. Thus the percentage of data that will be randomly selected may be small relative to the original signal size but will be large in order to recapture the original signal successfully.
- The desired time: The sampling rate will define the time resolution of the measurement and that means the size of the captures signal, so it is important to consider the desired time resolution when selecting a sampling rate
- The trade-off between accuracy and efficiency: The selection of a sampling rate is a trade-off between accuracy and efficiency becouse higher sampling rate may provide more accurate measurements, but also may require more processing power and storage, which can be more costly and time-consuming.

In the figure Figure 26: Acceleration sampling points at different **SR** we used our sensor to make a circle and and we sampled this movement at 3 different sampling frequencies, 119, 90 and 50Hz. Then at each sampling frequency we plot the value of the accelerometer amplitude on each axis and in three different subplots, one for each axis.

To perform one of the three desired movements (circle, triangle, and letter M) we saw that approximately 1.5 seconds were required. Depending on the sampling rate, a different sample size was created each time in the time of 1.5 seconds. For example with SR=119 in 1.5 seconds we had a sample size of 178 points on each axis, since I have 3 axes my final sample would be 1.5*119=178*3=534 points. Similarly for 90 it would be 1.5*90=405 points.

At this point we should mention that, the number of points (sample size) defines the size of the data set on which we will train our machine learning model. That's because we will take every measurement of the accelerometer in each axis as a feature. Taking measurements over time as machine learning features is not the best approach to design a classifier but it creates long enough vectors (samples) to see the affect of compressive sampling to model prediction ability. We will also see that we will create a model that is very accurate.

Following the above, the sampling rate of the project was set at 119Hz.

In the following figure we see that if we keep the number of features 534 and sample at various SRs we see that the lower the sample rate the larger the sample we get. This means that for example at a rate of 50Hz for 178 sensor points we get a signal about 3.5 seconds long. Since the average time to make a move is 1.5 seconds we have an excess of 2 seconds of noise. This is a big problem. we observe the same (to a lesser extent for sampling at 90hz).

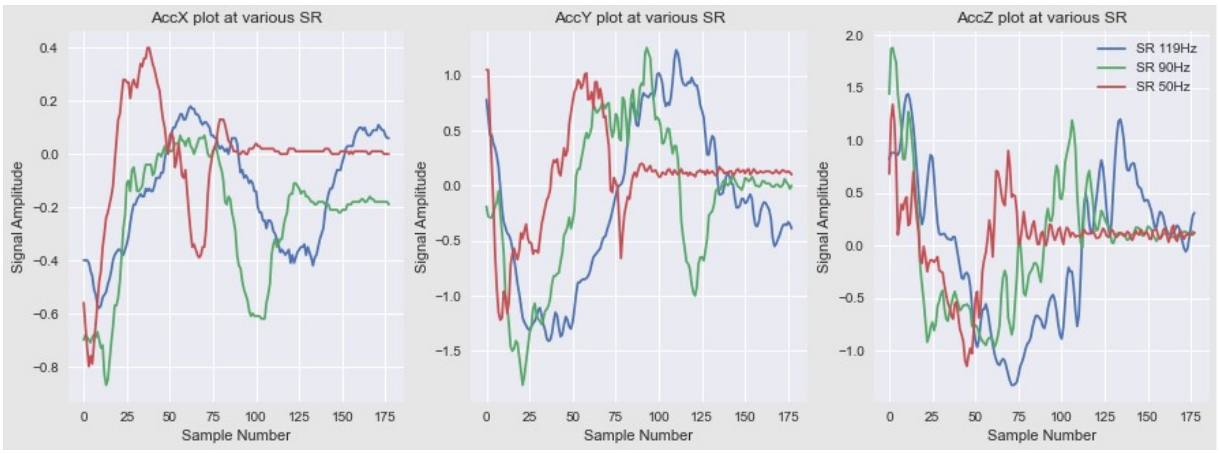


Figure 26: Acceleration sampling points at different SR

Following the above study, the basic parameters of the feature extraction system were clarified, and the appropriate values were extracted. In the system developed on the microcontroller, when a movement is detected, with acceleration above a threshold, 178 values are recorded for each of the three axes. These values are incorporated into a vector of 534 values and are simultaneously stored in a CSV file.

An issue regarding the best arrangement of accelerometer values was extensively studied, and the study is presented in the next section. In the following graph, the same signal is depicted, forming a circle with the IMU, with the values obtained from the sensor arranged in different arrangements, arrangement1 and arrangement2.

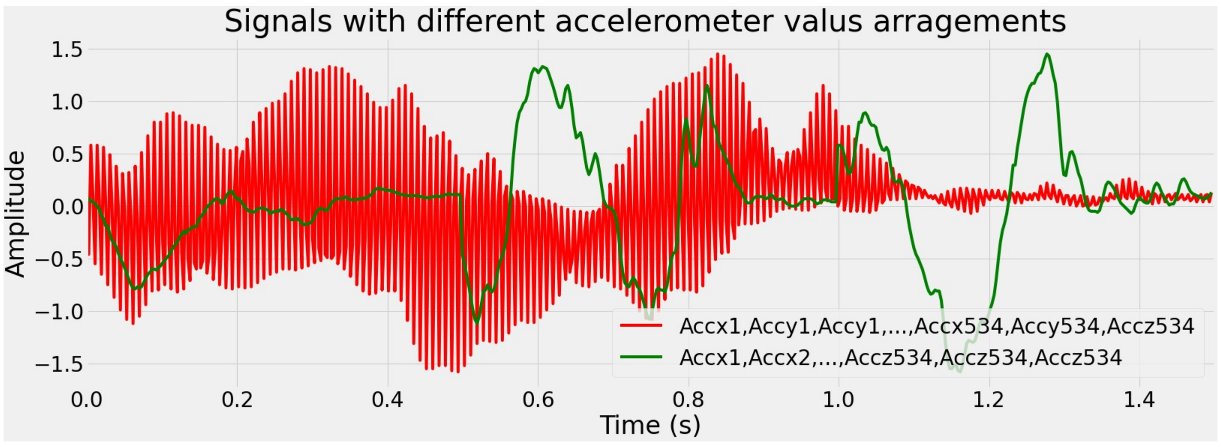


Figure 27: Accelerometer values plot with different values arrangements

Since the number of features is large, a relatively large and equal number of samples are required for each movement. For this purpose, 100 samples were taken from each movement, with 534 features, and stored in a CSV file. This created a dataset with 300 rows and 534 columns. This dataset will be used by the feature selection mechanism to create a dataset with fewer features, and the classifier will be trained on that.

4.1.2 Classification subsystem implementation

In this section of the work, the design and development of the computational intelligence software for the needs of this postgraduate thesis will be presented in detail.

The aim is to create a model that, when a movement is performed, an inertial sensor can record the accelerometer values and recognize it, as long as it belongs to the group of movements it has been trained on. If it doesn't belong, it simply ignores it. This is a case of supervised machine learning classification because our model is fed with a dataset containing both the data (features that have been extracted) and the corresponding classes. We want to train a model that can learn to map the features (also known as explanatory variables or features) to the target, and in this case, the correct class. The problem is a multi-class classification case because each new input element processed by the system must correspond to one of the existing categories (classes).

During training, we want the model to learn the relationship between the features and the classes we want to separate, so we provide both the features and the answer (class). Then, to test how well the model has learned, it is evaluated on a test set where it has never seen the answers. There, the final evaluation will take place, and the extraction of the final model and its integration into the microcontroller for subsequent predictions, as they are called, will be carried out.

The process of designing and developing a machine learning model is a complex one. Each individual stage is closely connected to the others, and failure in one leads to the overall failure of the model. In the following Figure 28, the overall process we will follow and describe in order to create the model is presented.

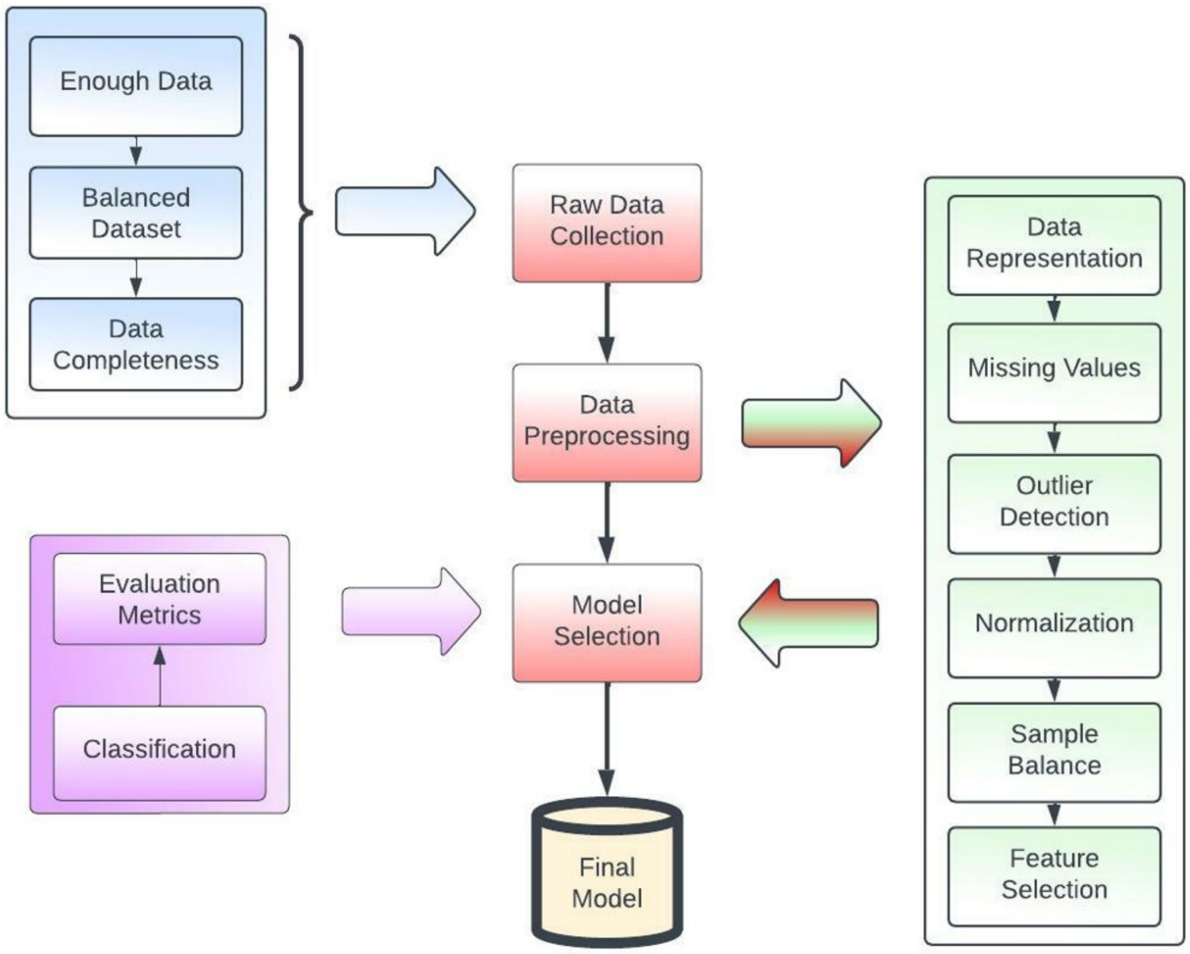


Figure 28: Machine learning model creation map [21].

4.1.2.1 Raw Data Collection

As analyzed in the relevant section, feature extraction subsystem and data-set creation, the feature extraction subsystem is the one that will give us for each of the movements of our interest a representation in the time domain. In each execution, the subsystem will give us a set of 534 features for an input signal and a 535th one, which will be the class to which the specific movement belongs, something necessary since it is supervised machine learning. The classes are defined as [Class 0: Circle, Class 1: M, Class 2: Triangle]. This set of 535 features is represented in the system as a vector and will be processed in this form. For each movement, the extraction process is repeated with different approaches to the movement for better generalization of the model to be generated. That is, for example, each triangle is formed in every possible way (from left to right, the reverse, etc.), varying the formation speed, etc. A set of 100 records (vectors) is created for each movement that the system wants to recognize.

The dataset has 535 columns and 100 rows per movement, and this size ensures the statistical adequacy of the sample. The equal presence of vectors from each class ensures that the sample is balanced (Figure 29). Since the sample was created by us, we are sure that it has no missing values, and therefore its completeness is given. By ensuring these basic parameters (statistical adequacy, balance, and completeness), we are confident that the model to be created will have good generalization.

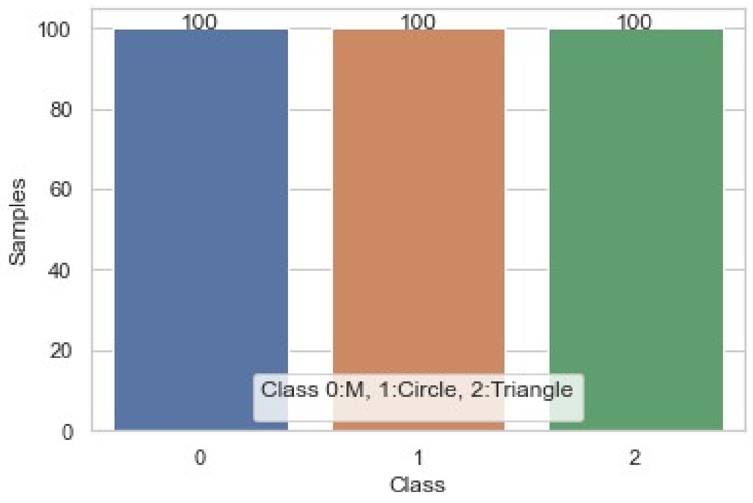


Figure 29: Dataset Balance

4.1.2.2 Data preprocessing

Data preprocessing is an integral step in machine learning, as the quality of the data and the useful information that can be derived from it directly affect our model's ability to learn. Therefore, it is extremely important to preprocess the data before incorporating it into the model.

Principal Component Analysis (PCA) is a feature reduction technique for high-dimensional datasets that is often used in machine learning. PCA is used to identify patterns within the dataset and to reduce their number while maintaining all the contained information. It is a technique that detects the so-called principal components, which describe the variation of the dataset. With their help, we can project the many dimensions of the dataset into fewer, usually 2 or 3. This technique is used for data visualization or even for feature extraction.

In the following Figure 30, we see the PCA representation in two dimensions of our dataset. We observe that our data does not seem to be linearly separable in the two dimensions. At first glance, we estimate that an SVM classifier with an RBF kernel or polynomial would perform well.

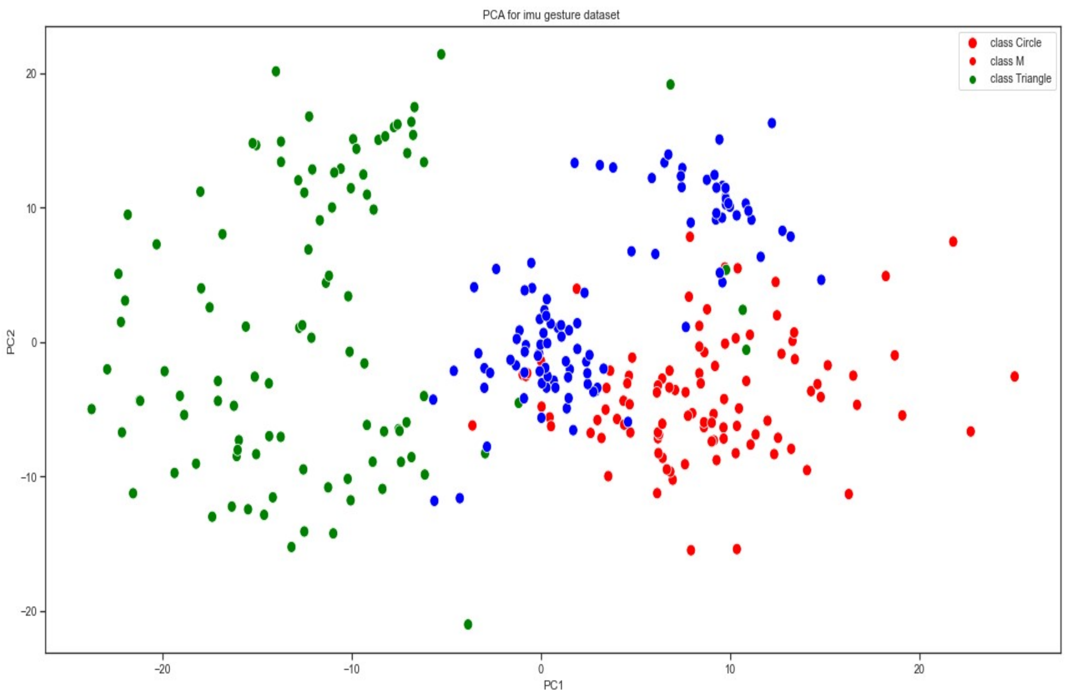


Figure 30: PCA 2-Dimentions of the initial dataset

We can see a representation of the data using the same PCA method but in 3 dimensions in the following Figure 31, from which it becomes evident that a hyperplane could be a good separator for our data.

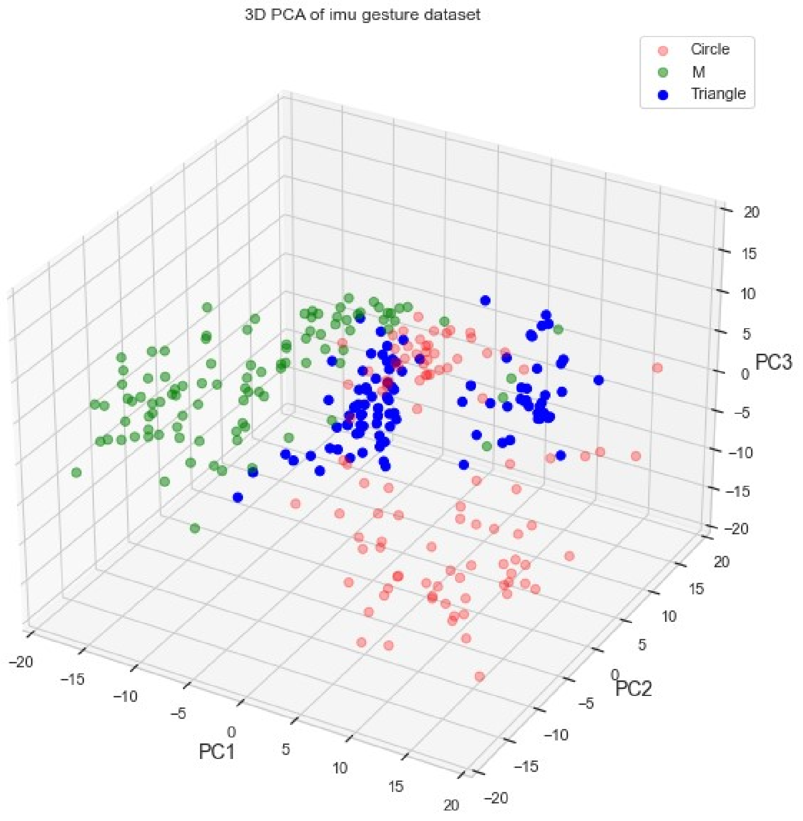


Figure 31: PCA 3-Dimentions of the initial dataset

4.1.2.3 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a process by which we have graphical representation and calculation of significant statistical elements that help in the investigation of data. The purpose is to find anomalies, patterns, trends, or relationships among them. The findings can be interesting in them (for example, finding a correlation between two variables) or can be used to inform modeling decisions, such as which features to use [53].

At this stage, in the data (since as we know they had no missing values), detection was carried out and no extreme values were found that could be due either to poor sampling or to errors. Also, all data were in numerical form and therefore no conversion of categorical to numerical was needed. To quantify the correlations between the features (variables) and output, the Pearson correlation coefficient can be calculated. This is a measure of the strength and direction of a linear relationship between two variables: a correlation value of -1 means that the two variables are perfectly negatively linearly correlated, and a value of +1 means that the two variables are perfectly positively linearly correlated. Figure 32 shows different values of the correlation coefficient and how they appear graphically.

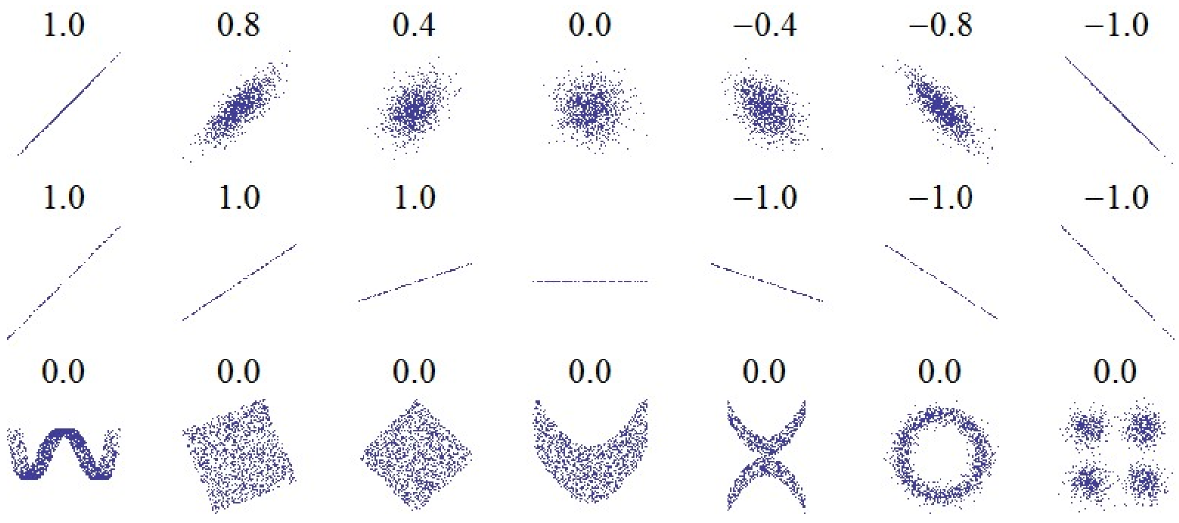


Figure 32: Pair plotting various Pearson correlation coefficients

Since the dataset is large, we will present a small sample of the correlation between the output and 10 features that include some strong features among them. The positively strong correlation of features x350 and x353 with the output is shown with a green outline. Note that a large negative correlation means that as one quantity increases, the other decreases intensely. A positive correlation is defined in a similar way.

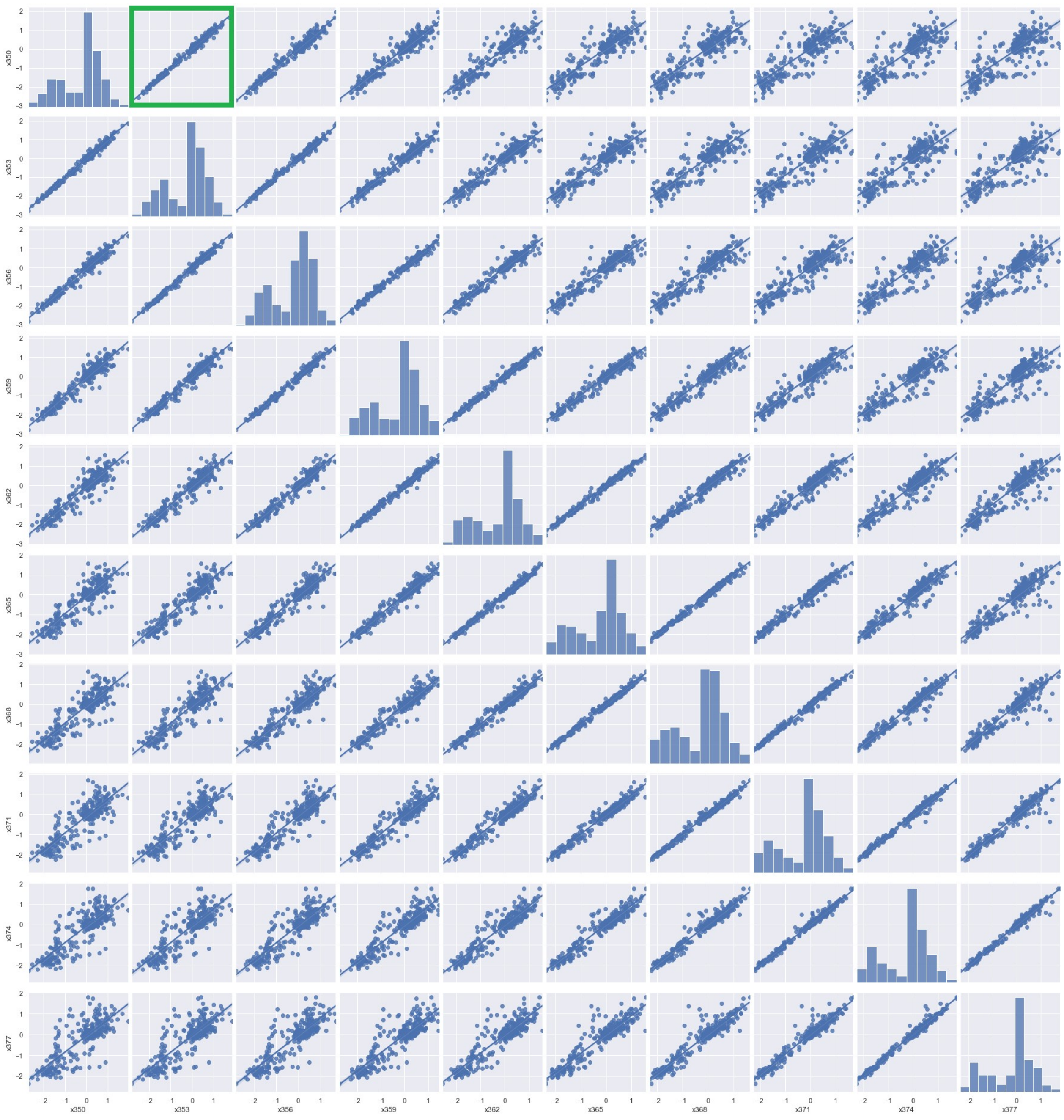


Figure 33: Pearson correlation coefficient between features (by two).

In a much more visual way, we can see the negative and positive correlations using a heat map, which shows us the same information with color gradations and allows us to study a larger portion of the dataset.

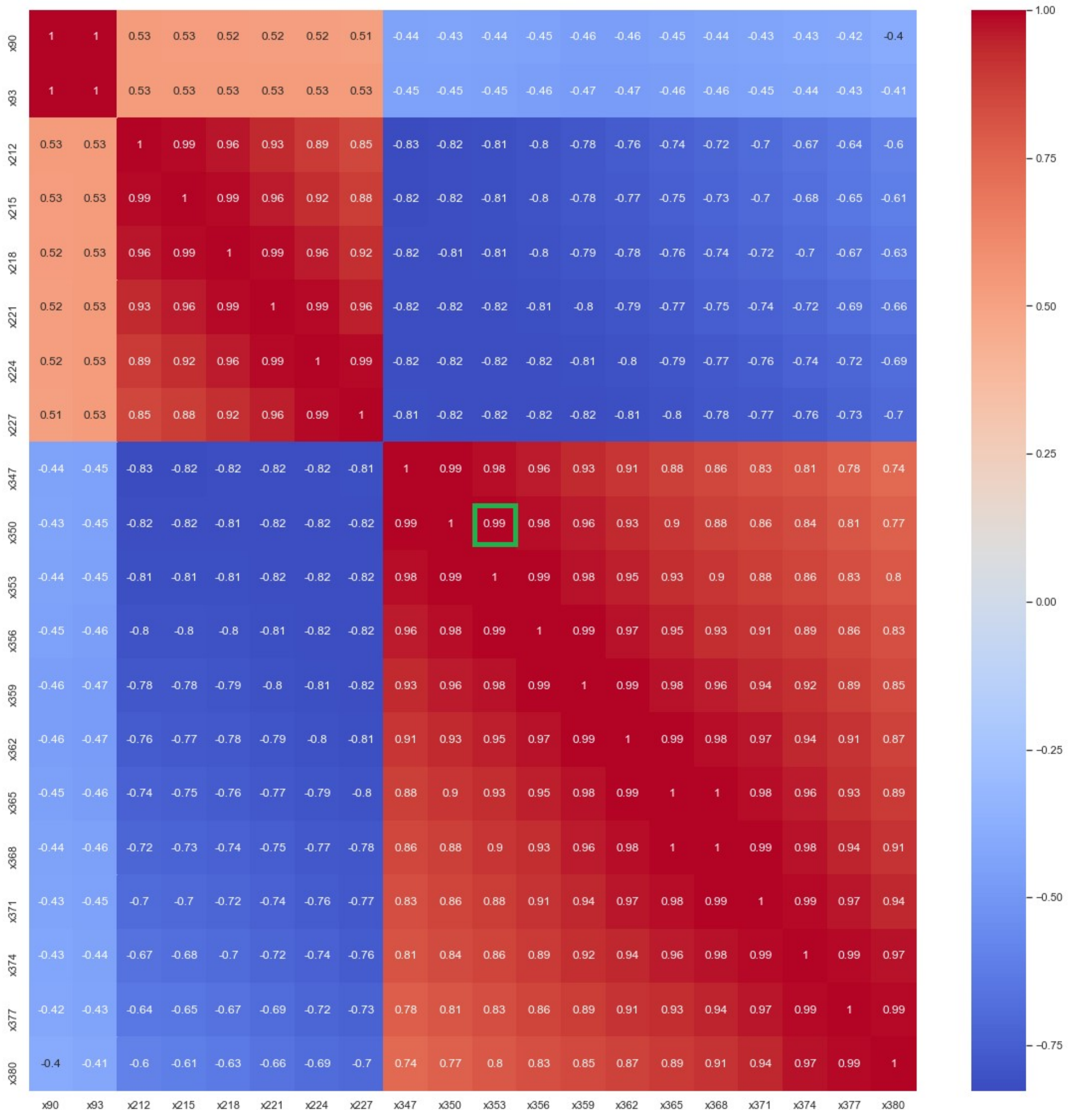


Figure 34: Pearson correlation coefficient between features as heatmap.

In Figure 33 and Figure 34, we see the positive correlation between features X350 and X353 highlighted in green.

It is important to mention that, the heat-map is a graphical representation of the correlation matrix of the dataset's features. The correlation coefficient ranges from -1 to 1 and how strongly two variables are related to each other, with -1 indicating a strong negative correlation and 1 indicating a strong positive correlation. The heat-map uses colors to indicate the strength of the correlation, a blue-white-red color map with cool colors (blues) representing negative correlation values and warm colors (reds) representing positive correlation values.

In this particular case, the heat-map shows the correlation between the features that were selected by the SelectKBest function where in this graph k=20. A technique to select k features which have the highest correlation with the target. So the heat map shows the correlation between the top k features.

It is important to take into account the following points:

- When two features have a strong positive correlation, means that they have a strong relationship, and they might contain similar information. So, one of these features could be removed to avoid unstable results in some models like linear regression. We see that we have a lot of them so we have make more actions to delete these features in order to avoid over-fitting.
- The darker the color is, the stronger the correlation is. The darker blue colors represent a strong negative correlation, and the darker red colors represent a strong positive correlation

From the insights we gain through the graphs in the EDA section, we can conclude that we are likely to achieve a classification with very high results, and that a highly effective model will be created.

4.1.2.4 Feature Engineering and Selection

We saw from the above study that we have features that are highly correlated with each other. A procedure was created which removed collinear features in the data-frame with a correlation coefficient greater to each other more than a specific threshold. If we look at the (Figure 34.B) we can clearly see that when we increase the threshold (from 96 to 100%) the accuracy (cross-validation accuracy) is increasing. We get maximum accuracy 97% if we remove the features that are related to each other more than 99% and 100% according to Pearson correlation. We can also notice that if we increase the threshold we increase the remaining dataset. So we select to remove the features that have Pearson correlation 99% and above because we get the max accuracy and the minimum remaining dataset.

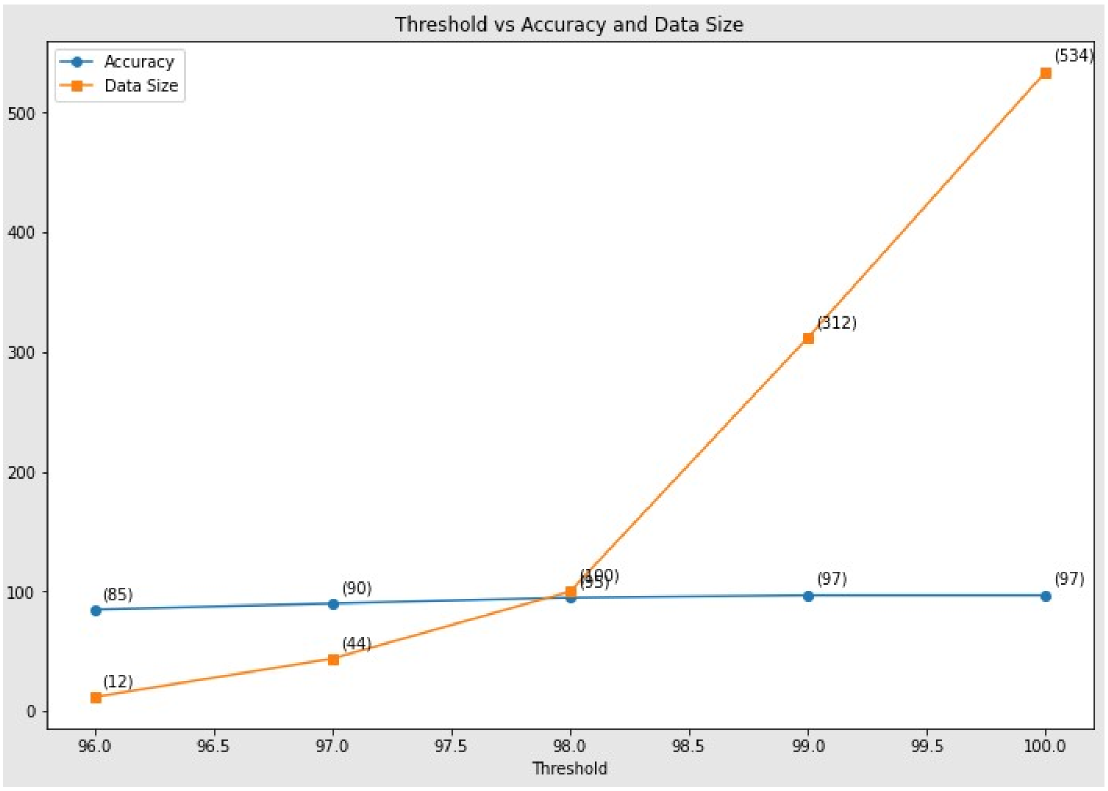


Figure 34.B: accuracy vs. Pearson correlated features removal.

With that threshold we were able to remove 222 features and the final shape of the dataset was 300 rows and 312 columns (311 features and the target). Removing collinear features helps the model to generalize and improves its interpretability. At the end of the process we have a data-frame that contains only the non-highly-collinear features.

A challenge we face is whether the number of necessary features can be further reduced in order for our model to make accurate predictions. We removed those with high correlations among themselves, and among the remaining ones, we did not have any with high correlations.

In the stage of additional feature selection, the Kbest function was used, which takes a number as a parameter. For example, Kbest=5 means that we want to take the 5 strongest features. With the parameter kbest=293, we obtained an SVM model with a cross-validation evaluation score of 97.5% (Figure 35), achieving the same performance as with 312 features.

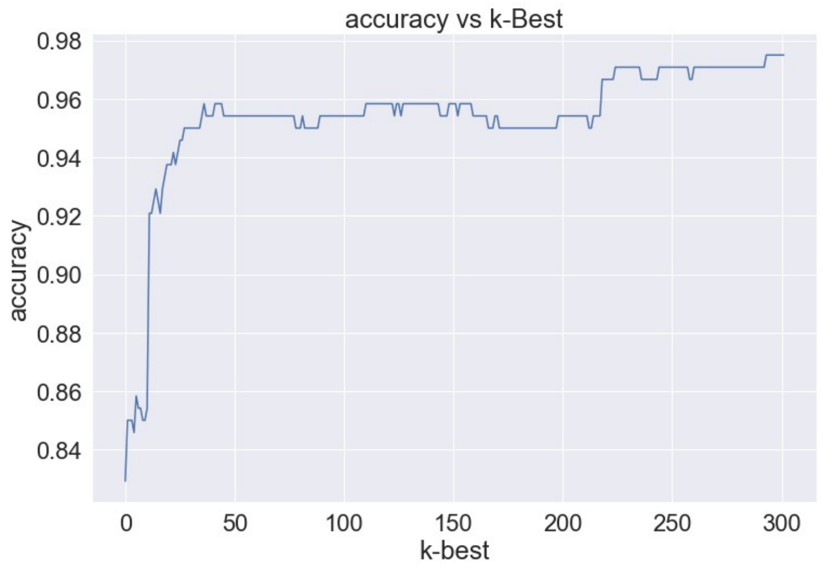


Figure 35: Accuracy vs. k-best

This feature selection approach, in which we chose a dataset without elements with very high correlation among themselves and secondly selected kbest=293, although it provides a model with very good performance, will prove in the evaluation section that it is not the best. This is one of the related trade-offs between ML and CS we mentioned. More specifically, with this feature selection process, we have good recognition, but very poor generalization in the reconstructed signals, as we will see in the relevant study. This is due to the fact that some of the elements that were removed also had a very good relationship with the output.

4.1.2.5 Train and evaluation set creation

After reducing the set of features, the train and test sets were created with an 80-20 ratio. To achieve balanced samples with equal presence, each class in the sets used the stratified Kfold process, which has been extensively discussed in the theoretical section of the present work.

4.1.2.6 Model Selection

In order to find the most suitable model, a set of algorithms was created, consisting of the most well-known and frequently used algorithms for which a presentation was made in the theoretical part. More specifically, we used:

- KNeighborsClassifier()
- AdaBoostClassifier()
- GradientBoostingClassifier()
- DecisionTreeClassifier()
- GaussianNB()
- SVC()
- LogisticRegression()

We used, parameter for Kfold k=5 and for the evaluation method, the accuracy.

Several models achieved a high cross-validation score, GaussianNB, GradientBoosting, and SVC in default settings close to 93% for SVM and 95% for GradientBoosting. The performance per algorithm is shown in Figure 37. From these, we will choose the Support Vector Machine (SVC) for the following reasons:

- It is the one that tries to find the maximum range between classes to achieve better generalization.
- It is the most resource-efficient solution for integration into the microcontroller.
- It achieved the best score.

Before we proceed, we need to discuss a very important process that must be followed when working with SVM, the so-called feature scaling. It has been mentioned that the purpose of an SVM is to maximize the distance between classes. Since this goal is based on geometric distances, an axis with a very different scale makes the model favor the direction with the larger values.

In Figure 36, we can see that before scaling the features, the SVM seeks a decision boundary so that the distance vector d_1 has the largest vertical element. This is the reason why we always need to apply feature scaling before placing an SVM.

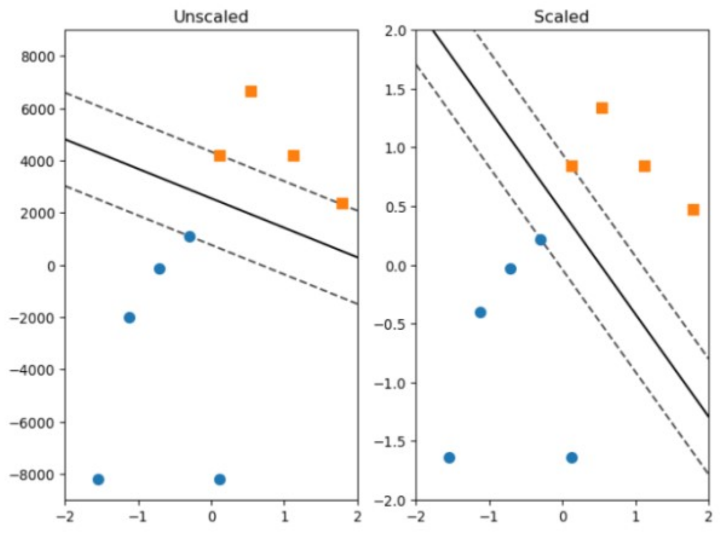


Figure 36: The importance of scaling in SVM

For the importance of scaling, W.S.Sarle mentions that the main advantage of scaling is to avoid features with a larger numerical range dominating smaller numerical ranges. Another advantage is avoiding numerical difficulties in calculations, which accompany such a phenomenon.

The values of the cores of linear SVMs as well as polynomials are the result of the inner product between feature vectors. Thus, large-sized operands in the operations of the inner product create computational problems, especially in microcontrollers with limited computational capabilities. It is recommended to linearly scale each feature in the range [-1, +1] or [0, 1]; in our work, we scaled in the range [0, 1]. Of course, the same method must be used for scaling both the training and evaluation sets.

Below is Figure 37, which shows the performance of each algorithm in accuracy, before and after scaling.

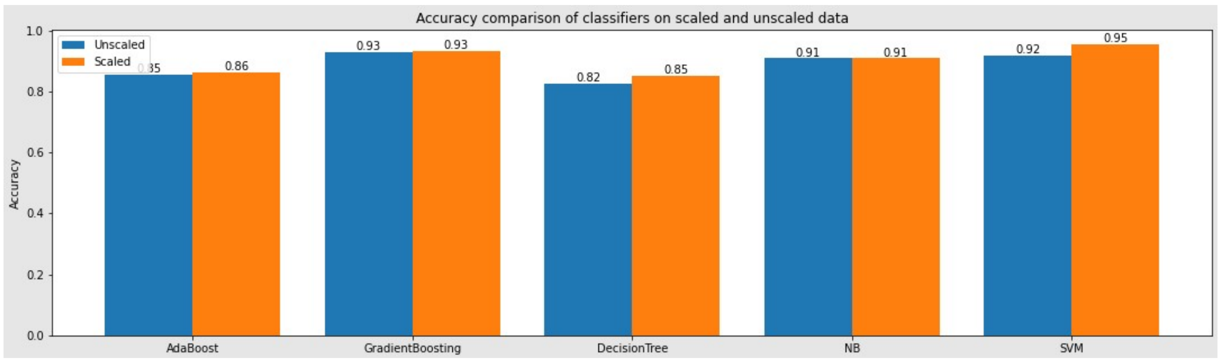


Figure 37: Classifier Cross validation score during model selection on scaled and unscaled data

4.1.2.6.1 SVM tuning hyper parameters.

Next, the optimization of the selected model followed. In machine learning, optimizing a model means finding the best set of hyper parameters for a specific problem.

First of all, we need to understand the hyper parameters of the model and the difference they have with its parameters. Hyper parameters are considered to be the parameters that have an exceptionally significant impact on the performance of the machine learning model. The settings of the hyper parameters are made by the data handler before training.

SVM has hyper parameters such as C or gamma and the type of kernel (linear, rbf, Gaussian polynomial, and others). Finding the optimal hyper parameter is a particularly demanding process. However, it can be found simply by trying all combinations and seeing which parameters work best. This method is implemented by the GridSearchCV function.

Below is Figure 38, we can see how the accuracy is shaped as the values of C change for the various SVM kernels. In this graph, we see the performance of the SVM with an rbf kernel being clearly depicted, with an optimal performance at C=2. We observe that the kernel's shape can become polynomial in some cases, but the predominant shapes are rbf. Of course, we will see that when the composition of the dataset changes with a feature selection method, the kernel's shape also changes. While a linear kernel does not create a linear separator, once redundant features are removed, we will see that it is the most prevalent.

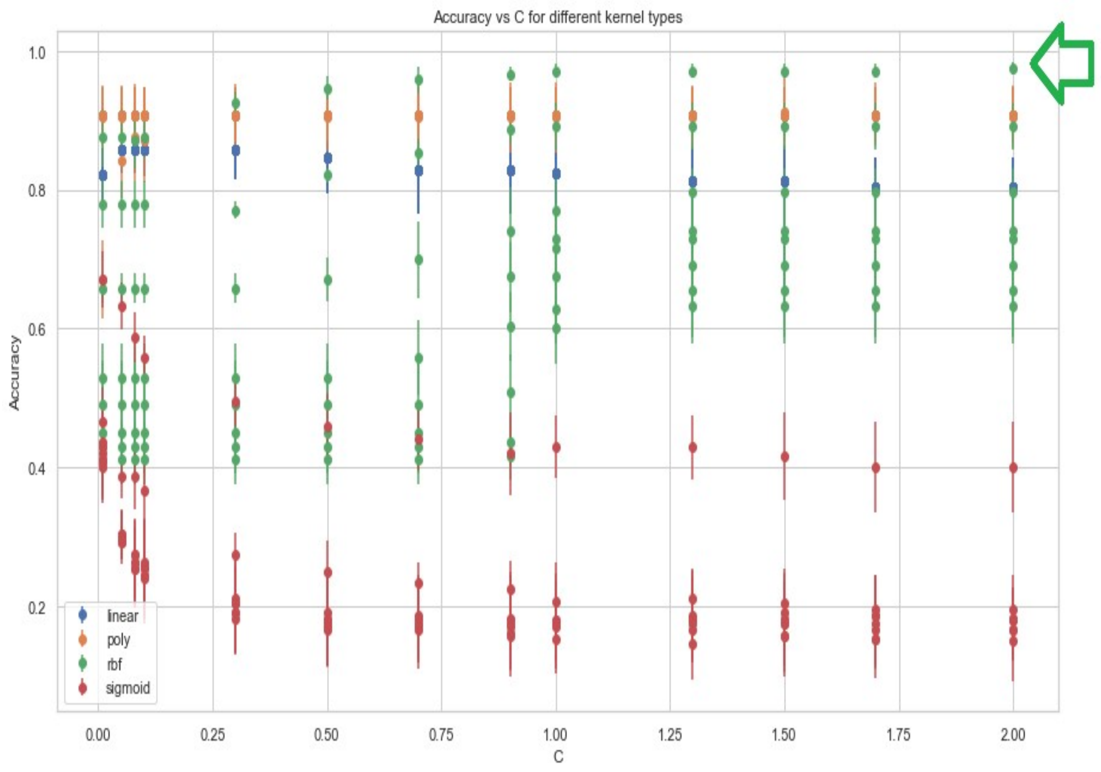


Figure 38: Plotting the SVM hyper-tuning process for c and kernel type

In the following graph, Figure 39, we can see how the accuracy is shaped as the values of gamma change for various types of SVM kernels and different values of C. It seems that the logic of the change is the same as the one described in the commentary of Figure 38.

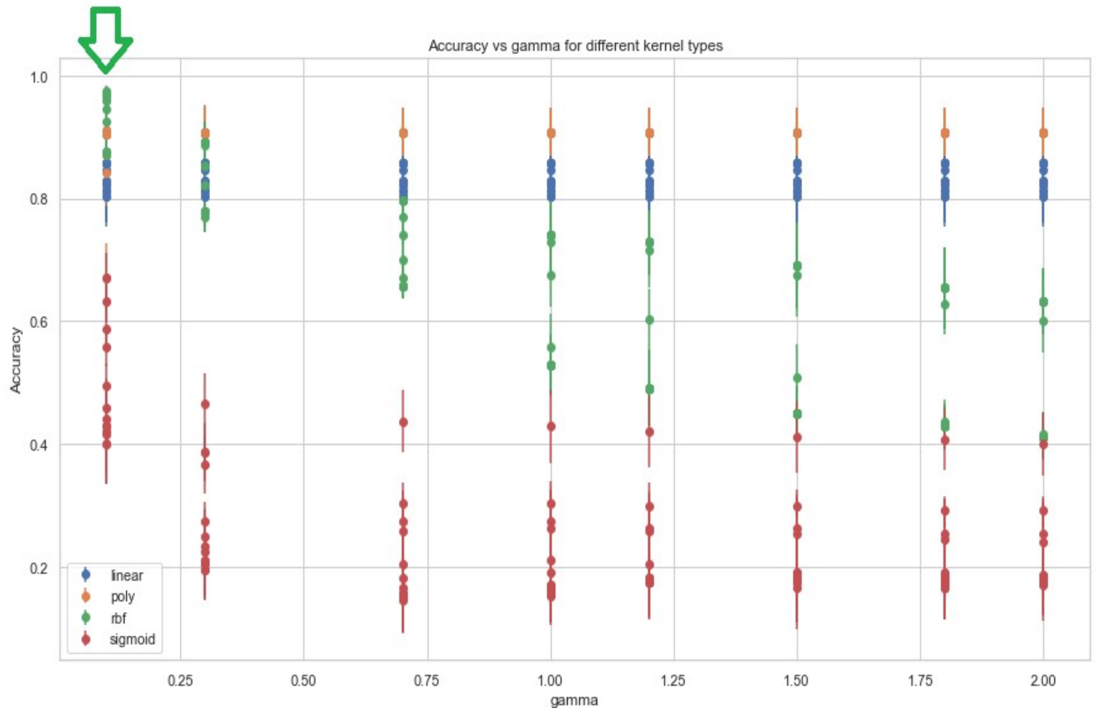


Figure 39: the SVM hyper-tuning process for gamma and kernel type

Using the GridSearchCv process and StratidiedKFold, we found that the best settings for its hyperparameters are

{'C': 2, 'gamma': 0.1, 'kernel': 'rbf'} and the accuracy using the train set is 98,3%.

We use the best model resulting from the hyper parameter tuning to make predictions on the test set. Remember that the model has never seen the test set we created earlier, so this performance should be a good indicator of how the model would behave in the real world. Additionally, we have an extra evaluation dataset that does not participate in the training process and internal evaluation, which we will use to evaluate the model's generalization and overall predictive ability.

4.2 Export trained model to C.

With the completion of training and evaluation, we now have in our hands a machine learning model capable of performing prediction functions. The goal is to export this model to C to be integrated into the microcontroller.

4.2.1 Existing alternative solutions

Before starting the effort, research was conducted to find what exists, although from the beginning of the elaboration of this thesis, until today, the landscape has dramatically changed due to the huge trend of placing machine learning on the edge and extreme edge.

4.2.1.1 TensorFlow Lite for micro-controllers

TensorFlow Lite for microcontrollers is designed for devices with generally low memory. General minimum requirements for an exported model to run are 16 KB, which may be small but is quite large compared to the 2K of Arduino Uno, especially if we want such a microcontroller on the leaf node. TensorFlow Lite can support many basic models.

TensorFlow Lite for microcontrollers is written in C++ 11 and requires a 32-bit platform. It has been extensively tested with many processors based on the Arm Cortex-M Series architecture and has been ported to other architectures, including the ESP32. The framework is available as an Arduino library.

The following systems are supported:

- Arduino Nano 33 BLE Sense
- SparkFun Edge
- STM32F746 Discovery kit
- Adafruit EdgeBadge
- Adafruit TensorFlow Lite for micro-controllers Kit
- Adafruit Circuit Playground Bluefruit
- Espressif ESP32-DevKitC
- Espressif ESP-EYE

4.2.1.2 Sk-Learn Porter

Transpiles trained classification models created using scikit-learn into C, Java, JavaScript, and others. It is recommended for a variety of embedded systems where performance is of greater importance. Using this package, the following machine learning algorithms can be exported to C::

- SVM.SVC
- SVM.NuSVC
- SVM.LinearSVC
- tree.DecisionTreeClassifier

This transpiler seemed like it could be the tool for exporting the model, but it has a significant disadvantage: the code it produces is not optimized for microcontrollers. To give a small example, let's mention that for a small dataset of 2 classes with 569 samples and 30 features each, the data are real, positive. A simple SVM classifier required a 57x30 (doubles) matrix, which is 6840 bytes, just for the support vectors.

4.2.1.3 Edge Impulse

Edge Impulse is a cutting-edge platform for developing and deploying machine learning models on edge devices. The platform offers a user-friendly interface, making it accessible for developers with limited machine learning experience. With Edge Impulse, developers can create, train, and deploy models on edge devices in real-time, without the need to transmit data to the cloud. This allows for low latency and high security, making it ideal for applications such as industrial automation, smart buildings, and IoT.

One of the main drawbacks of Edge Impulse is that it currently only supports a limited set of microcontroller platforms. Additionally, the performance of models deployed on edge devices may not be as high as those deployed on more powerful cloud-based servers. Furthermore, the storage and computational resources of edge devices are limited, so the complexity and size of the models that can be deployed on them are also limited. Finally, while Edge Impulse allows for easy deployment of models, it may not offer as many tools for monitoring and maintaining deployed models as some other platforms.

Edge impulse, as we said, does not support a large number of micro-controllers and especially boards with limited capabilities such as arduino mega. As a result, we have a limitation on choosing a board as leaf node. With the method we propose, the resources required are the minimum possible and dramatically less than the models exported from the specific platform.

4.2.1.4 Emlearn

Emlearn produces code that has been optimized for microcontrollers. It also has some very interesting features:

- Portability with C99 code
- No need for Libc
- No dynamic allocations
- Support for integer/fixed-point arithmetic

Supported machine learning models are :

- Decision Trees
- Random Forest
- Naïve Gaussian Bayes
- Full connected neural networks.

The problem in each case was that it does not support SVM, which, as we have documented, is the best solution for our problem. As a result, it cannot be a solution for our problem.

4.2.2 The developed approach.

In a previous section of the background, the mathematics behind SVMs were presented, and based on this, we will proceed to implement the prediction system that relies on our trained model, which is a linear SVM classifier. For a linear kernel, like the one we have, the prediction equation for a new input data (x) is calculated as follows:

$$f(x) = B(0) + \sum_{i=1}^{n} a_i * (x \bullet x_i)$$

Equation (13). Class prediction function for input data x with linear kernel SVM

Where:

- The B(0) (bias) and the a_i (alpha coefficient) of each support vector are provided by the scikit-learn library when we have trained our model.

- (x*xi) is the inner product of the input vector and each of the i support vectors, which can also be extracted from the trained model.
- n is the number of support vectors.

Below, the algorithm is presented, expressed in natural language with steps, for a more complete and clearer presentation.

4.2.3 Prediction system Algorithm

After training the model, the process of extracting the necessary parameters of the model begins, which will allow us to recreate it in the Arduino IDE. To complete the process, we will need:

- The list of support vectors per class. In our case, we will get a 1x3 list since we have 3 classes [x y z]. Each number x, y, z indicates the number of support vectors per class.
- The list of support vectors for our model. It will be of dimensions NxKbest. More specifically, it will have N rows (N=x+y+z), the total number of support vectors. The columns will be the number of features determined by the feature reduction process in the machine learning part (kbest).
- The list of kernel weighting coefficients, dual coefficients (dual problem solution). With their help, we will define the decision function, that is, our model. The dimension of the list is MxN, where M = number of classes-1 and N is the number of support vectors.
- The list of intercepts of the model, which essentially is the constant part of the line (y=ax+intercepts).

Below is a summary of the algorithm of the model extraction method and its recreation in another environment beyond the development system (scikit-learn). The representation of the algorithm is in natural language with steps so that it can be developed in any programming environment.

Step 1. From the trained model, the list of support vectors, dual coefficients, number of SVs, and bias are extracted.

Step 2. For an incoming input signal X, the inner product with each of the support vectors is calculated. In this way, we create a set of kernels, as they are called, with the same number as the support vectors. Ki = X dot SVi, Ki is the ith kernel.

Step 3. For each pair of classes classi and classj, we create a decision function which is the linear combination of the "kernels" with the dual coefficients (weightening of kernels) as follows: decision functionij = interceptij + kernels of classi * dual coefficients of classi (vs classj) + kernels of classj * dual coefficients of classj (vs classi)

Step 4. We apply a voting mechanism using the decision functions calculated in step 3 as follows:

For each pair of classes y^1, y^2, if the decision is > 0, then we add a vote to the class y^1; otherwise, we add a vote to the class y^2. In the end, the class that has garnered the most votes is the class to which the input element we studied is predicted to belong.

The implementation of the above algorithm was done in C++, and the model is imported into the code in the form of an accompanying library.

4.2.4 Embed to microcontroller.

The integration into the microcontroller is a process that is demanding in terms of minimizing the computational requirements due to the limitations imposed by the microcontroller itself. In reality, once the process of creating the model (offline) is completed, we then need to implement the algorithm that implements the SVM, as described in detail in the previous section.

Throughout the study of this thesis, we had to create different implementations of the algorithm within the framework of experiments and tests, which would result in the need to implement the above algorithm many times. Thus, a system (porter) was developed, with the help of which the creation of the model was simply an automated process of extending the training of the model. The important code documented, that implements the SVM model for the microcontroller is contained in the appendix in the porter code section.

The process generally proceeds as follows: After training the model, we extract the basic parameters of the SVM that interest us, namely the support vectors, dual coefficients, and bias terms. Using jinja2, a template in Python was created, with the help of which the code is generated in C++. Jinja implements loops to iteratively use all the support vectors and dual coefficients and create code snippets that implement the decision function for the SVM. The exported code is stored in a Model.h file, which is then integrated as a library into the prediction execution program. The jinja2 code that implements the above is in the appendices in the jinja2 porter code section.

4.3 CS system

Compressed sensing is a technique that consists of two subsystems, the compression subsystem and the reconstruction subsystem. The first is responsible for taking the measurements of the IMU signal and performing the compressive sampling of it. In this subsystem, the leaf node collects the measurements of the IMU signal, applies a sub-sampling array to the signal and sends it to the sink node.

The reconstruction subsystem is responsible for recovering the original signal from the compressed measurements. In this subsystem, the sink node uses an appropriate reconstruction algorithm, to reconstruct the original signal from the compressed measurements. The sink node also applies normalization to the original and to the reconstructed signal and calculates the MSE to evaluate the accuracy of the reconstruction. It's important to note that the performance of the compressed sensing system depends on the design of both subsystems

4.3.1 The compression Subsystem.

4.3.1.1 Signal creation.

The compression subsystem was implemented and operates on the Arduino Nano. Using the selected sampling frequency of 119Hz, as previously mentioned, 178 points were chosen for each of the 3 axes, totaling 534 points. In this way, we created the vector of the original signal. Each point we receive at a specific time t consists of 3 accelerometer values, one for each x, y, z axis. A question is, in what arrangement will the values be placed within the signal? A possible arrangement (arrangement 1) would be to place the points as they are received in the x vector, that is, accx1, accy1, accz1, ..., accx178, accy178, accz178. Another arrangement (arrangement 2) would be to have the data for each axis separately, first for x, then for y, and then for z. Below in Figure 40, a representation of these two ways is provided to give a better understanding.

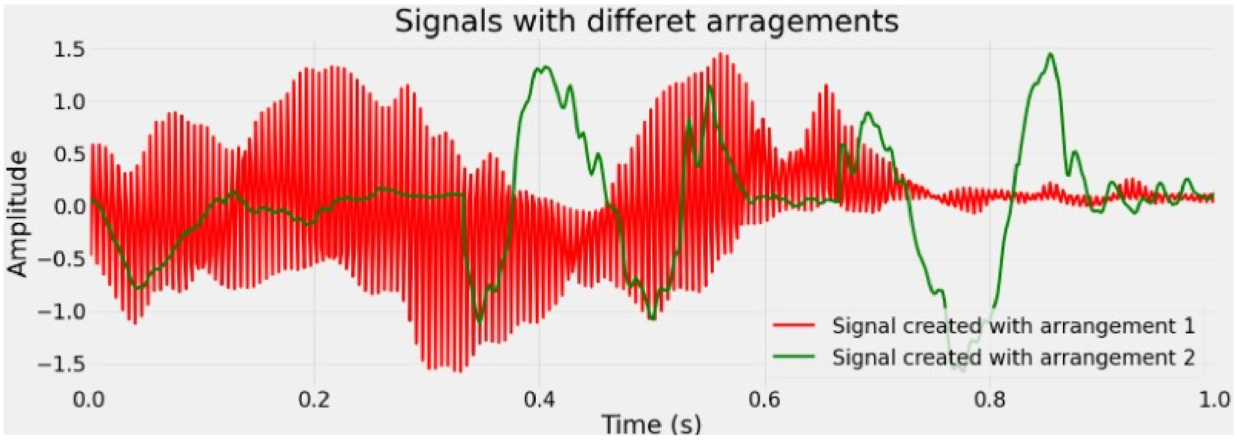


Figure 40: Plotting capture signals with different value arrangements

In order to determine which arrangement was the best, we did the following. We performed hyper-tuning on the Lasso solver and studied which arrangement had the lowest reconstruction error during the hyper-tuning process. The results are presented in Figure 41:

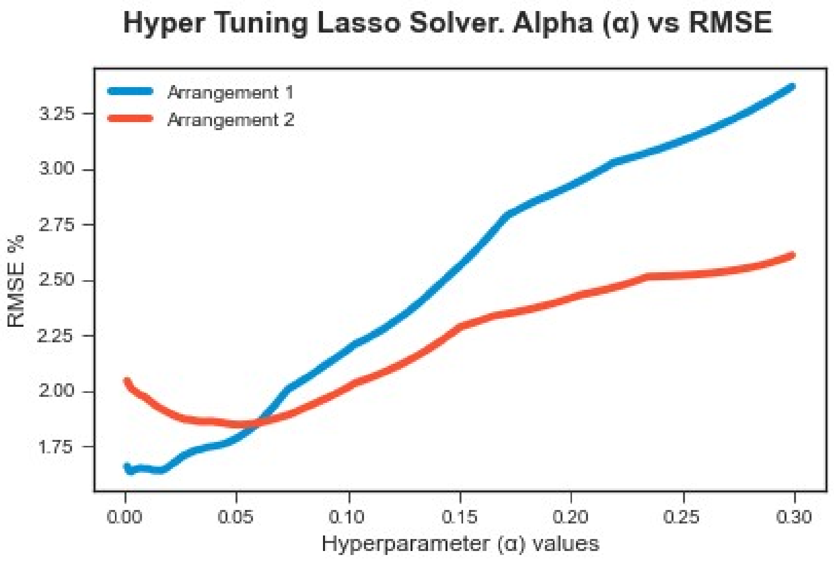


Figure 41: Lasso reconstruction MSE arrangement1 vs arrangement2

In Figure 41, we see that low values of the hyper parameter alpha are crucial for both cases (with the significance we analyzed in the corresponding part of the solver). We observe that, in general, the reconstruction error for arrangement 1 at its optimal point (approximately 1.5%) is lower than that of arrangement 2 at its lowest point (approximately 1.8%). The entire process of our study showed that arrangement 1, both in terms of machine learning and signal reconstruction, yielded better results, which is why it was adopted.

4.3.1.2 Selecting Sampling Matrix.

One of the key elements we focus on in the compression section of this report is the measurement matrix A. The goal is to acquire a compressed version of a signal x of high dimension, through a small number of linear measurements y = Ax. The measurement matrix A plays a crucial role in this process as it determines the properties of the measurements and the ability to reconstruct the signal x from y. It is important to remember that we ar at the extreme edge so the leaf node is a micro-controller which means, limited resources and computation power.

A common choice for the measurement matrix A is a random matrix, such as a Gaussian or Bernoulli matrix. These types of matrices have the property of being incoherent with the signal's representation in a certain basis [54]. This incoherence as told before, leads to stable and efficient recovery of the signal x from y using convex optimization algorithms.

In order to randomly sample our initial signal X according to the compressed sensing method on the Arduino Nano 33 BLE Sense random subsample matrix will be our choice. There are several reasons why this is the best choice among the various types of random measurement matrices, such as Bernoulli, sparse, and Gaussian matrices:

- A random sub-sampling matrix is a binary matrix with only one non-zero element per row, it's constructed by randomly selecting indices and setting the corresponding values to 1. This type of matrix ensures that each element of the signal X is measured only once. That means that, every measurement in the compressed signal is a direct measurement of the original signal and is not a linear combination of various elements of it, which has a positive impact on the reconstruction quality.

- In terms of computational cost, dealing with Gaussian matrices can be computationally expensive and may not be feasible for use on the Arduino Nano 33 BLE Sense due to its limited processing capabilities. On the other hand, random sub-sampling matrices are computationally efficient and can be implemented in a relatively short amount of time on the micro-controller. This makes them well suited for use in real-time applications such as gesture classification, where speed and processing efficiency are important considerations.
- In terms of memory constraints, random subsample matrices have a small memory footprint and can be stored efficiently in the limited memory of the Arduino Nano 33 BLE Sense. This is in contrast to sparse or Gaussian matrices, which can require a large amount of memory to store and may not fit within the constraints of the micro-controller.
- A Bernoulli matrix is a matrix whose entries are independently and randomly chosen from the set {-1,1} or {0,1} with equal probability. This type of matrix does not guarantee that each element of the signal X is measured only once, which could lead to increased noise in the measurements and decreased ability to accurately reconstruct the signal. In
- Most important, random subsample matrices have low coherence with the underlying transformation basis, such as the DCT, which is important for achieving good reconstruction quality.

The combination of computational efficiency, low memory footprint, low noise presence, and low coherence with the transformation basis make random subsample arrays the best choice for compressive sensing applications on the Arduino Nano 33 BLE Sense.

In the following graph, we can see our initial signal and the selected points at 20% of the original signal, as well as how the sampling of a set of points is performed with the help of a sampling matrix.

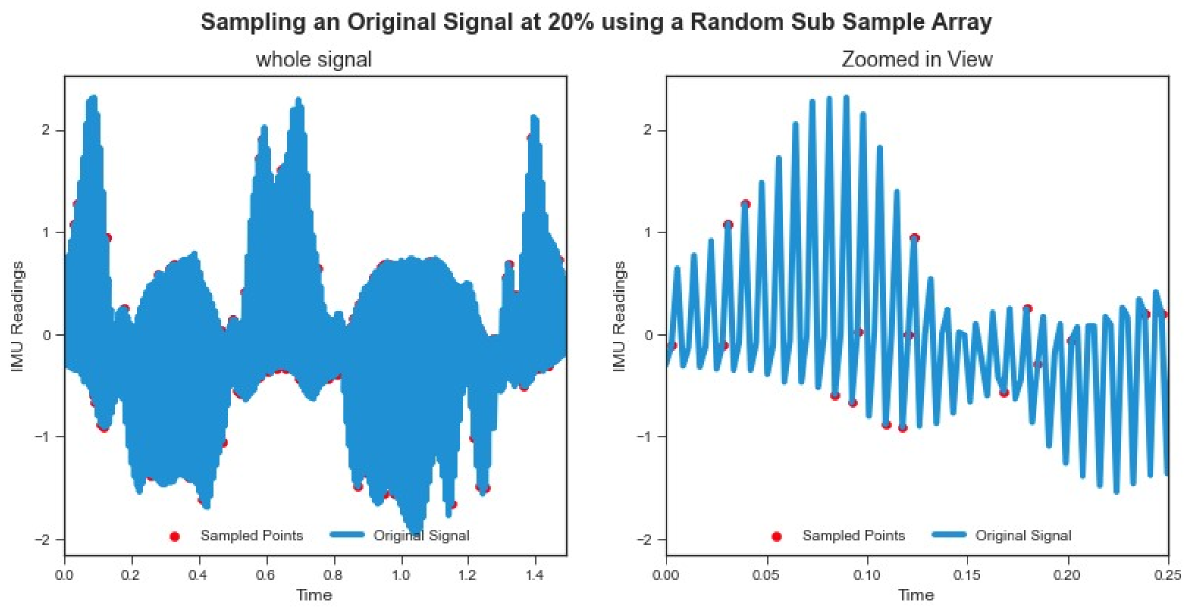


Figure 42: Using subsample array to measure 20% of initial signal

4.3.2 The Reconstruction Subsystem.

The reconstruction system is the one that, from the m measurements of the initial signal, will attempt to reconstruct it. One of the essential components belonging to the reconstruction section is the representation basis of the signal.

4.3.2.1 Selecting representation basis.

The choice of the representation basis is extremely critical for compressed sensing because it has a direct impact on the sparsity and compressibility of the signal. The representation basis, also known as the dictionary, is what will help us represent the signal we captured in a sparser form.

For an IMU signal that records human gestures, two representation bases that are commonly used and have proven effective are the Discrete Cosine Transform (DCT) and the Discrete Wavelet Transform (DWT). Both the DCT and the DWT provide a sparse representation of the IMU signal.

To see which the best representation basis in our case was, we studied the reconstruction of a signal we received from the IMU in both wavelet and DCT representation bases. The representation of the reconstruction in relation to the original signal is shown in the following Figure 42B.

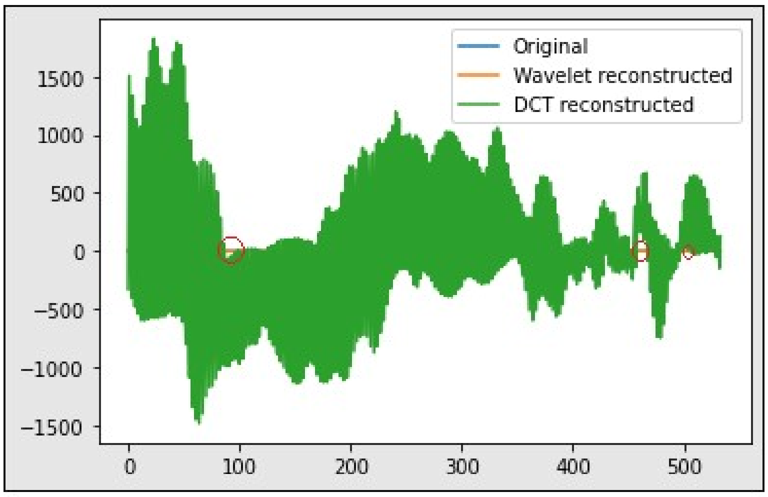


Figure 42.B: Lasso reconstruction MSE arrangement1 vs arrangement2

We observe that the reconstruction in general from both bases is very good. The reconstruction from DCT is almost perfect, while relatively greater inaccuracies (red circles) are presented in the reconstruction from DWT.

To quantify the visualized information of the previous graph, we found the MSE (Mean Squared Error) of the reconstruction of the two representations. The DCT showed a small difference in the reconstruction error (MSE) of the original signal. The output of the control program showed:

DCT representation basis is better for this IMU signal with MSE: 2.9192736096910596e-31 Wavelet representation basis is better for this IMU signal with MSE: 9.791785962405056e-31.

Another quality metric we used to compare the two bases is the PSNR (Peak Signal-to-Noise Ratio). It is a measure of the quality of a reconstructed to the original signal. It is used to evaluate the performance of signal compression algorithms. PSNR is expressed in decibels (dB) and the higher the value, the better the quality of the reconstructed signal. For our comparison the results was

DCT PSNR: 366.72254948506634

Wavelet PSNR: 365.2026569122738

The very small (but existent) differences in the two comparisons, shows us that in general both bases are able to represent our signal but DCT is a better choice. Of course, if we had high frequencies in our signal, then the small difference would not be as important as the fact that wavelets handle these types of signals better. In the continuation of the work, DCT will be used as a representation basis.

It should be noted that wavelets have a hyper parameter called wavelet_type, which has different behavior depending on the signal. To be fair in the comparison, we had to take the best DWT (Discrete Wavelet Transform) for our signal. We weighed this parameter in order to have the optimal choice of wavelet type that would participate in the comparison with the DCT (Discrete Cosine Transform). The results of the comparison are shown in the following Figure 43, which makes the use of the db2 wavelet type unquestionable.

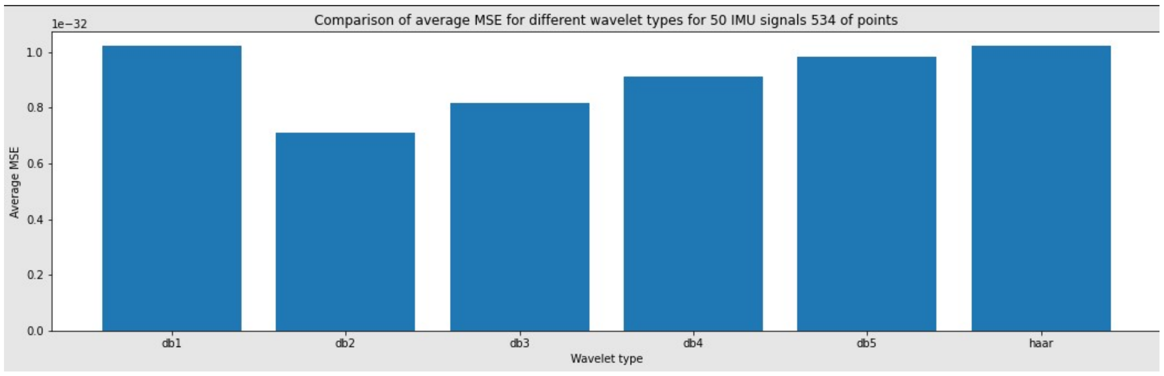


Figure 43: comparing various wavelet types reconstruction MSE

4.3.2.2 Selecting Solver

The most important role in the reconstruction part is played by the so-called solver algorithms. These algorithms take the compressed signal y, the subsample array A, and the transformation basis psi (as we proved in our case, DCT) and solve the equation y=A*psi*S for s. These algorithms search for and find the sparsest s that serves this equation, and then, with inverse transformation from x=psi*S, we obtain the reconstructed signal x. In the DCT transformation basis we chose, the reconstruction from s is done with idct(s). There are many libraries that implement this conversion; in this work, we used scipy.

We should emphasize that we also have an evaluation dataset, which consists of 10 signals from each movement. Each of these signals will be compressed and reconstructed, and the most suitable algorithm will be selected based on the MSE criterion. The methodology followed to select the appropriate solver will be presented below:

- The following representative algorithms were selected: CoSaMP, LASSO, and CVXpy. These algorithms are often used as solvers for CS problems, including the reconstruction of IMU signals. Additionally, they can handle noise and other forms of interference that may be present in the signal measurements.
- The same sampling matrix will be chosen to take 20% of the original signal as a compressed signal to make the results comparable.
- For each algorithm, after performing hyperparameter tuning where applicable, we will find the average minimum MSE for the entire evaluation dataset. LASSO will be the solver that we will use.

4.3.2.2.1 Cosamp

The CoSaMP, as a greedy algorithm, takes the desired sparsity level of the signal as a hyper parameter. That is, we define the sparsity we want the returned s to have. In Figure 44, the MSE for sparsity levels from 1 to 534 is plotted. This information will help us identify a range of values for the sparsity level that we will define, in which the algorithm exhibits good behavior.

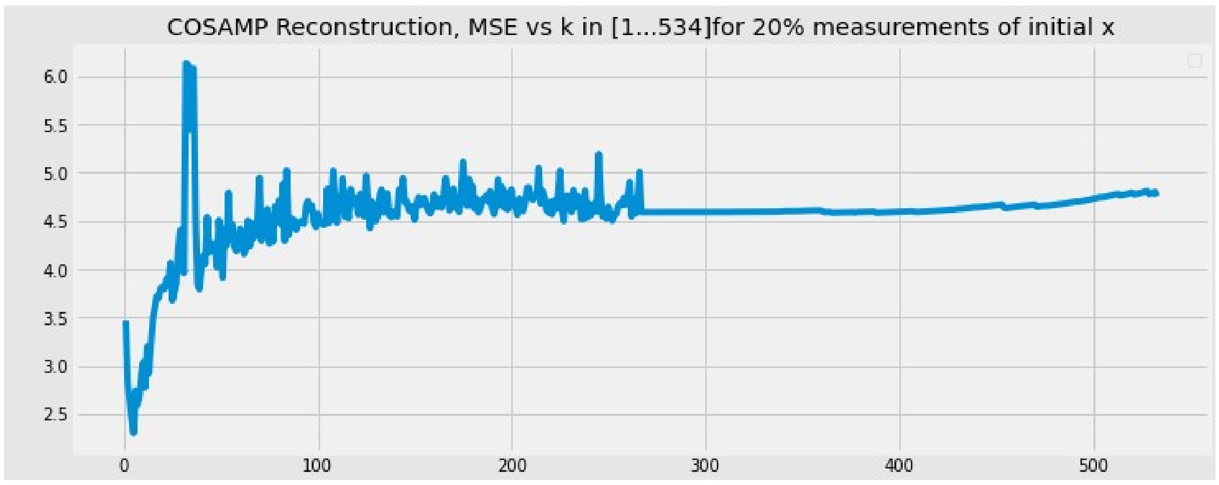


Figure 44: Plotting COSAMP reconstruction MSEfor various sparisty levels

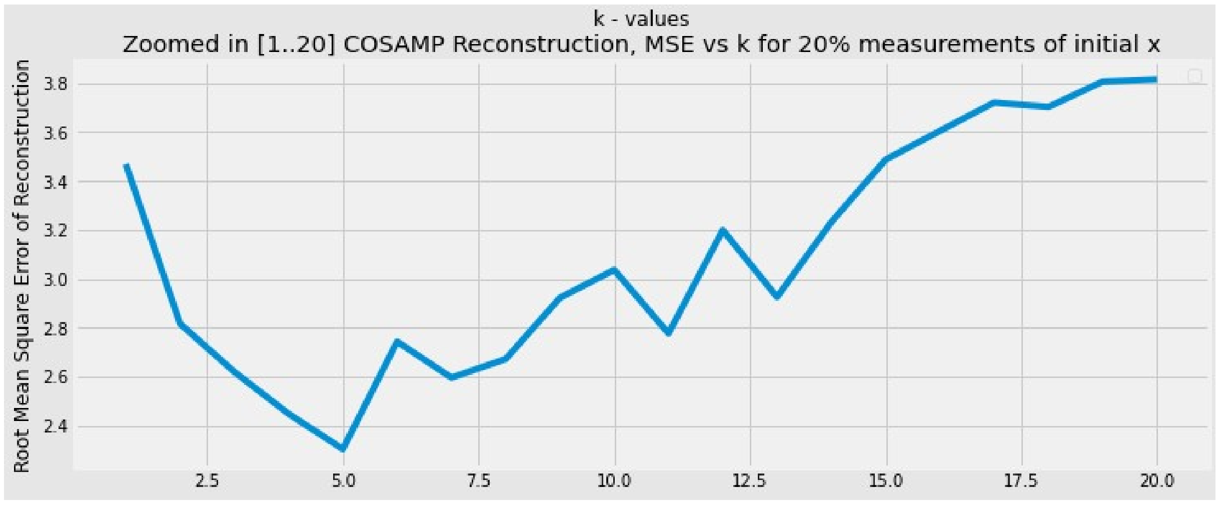


Figure 44B: Zoomed Plotting COSAMP reconstruction MSEfor various sparisty levels

From the above graph, we conclude that with k=5 for 20% of the original signal, we have the smallest error for CoSaMP. Moreover, the range of k values after 200 exhibits stable error fluctuations from 4.5% to 5%, which leads us to focus on the range [0...200] in our next experiment to study all signals.

The percentage of measurements of the original signal x that will create the compressed signal y is one of the main objectives in our work. It is obvious that the more elements we sample, the easier it is for the reconstructor to reconstruct the signal. However, the larger the produced signal becomes due to the smaller compression ratio.

To understand the impact of the sampling rate on the reconstruction error and, consequently, the quality of the reconstruction, the following graph of the CoSaMP solver is provided.

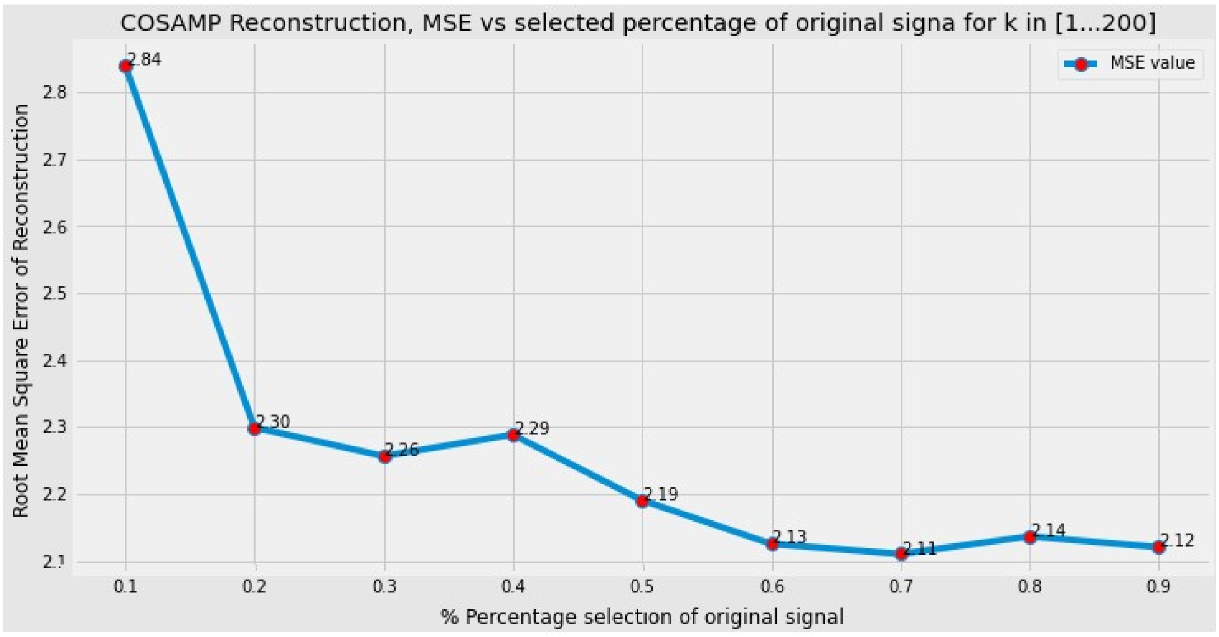


Figure 45: Plotting MSE vs. Measurement of initial signal as a percentage for various k

It should be noted that in Figure 45, we see the best possible performance of CoSaMP for various sampling rates [0.1, 0.2, ..., 0.9] on a specific signal (triangle). Also note that in Figure 45, the calculation was made for all sparsity level values k [1, ..., 200]. For example, the MSE of 2.3% for a 20% rate is the smallest for all sparsity levels, where again, the value of 2.3 was obtained for k=5.

We can see that with a 30% rate, we can achieve better reconstruction results up to 40% of the points of the original signal for this particular solver. Of course, we have better percentages after 40%, but considering the trade-off between accuracy-size, the 20% rate, which means m=160 points, is a very good choice, giving us a very good balance. However, as we said about the trade-offs between ML and CS, the 30% rate, which gives a smaller MSE compared to 20% of the points, will also be studied.

We will study the performance that the machine learning model will have in its attempt to recognize a signal that was reconstructed with 20% and 30% of its original points. The previous experiment was conducted only for one type of signal, the triangle. However, what needs to be studied more extensively is to what extent the results generalize to various input signals. For this purpose, the following graph is provided, showing the MSE of reconstruction for the "good" rate of 20% and for k in the range [1...100], in which it has shown very good results for 3 different signals. One signal for each gesture (M, triangle, circle).

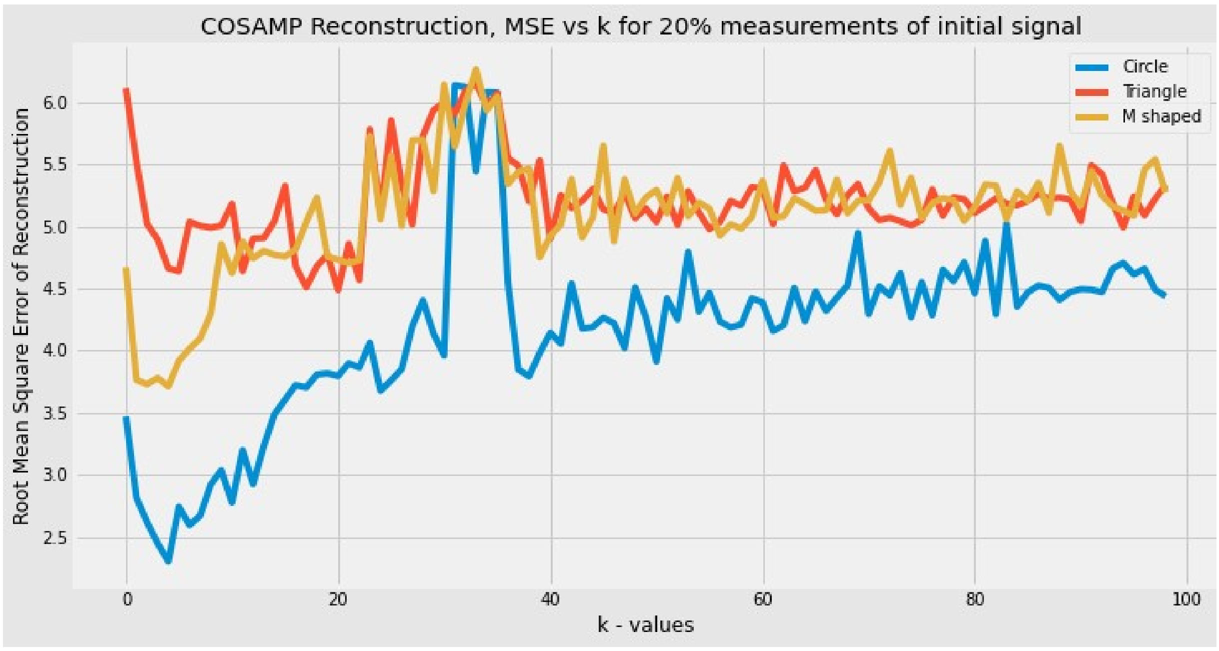


Figure 46: COSAMP Reconstruction MSE vs k 20%

From the above graph, we observe that the region with K=5,6,7 exhibits the best behavior. However, with such a small percentage, we understand that several parameters (s coefficients) with a small impact on the original signal are zeroed out, and therefore, the reconstructed signal is expected to be lacking in information. We also observe that CoSaMP does not generalize well to all IMU signals. For the circle gesture signal, which is governed by low frequencies due to motion, the algorithm performs very well. On the contrary, in the other two motions, M and triangle, due to the abrupt changes in sensor values (higher frequencies), we have a larger reconstruction error. It already seems that, in general, CoSaMP is probably a poor choice if we desire the most faithful reconstruction of the original signal possible.

4.3.2.2.2 Lasso Solver

The Lasso method is essentially a linear regression model with L1 regularization. In this method, a constraint is added to the sum of the absolute values of the coefficients. This reduces the number of non-zero coefficients and identifies the most significant features in the data, preventing overfitting.

One of the key parameters in the Lasso solver is the alpha parameter. It represents the strength of the regularization, or the amount of shrinkage applied to the coefficients. A lower alpha value results in less shrinkage, which leads to a more complex model with more non-zero coefficients (lower sparsity) ass seen in Figure 47. That means that more of the signal's features will be retained. Analyzing the importance of the alpha hyper-parameter on an IMU (Inertial Measurement Unit) signal, we could say that, this could be beneficial if the signal contains important information that needs to be preserved, such as fine details in the movement of the IMU. A low alpha value result in more detailed motion features like acceleration.

On the other hand, a higher alpha value results in more shrinkage and a simpler model with fewer non-zero coefficients. Large alpha will result in a more abstract representation of the signal, with fewer non-zero coefficients, which could mean that some of the signal's features are lost. It is easy to understand that the reconstructed signal would be less informative or less useful for α specific task. This could be beneficial if the signal contains noise or irrelevant information that needs to be removed in order to improve the overall performance of the system.

The alpha parameter vs. Sparseness is presented Below in Figure 47. The graph is showing the relationship between the alpha parameter of the Lasso algorithm and the sparsity of the solution obtained by the Lasso algorithm. It can be observed that at a very low alpha value, around 0.2, the algorithm is able to produce a highly sparse solution with almost 98% sparsity. As the alpha value is increased to the range of [0.01..0.08] (as seen in the right zoomed subplot), the sparsity remains high at around 96%. This indicates that Lasso is able to effectively regularize the solution and produce a sparse result with a high percentage of zero coefficients when the alpha value is very low.

However, it's important to note that as the alpha value increases, the Lasso algorithm also becomes more restrictive, in terms of the parameters, it allows, and as a result the fit of the model to the data may decrease. This is the trade-off when using Lasso, as it balances between the sparsity of the solution and the fit of the model to the data. It can also be inferred that the problem, may have a high degree of sparsity and therefore, Lasso is able to recover the correct sparse solution using a low regularization strength. Therefore, it's crucial to experiment with different values of alpha and evaluate the model's performance using appropriate metrics to find the optimal value that balances the trade-off for the given problem.

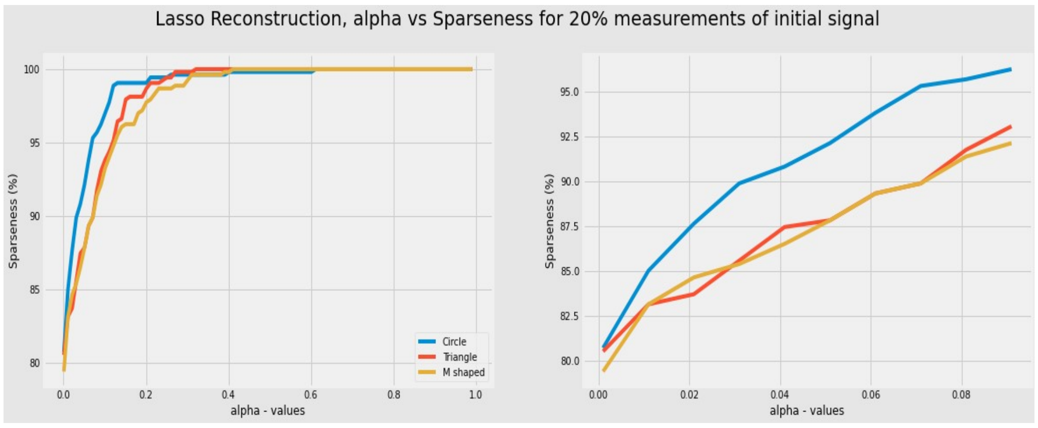


Figure 47: plotting alpha lasso hyper parameter vs sparseness

The study of alpha versus the mean squared error (MSE), as a metric of the reconstructed signal can be a useful method for finding the optimum alpha for the Lasso algorithm. By plotting the MSE of the reconstructed signal for different values of alpha (Figure 48), we can observe the relationship between the regularization strength and the fit of the model to the data. As the alpha value increases, the Lasso algorithm becomes more restrictive, resulting in a more sparse solution but a decrease in the fit of the model to the data, resulting in a higher MSE.

It can be observed that at very low alpha values, around 0.031 and 0.051, the Lasso algorithm is able to produce a highly accurate solution with a minimum MSE. As the alpha value increases past that point, the MSE constantly increases, indicating that the regularization strength becomes too high and the fit of the model to the data starts to decrease.

This is consistent with the trade-off between sparsity and the fit of the model to the data (Figure 47), as the alpha value increases, the Lasso algorithm becomes more restrictive, resulting in a more sparse solution but a decrease in the fit of the model to the data, resulting in a higher MSE. It can also be inferred that for the specific signals of circle, triangle and M the Lasso algorithm is able to recover an accurate solution using low regularization strength.

Additionally, it can be seen that the MSE is becoming constant after an alpha value of around 0.3, this suggest that above that value the regularization becomes too strong, and the Lasso algorithm is unable to produce a good fit of the model to the data, resulting in a relatively constant MSE. So we can clearly see in the zoomed part of the plot that the optimal values for alpha for the given signals are 0.031 and 0.05. We will use the mean of them alpha=0.037 as the best alpha for all the signals.

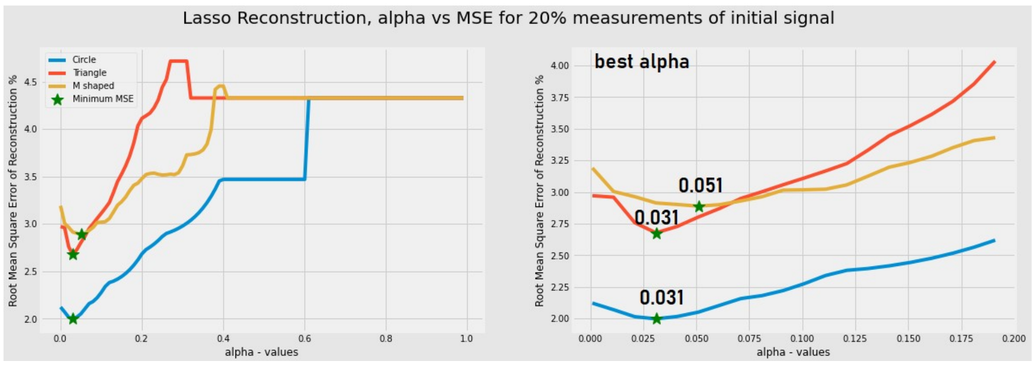


Figure 48: alpha parameter vs MSE

To sum up all the previous a plot of sparseness versus mean squared error (MSE) of reconstruction is presented in Figure 49. We can use it to understand the relationship between the sparsity of the solution obtained by the Lasso algorithm and the fit of the model to the data. It visualizes the trade off we talk before. We clearly see that there is a point where a signal is sparse enough to have a well fitted model and to give the smallest MSE. More sparsity we have lack of information because of too many coefficients (some of them important) are set to zero. On the other hand if the signal is not sparse enough there is noise, or redundant information.

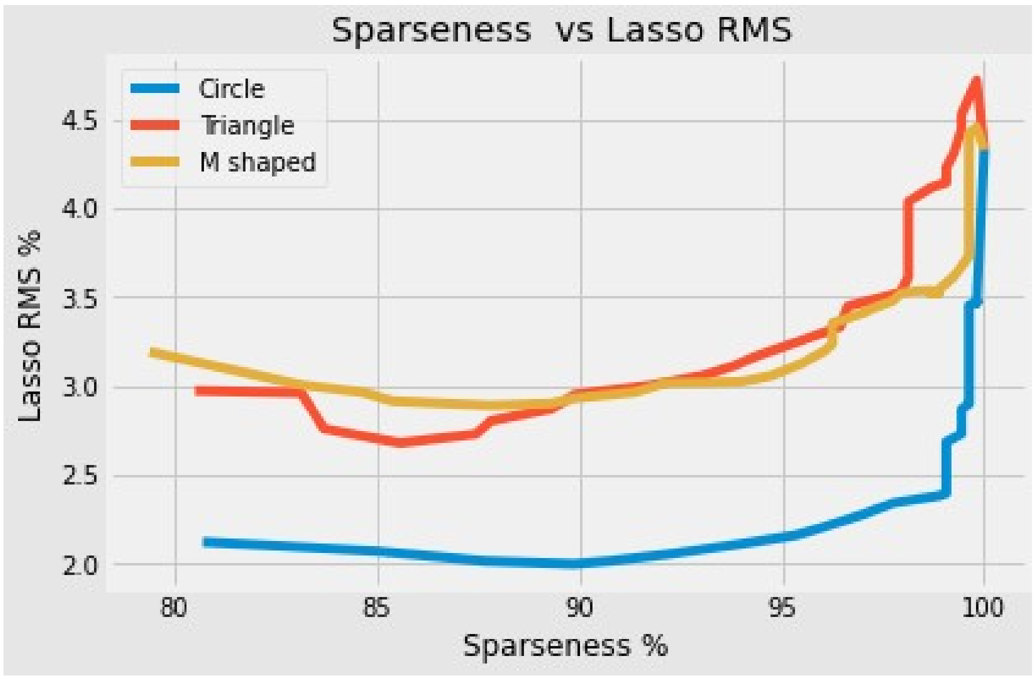


Figure 49: Sparseness vs. MSE tradeoff

So far we have sampled 20% of the points of the original signal to see the effect of alpha on sparsity and MSE. Now keeping alpha value as 0.037 we will study this effect at different sampling rates from the original signal in order to apply CS and reconstruction.

The plot in Figure 50 represents the relationship between the percentage of measurements of three initial inertial measurement unit (IMU) signals of human gestures, and the mean squared error (MSE) of Lasso reconstruction for an optimal alpha parameter of 0.037.

The relationship between the percentage of measurements and the MSE can be analyzed in terms of the trade-off between the amount of data used for reconstruction and the computational cost. As the percentage of measurements increases, the amount of data used for training also increases, which improves the fit of the model to the data, as evidenced by the decrease in the MSE values. However, we need to see the impact of the amount of data that represents the compressed signal, on the computational cost of training the model.

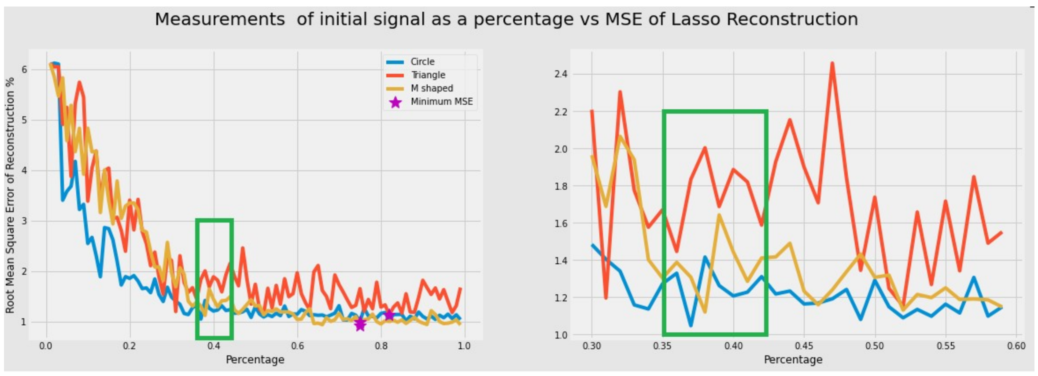


Figure 50: Random sampled measurements of x (as a percentage) vs MSE of reconstruction using alpha=0.037

It is important to analyze the point (green area) beyond which the MSE stops decreasing significantly as the percentage of measurements increases. This point is in the green area of the plot and represents the optimal balance between the amount of data used for training and the computational cost. By choosing the optimal percentage of measurements, we can achieve a low MSE without incurring in an excessive computational cost. Additionally, it's crucial to consider that this graph assumes that the optimal alpha value for the Lasso (0.37) has already been selected, which would affect the sparsity and the MSE of the reconstructed signal.

In a research context, this plot can be used to support the hypothesis and the conclusion of the thesis, also this plot (Figure 50) can be used to compare the results with other methods and to evaluate the performance of the proposed method.

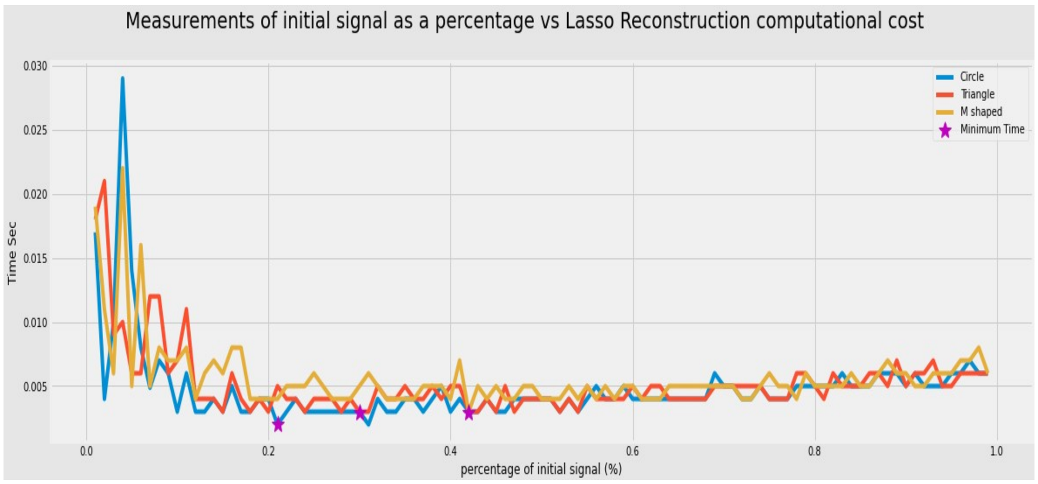


Figure 51: Measurements of initial x for CS vs. computational cost for reconstruction

Observing the Figure 51, we notice that it is not what we would initially expect.

One would expect that as the sampling rate from the original signal to create the compressed signal y increases, so would the processing time to reconstruct the original signal x from y. The figure shows the change in computational cost in sec, as increasing the number of random selected points from the original signal. We notice that, for very few points of the original signal, we have a very high computational cost. This happens because the fewer points the compressed signal y leads to fewer equations for our non-deterministic system of equations. We should not forget that we have: y=A*x=x=A*Ψ*S=Θ*S. S is a vector of n points and this equation defines a system of m equations, (as many as the measurements of original signal x), with n unknowns and we search for the sparsest s that solves it. If m is very small then we have a high computational complexity and therefore a high time cost of solving the system. As the points increase, so does the number of equations and therefore less time is required to solve them. Additionally, when the number of measurements is increased, the Lasso regularization becomes less important and the coefficients are less likely to be close to zero, reducing the sparsity of the solution and making the problem easier to solve.

We see that in the green area we had identified and reported in the previous Figure 50 has entered a zone of low MSE. We notice that this zone (green area) is of low computational cost. This means that the system now has enough equations to solve it in a very short time, and indeed in the minimum possible time. The region of low computational cost starts at 20 % and is maintained even if we select all points of x. Roughly we can say that the percentage of 20% is a percentage that is going to play a very important role in our study in connection with Machine learning and will be studied further

4.3.2.2.3 CVXPY as a solver

The problem of reconstructing a signal from a small number of measurements can be formulated as a convex optimization problem, and CVXPY is a powerful Python library for solving convex optimization problems. In this section, we will study CVXPY in compressed sensing. The optimization problem can be defined by the user in a natural, mathematical style with CVXPY, which then automatically transforms the problem into a standard form that can be addressed by a number of solvers including ECOS, SCS, and OSQP. This makes it simple to use and removes the requirement for the user to manually transform the issue into a standard form. The library includes support for various objective functions and restrictions as well, making it possible to address a variety of optimization issues. It can manage both dense and sparse data and supports complicated data types like matrices and arrays. As a result, CVXPY is a flexible library that may be applied to a variety of compressed sensing issues. Another advantage of using CVXPY is its efficiency.

CVXPY uses advanced optimization algorithms and solvers to solve the convex optimization problem, and it is designed to scale well to large-scale problems. However, one of the main disadvantages of using CVXPY is the computational cost of the optimization algorithm. As the size of the problem increases, the computational cost of solving the optimization problem also increases. This may make CVXPY less suitable for large-scale problems or real-time applications where computational cost is a major concern such a raspberry pi sink node.

Without hyper parameters, cvxpy was applied to our three signals for different percentages of the original signal, and we studied the reconstruction error. The results are presented in Figure 52:

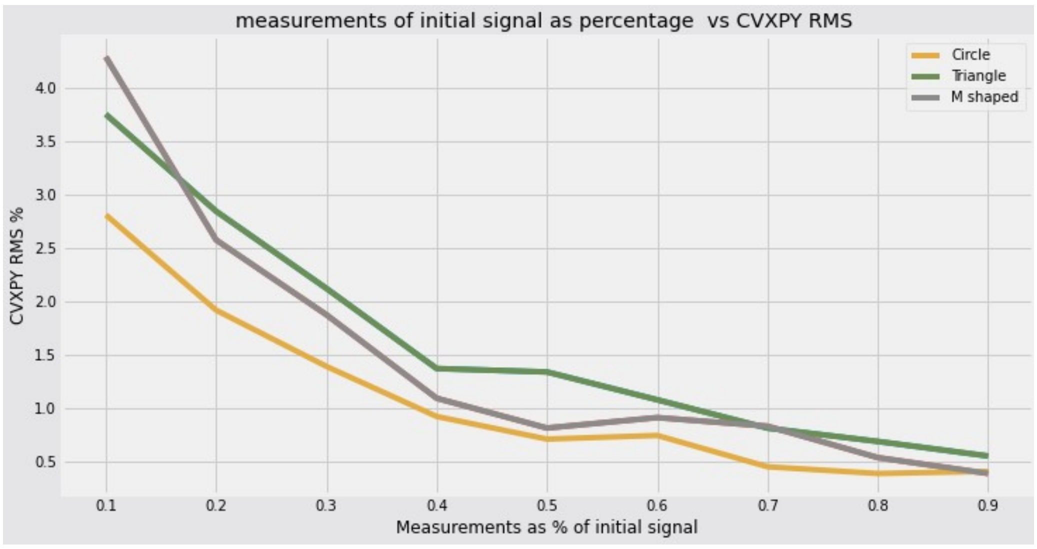


Figure 52: sampled measurements of initial signal vs MSE of reconstruction

We can easily observe that as the percentage we take from the original signal increases, we have a smaller reconstruction error, which is logical and expected. However, we notice that the error is very small, especially in the 30 to 40% range we have mentioned before. At first glance, we can say that it provides the best results compared to the others, although we will examine this more thoroughly later.

It is known that CVXPY has a high computational cost. Below, in Figure 53, a representation of the computational cost in relation to the percentage taken from the original signal is presented.

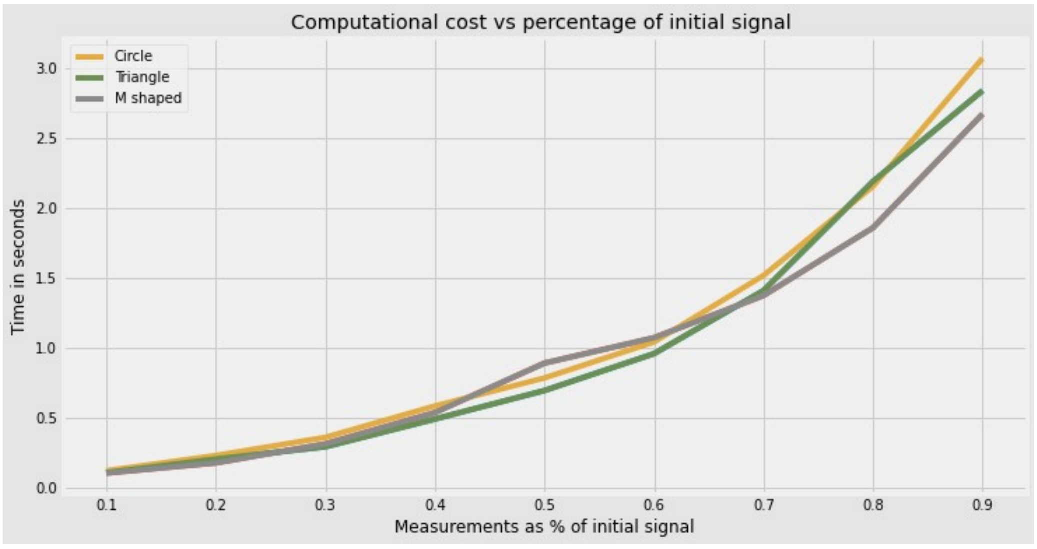


Figure 53: CVXpy computational cost vs. percentage of initial signal

It is obvious that the more points we take, the more time the process requires, in contrast to Lasso due to the method of solving the system. We even notice that for high percentages of initial measurements, it is on the order of 2 seconds. The 30-40% range continues to give us very good results in this case as well. We would expect similar behavior in computational cost with Lasso, which does not happen. There are two main reasons that affect the performance in solving this solver: the choice of the approach method for the solution and the size of the problem. The times presented have been calculated using the best approach for this particular signal form, ECOS. The other methods we examined and chose ECOS among them are: SCS, CVXOPT, GLPK, GUROBI, and MOSEK. Therefore, the delay increases as the number of measurements increases because the size of the optimization problem also increases.

4.3.2.3 Solver Comparison

Next, a comparison of the 3 algorithms with the optimal settings, as they emerged from the thorough study we followed and described in detail above, will be made.

Below in Figure 54, a summary chart is presented showing how the reconstruction error varies for each solver for each of the three different signals for various percentages of the original signal.

With the RMS error as the criterion, CVXPY appears to be the best choice for reconstructing the cycle. In the case of the other two signals in the 30-40% range, we have similar performance. CVXPY is slightly better. A key parameter for our system is the computational power of the sink node (raspberry pi). So, we will proceed to compare the methods concerning the computational cost.

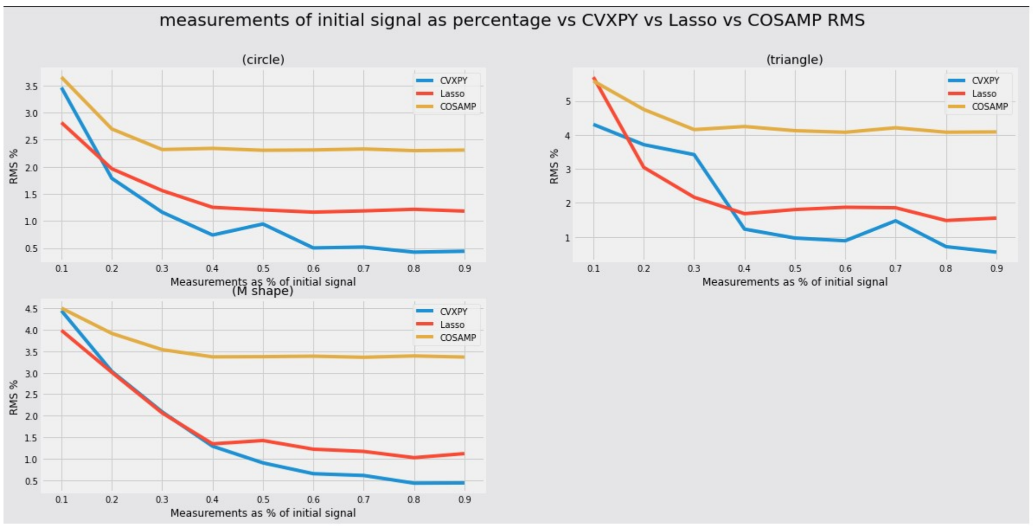


Figure 54: comparing the 3 solvers @ measurements vs RMS

In the following Figure 55, it is now easy to say that the best method for reconstructing the original signal for the case of IMU signals we are studying is Lasso. The reason is that not only in the 30-40% area that we have identified as the best, but also in general, XVCPY has a much higher computational cost compared to Lasso. Moreover, it is significantly higher in proportion to the smaller RMS it offers. It will be used in the subsystem for reconstructing the original signal of the system of the present work with the hyper parameter alpha set to 0.37.

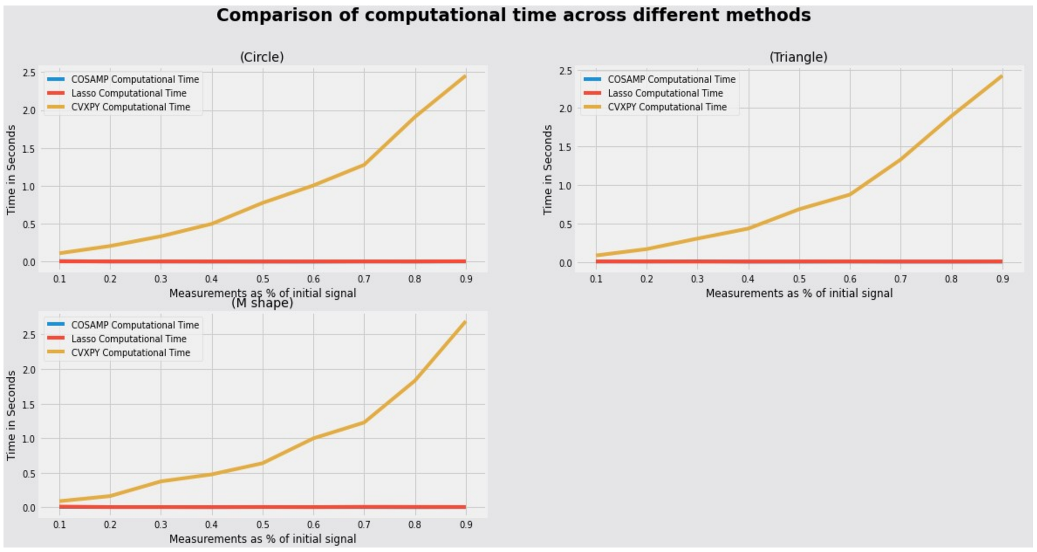


Figure 55: 3 solvers comparison @ computational cost

5 Evaluation

5.1 Implementing the ML και CS system

The system of the study of this thesis consists of two main subsystems, the machine learning subsystem and the compressive sampling subsystem. In the previous sections of this unit, we determined the basic parameters of each system to achieve the best performance by studying each system separately. More specifically, we saw that the most suitable classifier is the SVM, and we identified its hyper parameters (c=2, gamma=0.1, kernel=rbf). Similarly, the CS system will use DCT representation basis, Lasso solver (L1 minimization), and a random sub-sampling array m-Bernoulli. But what will be the final configuration of the hyper parameters of these two systems when they have to collaborate?

The main objective of this master thesis is to find the minimum number of observations m of the original signal x that I must take so that the reconstructed signal x_hat is recognizable by the machine learning model, as well as the original one.

In the context of answering this question, we must work considering three basic elements that define the system.

- Best performance at the ML level (best accuracy)
- Best performance at the CS level to have more faithful reconstruction and therefore recognizability
- Less need for computational resources due to the microcontroller.

By reducing the size of the dataset, the number of required support vectors is reduced, and therefore the size of the SVM classifier. However, what is the impact of reducing the dataset size on the generalization of the model? In other words, how well can it separate a reconstructed signal depending on the number m of points we will take from our original signal?

As mentioned in figure 34.B, we got a good cross-validation performance from SVM when we removed features from the dataset that had a 99% correlation between them. Next, we will essentially proceed to evaluate this feature selection method by which we removed features from the dataset that had a relationship between them over a percentage. We will study a model that has been trained a) on the entire set, b) on a set where features with a relationship over 99% were removed, and c) features with a relationship over 99.5%.

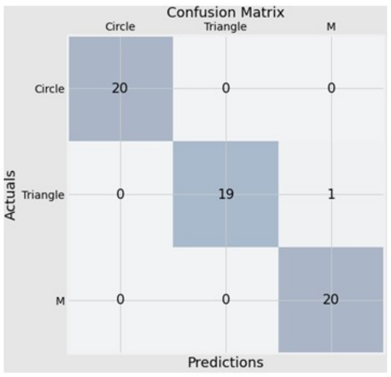
For the evaluation of the model's generalization capability, after training the SVM, we will use an evaluation dataset. This dataset contains a set of 30 different movements corresponding to the gestures we study. It includes 10 samples from each of the 3 gestures that the model has never encountered during the training stage. We should also make it clear that we have divided the original dataset into train and test. We train on the train and evaluate on the test. After getting the evaluation score of the SVM using the test set, we then fit the classifier to the entire original dataset and proceed with its further evaluation.

We should also make it clear that we have divided the original dataset into train and test. We train on the train and evaluate on the test. After getting the evaluation score of the SVM using the test set, we then fit the classifier to the entire original dataset and proceed with its further evaluation.

The evaluation of the final trained model will be done 1) on the evaluation dataset and 2) on the reconstructed evaluation dataset that will result from the compression and reconstruction of the original evaluation dataset for various sampling rates.

Training the SVM on the entire original dataset (534 points).

By training the model using all 534 points (without essentially any feature selection), we get very good test set evaluation results as shown in Figure 56. The model simply seems to confuse the formation of the triangle with the formation of the letter M. This is because the two movements have a high degree of similarity. Essentially, the triangle is almost a subset of the "M".



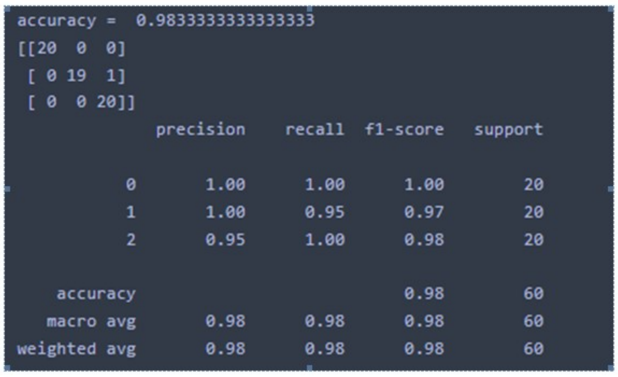
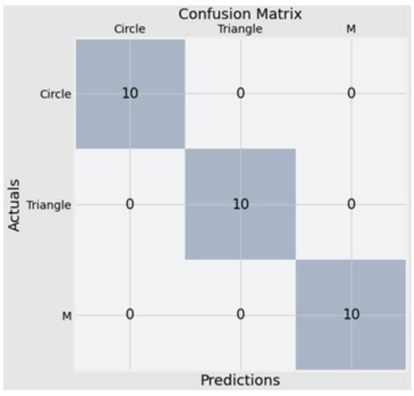

```
accuracy =  0.9833333333333333
[[20  0  0]
 [ 0 19  1]
 [ 0  0 20]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       1.00      0.95      0.97        20
           2       0.95      1.00      0.98        20

    accuracy                           0.98        60
   macro avg       0.98      0.98      0.98        60
weighted avg       0.98      0.98      0.98        60
```

Figure 56: SVM evaluation (trained in train set with 534 features) using test set

We evaluate the trained model with the evaluation set that the classifier has never seen before. We observe that all 30 movements are recognizable. The results and the confusion matrix of the evaluation can be seen in Figure 57. It is obvious that our model has excellent generalization.



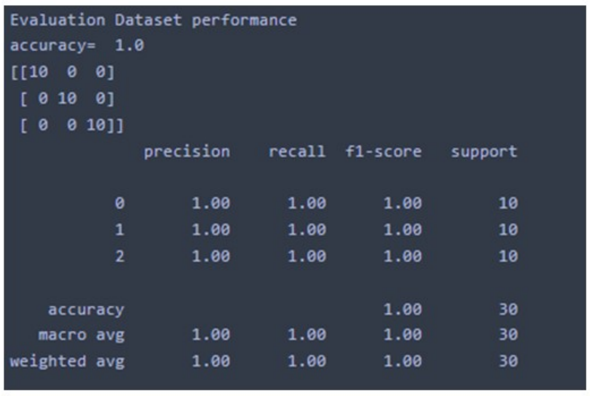

```
Evaluation Dataset performance
accuracy=  1.0
[[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        10
           2       1.00      1.00      1.00        10

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Figure 57: SVM evaluation (trained in initial dataset with 534 features) using evaluation set

Next, we will study the impact of compressed sensing on the performance of the trained model. For this study, we applied compression and reconstruction using the compressed sensing technique to the entire evaluation dataset. We fed the reconstructed evaluation dataset to the model and obtained its performance (accuracy). The process was carried out for various sampling rates from the original signal, [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]. For each percentage, we took a corresponding number of points from each signal in the evaluation dataset and reconstructed it with the CS subsystem. We fed the model with the reconstructed dataset each time, and the model's performance in terms of accuracy is depicted in Figure 58.

In the first row on the right are the sampling points expressed as percentages of the length of the original signal, and on the left in absolute point values.

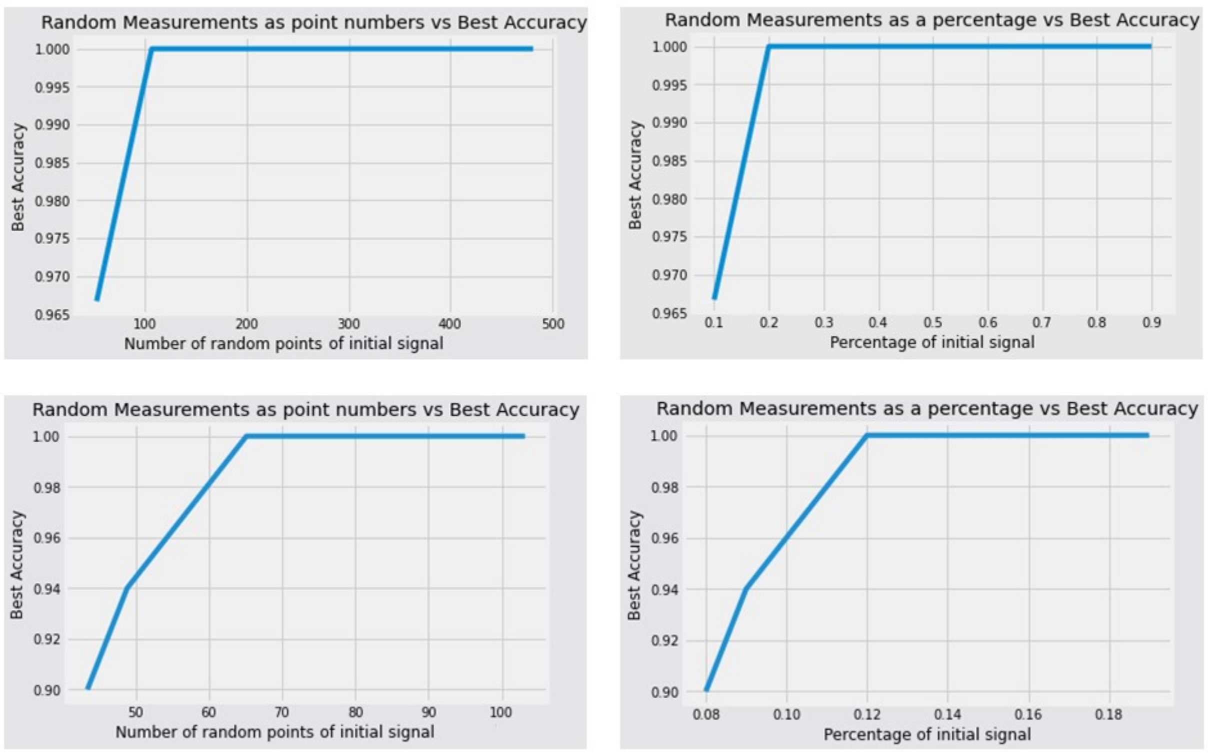


Figure 58: SVM (trained with 534 feats) performance on reconstructed evaluation dataset for various sampled measurements rates.

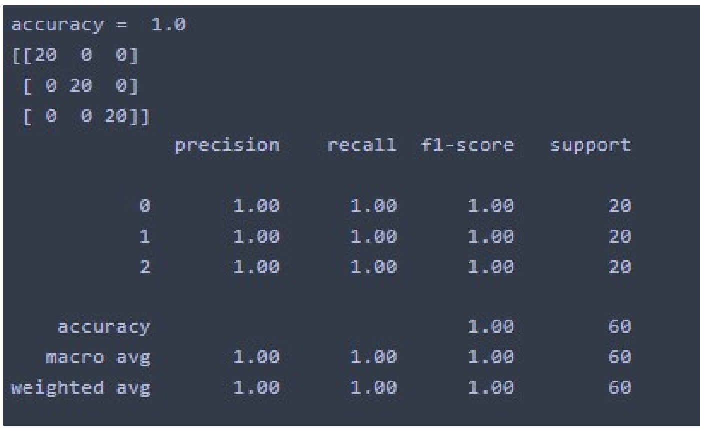
In the above **Figure 58**, we present the impact of CS on the recognition of the reconstructed signals in the evaluation dataset. We observe that in the area around 10 to 20%, we achieve a recognition rate of reconstructed signals of around 100%. However, since the fewer points we use for compression, the lower the transmission cost to the sink node, we further analyzed the area from 8% to 20% for greater accuracy, and the results are shown in the second row of the previous graph. We see that for 12% of the original signal, our classifier can recognize the entire reconstructed evaluation dataset (all 30 signals contained within it). Stating the above in relation to the main question of the present work, the minimum percentage of a signal that we must take for an entire dataset of sets to be recognizable after reconstruction is 12%. If we want to convert it to points, it is 0.12*534 = approximately 64 points.

We must not forget, however, that the model must be able to run on both leaf nodes (Arduino Nano 33) and sink nodes (Pi 4). Especially for the microcontroller, we have very limited resources, so we need to optimize the dataset by keeping only the necessary elements. Due to the extensive arrangement of the 534 points in the dataset, the SVM requires many support vectors, which means high memory consumption and computational power for executing large operations on the Arduino. Additionally, in the entire dataset, our data is not linearly separable, so an RBF kernel is of higher computational complexity. Therefore, we need to study whether fewer features can give us a dataset with equally high-performance models.

We saw in the Feature Engineering and Selection section that if we remove a percentage of features that have 99 or 100% correlation between them, we achieve a maximum accuracy of 97% in Figure 34B. Since the percentage we remove affects both the size of the model and the result of the reconstruction, further study is essential.

Training the model after removing all points with a Pearson correlation of over 99% between them.

At this stage, we removed all the features from the original dataset that were correlated to each other by more than 99%. A new set with 313 features was created, and we trained the SVM on this. We followed the same study procedure we analyzed earlier and got an evaluation score from the test set. The results are shown in Figure 59. At this point, we observe that the accuracy we get from the test set has increased, and we should mention that the grid search we applied for SVM hyper-tuning highlighted the linear kernel as the most suitable. In other words, our data is linearly separable, as the high percentages indicated earlier.



```
accuracy =  1.0
[[20  0  0]
 [ 0 20  0]
 [ 0  0 20]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       1.00      1.00      1.00        20
           2       1.00      1.00      1.00        20

    accuracy                           1.00        60
   macro avg       1.00      1.00      1.00        60
weighted avg       1.00      1.00      1.00        60
```

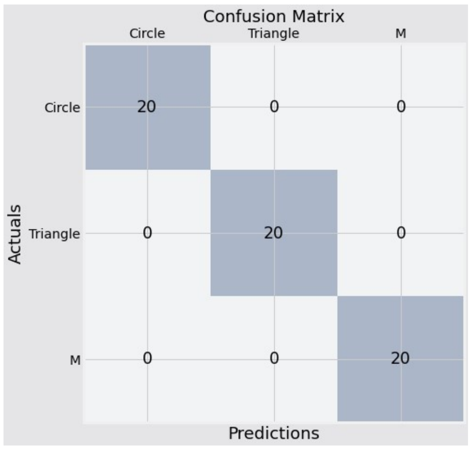
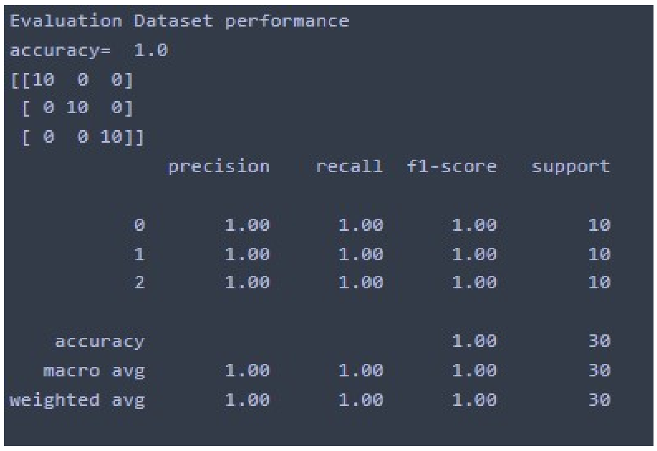


Figure 59: SVM (trained after removing features with 99% pearson correlation to each other) evaluation using test set

Applying the evaluation dataset, we get the following results:



```
Evaluation Dataset performance
accuracy=  1.0
[[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        10
           2       1.00      1.00      1.00        10

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

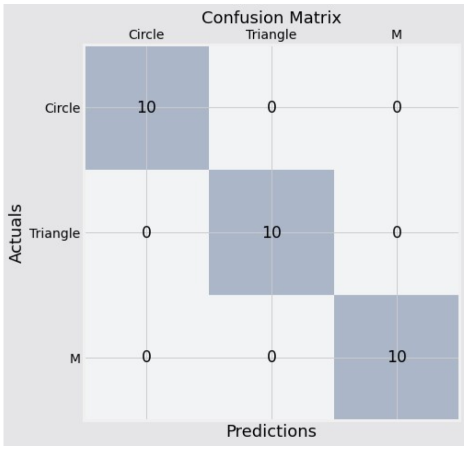
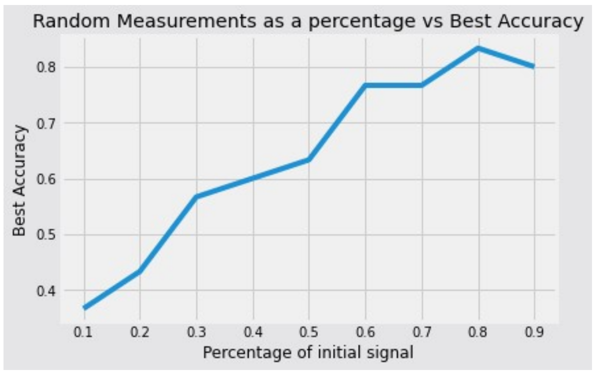


Figure 60: SVM (trained after removing features with 99% Pearson correlation to each other) evaluation using evaluation dataset

At first glance, it seems that an equivalent or even better model (test set evaluation) has been created compared to the previous one. The goal is to see how our new model behaves with the evaluation dataset when it is compressed using sampling and reconstructed. The results of this approach can be seen in Figure 61.



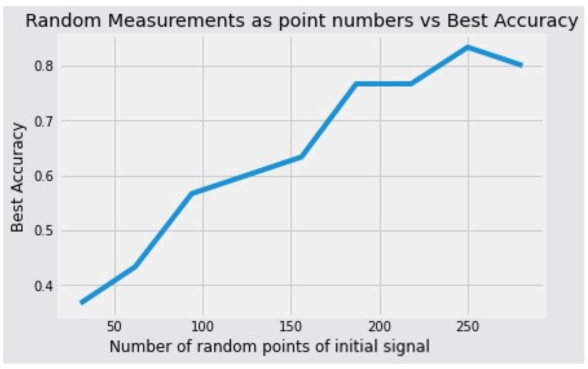


Figure 61: SVM (trained after removing feats with 99% Pearson correlation) performance on reconstructed evaluation dataset for various sampled measurements rates.

It is evident that some of the elements that were removed had a strong relationship with each other, but apparently, they also had a connection with the output. Although we had a good cross-validation performance, the model that was created does not generalize well. We can conclude that there was a significant amount of information within the elements that were removed, which the classifier needed. Additionally, given the loss of information and the inability to achieve 100% reconstruction, this led to a model that cannot even achieve the results we had before the removal concerning the recognizability of the reconstructed signals.

<u>Removal of points that have a correlation between them above 99.5%.</u>

After the reduction, the dataset consists of 491 features. In order to decide if this percentage provides results comparable to the model with the entire original dataset, we will further study the removal at 99.5%. At this percentage, we had the same results (as expected) in cross-validation scores and evaluation dataset accuracy. Next, we present the performance of the model on the reconstructed evaluation dataset.

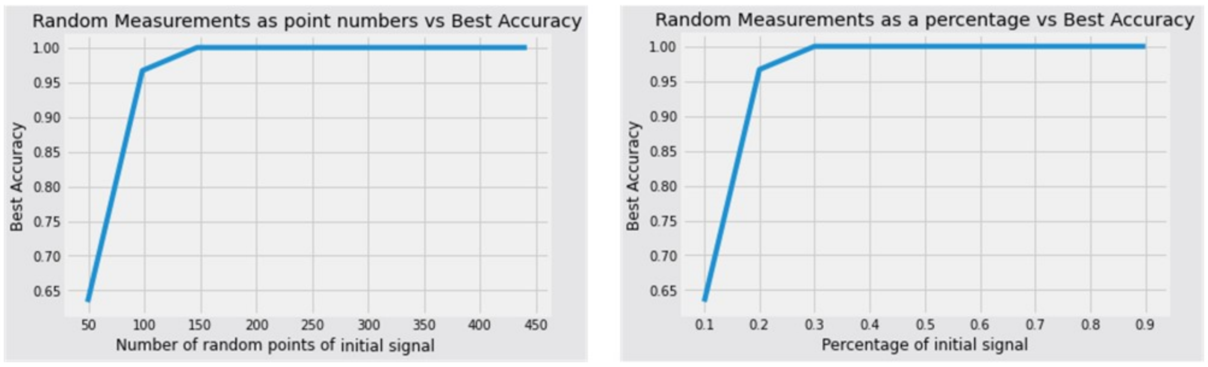


Figure 62: SVM (trained after removing feats with 99,5% Pearson correlation) performance on reconstructed evaluation dataset for various sampled measurements rates.

In the study of the number of points taken from the original signal, we have a better behavior than 99% but still achieve 100% accuracy with a percentage of points between 20 and 30%. This means that from 100 to a maximum of 30/100491=147 points. While the model derived from the entire dataset achieves 100% with only 12% of the total points. This translates to 12/100543=65 approximately points.

Figure Figure 63 presents a summary diagram of the performances shown above, with the difference that a more detailed approach is taken based on percentages. That is, we don't go directly from 10 to 20% but take all intermediate points to see where we achieve acc=1 for the first time.

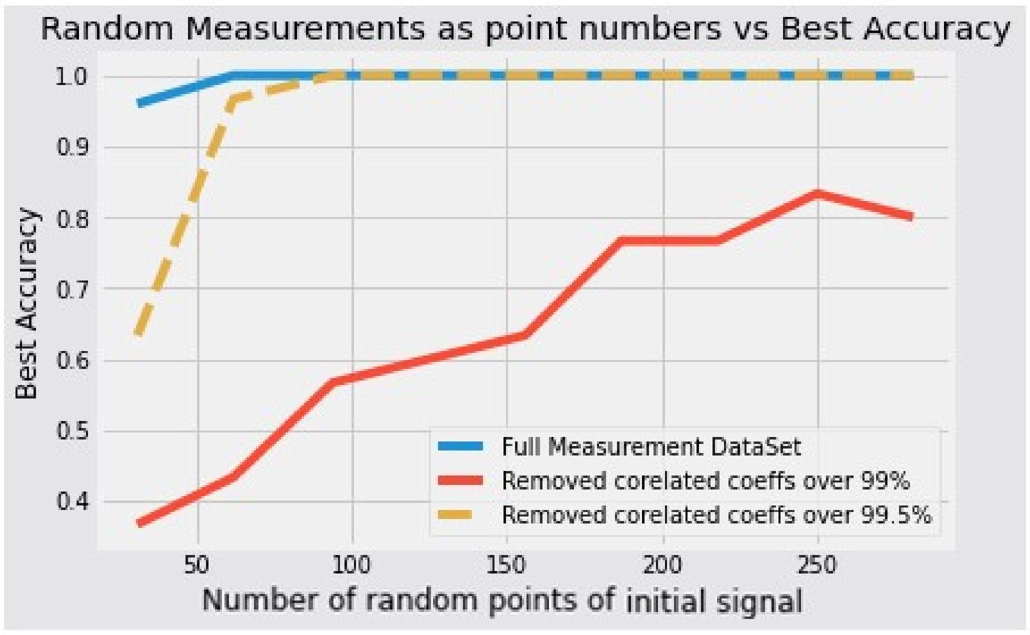


Figure 63: summary diagram according to the number of correlated coeffs removed

So it is obvious that we should choose to create the model from the entire dataset, something that will obviously create a problem in the implementation of the SVM on the Arduino. Therefore, the feature selection method based on the removal of Pearson correlated features, although it creates a very good ML model for the initial signals, cannot create a small dataset and model that successfully recognizes the reconstructed signals. We need to find a more effective feature selection method, which is done in the next section.

<u>Feature selection, creating a model from K-best points</u>

As is obvious from the previous graph Figure 63, the feature selection method by removing points related to each other did not prove to be good because some of the points that were removed had a strong relationship with the output on the one hand and on the other hand, we had additional removal of information with the reconstruction.

The k-best method is a feature selection technique used in machine learning to select the strongest k features that contribute more to the accuracy of a model's prediction. By the term strongest, we mean those that have the highest correlation with the output and not between them as was done in the previous method we presented. In this method, we rank all the features in the dataset based on their predictive power and only the top k features are retained for model training. This helps reduce data dimensionality, avoid over-fitting, and improve training efficiency as will be shown in the following graph. The value of k is a hyperparameter and is often chosen based on a trade-off between model complexity and performance and can be selected using cross-validation or evaluation accuracy.

Next, we studied the performance of the svm when trained on a dataset from which the k-best features have been retained. The k-best varies from the value of 10 up to 534 per 10. In Figure 64, the Cross-validation, the Evaluation accuracy from the test set, as well as the evaluation accuracy from the extra evaluation dataset of 30 movements that the model has never seen are presented.

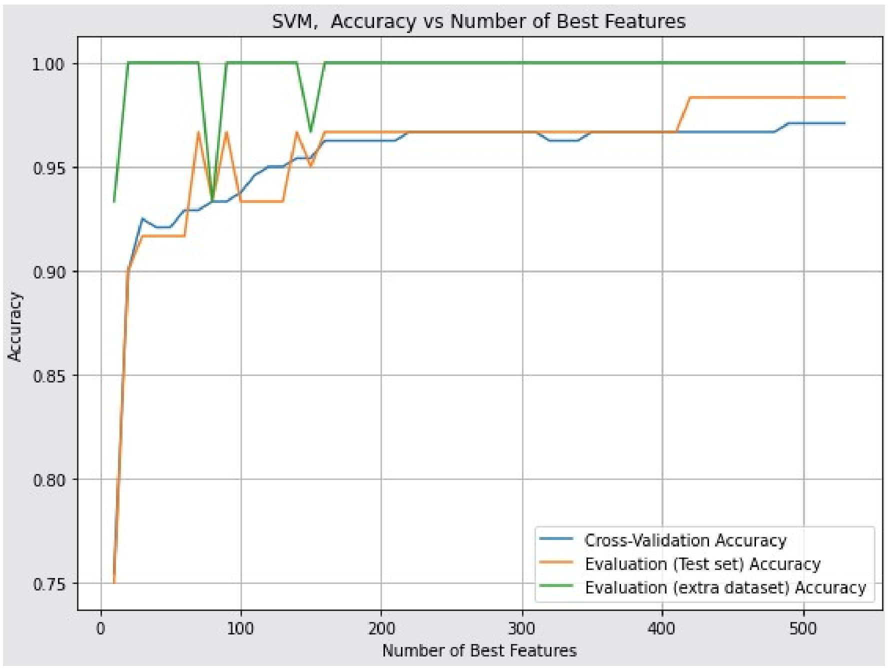


Figure 64: SVM (trained at kbest features) accuracy vs kbest number

It should be noted that the SVM classifier is with the settings we derived during the study of the entire dataset and has been described in detail in the Machine learning implementation section. The system's response to the evaluation dataset is interesting, as one would expect it to be stable, especially from a certain point onwards. Instead, it shows fluctuations. This is due to the fact that the dataset changed composition, and the data became linearly separable, and generally, with the change in the composition of the dataset each time, the shape of the most suitable kernel changed. The kernel of the svm we used was the rbf. The process was repeated, and this time we combined it with the grid-search method for hyper tuning, where it was shown that the linear kernel was the most suitable in the vast majority. We replaced the kernel in the previous process, ran the experiment again, and the results are shown in Figure 65.

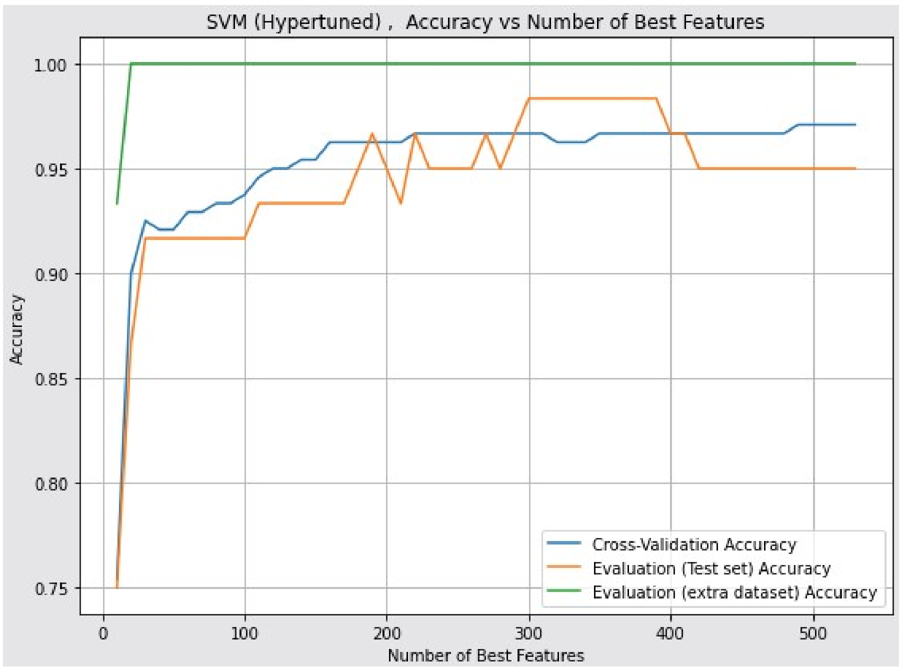


Figure 65: Hyper tuned SVM (trained at kbest features) accuracy vs kbest number

It is evident that the k-best part needs to be studied thoroughly, and this is because we see a very good performance of the model in the evaluation dataset for low values of Kbest, while the evaluation test set score for these values is low. It is obvious that we have a very "convenient" evaluation dataset.

In the context of this analysis, an algorithm was created which will show us the effect of kbest on the performance of the system. This algorithm should give us the minimum number (kbest) of points in the dataset at which we will train the SVM, which will recognize reconstructed signals created with the minimum number of points (m) from the original signal.

At the same time, we want the best possible model at the test set evaluation level. The algorithm will return both kbest and m values.

```
Load initial dataset and evaluation dataset
Create a kbest list to study
For each kbest in list
    Feature selection to both datasets as kbest defines, using grid-search
    Hyper-tune SVM (grid-search) using ft selected dataset
    Create a list of m of initial signal
    Take first m
    While model acc using m<1 or m list is not over
        CS of evaluation dataset
        Reconstruction of cs evaluation dataset
        Evaluate model using reconstructed dataset
        Find reconstruction accuracy for specific m
        Take next m
    Find best reconstruction accuracy per kbest
First m that achieves max accuracy reconstruction is what I am looking for.
```

In the algorithm m is the number of random sampling points that I get from initial signal x in order to create compressed measurement vector y.

The implementation of the above algorithm in Python3 can be found in the appendix of this thesis. According to the above algorithm, we essentially create an optimized SVM based on the initial dataset formed by the feature selection of each kbest. Then we compress and reconstruct each signal found in the evaluation dataset and provide the reconstructed dataset for recognition. Using the accuracy of the model on the reconstructed dataset as a criterion, we find at which kbest and at what percentage m we encounter the maximum response. At the first appearance of max accuracy, we extract the current kbest and m as a result.

By running the above algorithm, we get the performance of the system for each kbest and for each m. The representation of these results can be seen in Figure 66.

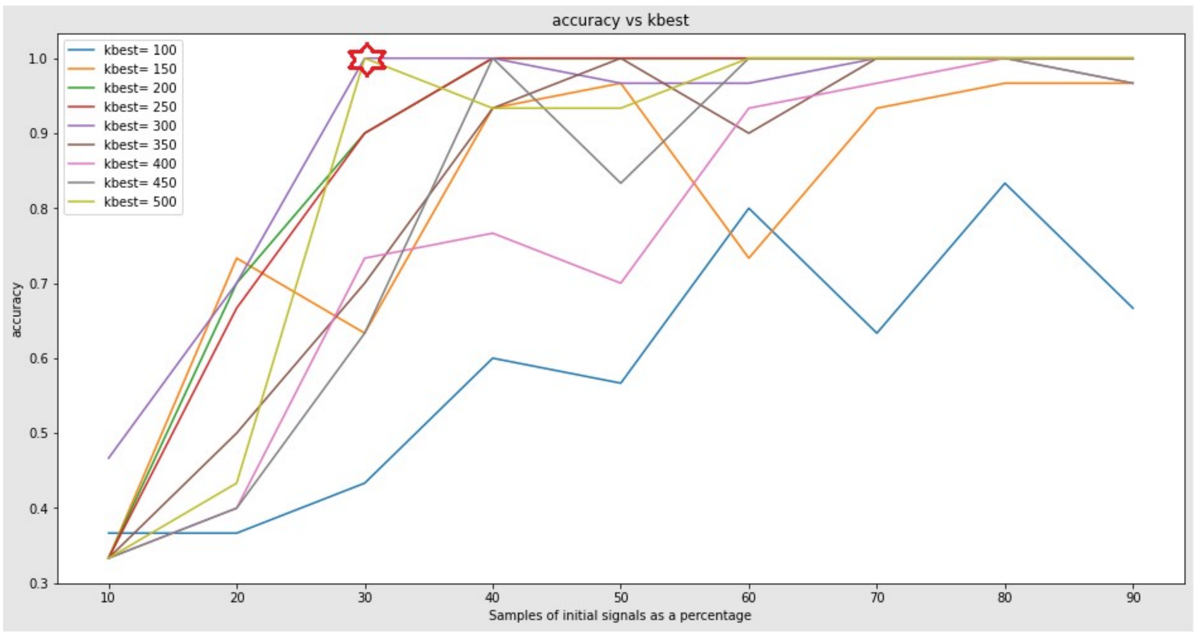


Figure 66: SVM (trained on various kbest) accuracy on reconstructed dataset vs number of measurements of initial signal (using random sampling matrix)

Observing the above graph, we see that we have an excellent response of the system for percentages around 25% for kbest = 300 and 500. That is, we are talking about points around m = 75 and m = 125, which have a significant difference. A question that arises is why do the intermediate values, e.g., 350, have such poor performance in this area? Additionally, since Kbest = 300 achieves 100%, why does it fail later with more points?

The answer comes from the sampling matrix. We saw that randomness is an essential element for CS to be effective. This randomness ensures reconstruction but different matrices will achieve different performances. Therefore, we could enrich the algorithm with the process of determining the most suitable matrix. For this purpose, the above algorithm is configured as follows:

```
Load initial dataset and evaluation dataset
Create a kbest list to study
For each kbest in list
    Feature selection to both datasets as kbest defines, using grid-search
    Hyper-tune SVM (grid-search) using ft selected dataset
    Create a list of m of initial signal
    Take first m
    While current m model acc <1 for or m list is not over
        Find best sampling matrix with m elements
           CS of evaluation dataset
           Reconstruction of cs evaluation dataset
        Evaluate model using reconstructed dataset
        Find reconstruction accuracy for specific m
        Take next m
    Find best reconstruction accuracy per kbest
First m that achieves max reconstruction accuracy is what I am looking for.
```

In the algorithm m is the number of random sampling points that I get from initial signal x in order to create compressed measurement vector y.

Within the framework of implementing the experiment indicated by the above algorithm, we obtain the following graph:

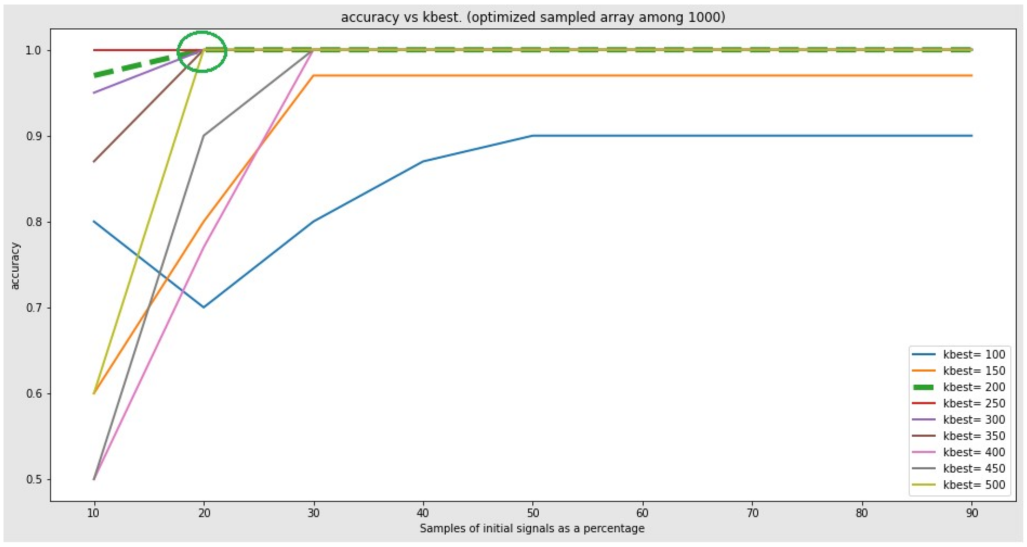


Figure 67: SVM (trained on various kbest) accuracy on reconstructed dataset vs number of measuremets of initial signal (using best samping matrix)

Comparing with Figure 66, the impact of choosing the most appropriate sampling matrix on the recognition by the classifier of the reconstructed signal is evident. In this graph, we can observe many interesting things about CS and machine learning.

In the algorithm that implemented the above experiment, the optimal sampling matrix $A_{mx534,}$, which was created randomly, has been sought among 1000.

We should also make it clear once again that the model that makes recognitions is trained on the dataset where its features have been reduced from 534 to k_best. We see that for k_best 100 and 150, the model does not generalize very well to the reconstructed signals of the evaluation dataset. It is obvious that at 100 points, not all the necessary information contained in a signal can be rendered. However, at 150 points, we can see that our model achieves the performance of the model trained on the whole dataset, about 97%. So, we certainly don't have the best results in these k_best, but in general, we don't have degradation.

The k_best = 200 is interesting, which for a percentage between 10 and 20 percent shows an increase from 97 to 100. We should look for the value at which the accuracy becomes 100 for the first time. This will show us the smallest m for which we have CS and reconstruction of all signals in the evaluation dataset, and our model has the maximum performance.

We conduct the same experiment, limiting the k_best only to 200 and redefining the study percentages to [0.11, ..., 0.19]. To see even better the impact of the appropriate sampling matrix, the results of the experiment are depicted in three ways. In the first plot (blue line), the performance is shown by simply taking the best random sampling matrix among 10 (best 10). In the second case (orange line), the best among 100 (best 100) was chosen, and in the third (green line), it was chosen among 1000 (best 1000), as in the previous summary graph. The results are shown in Figure 68:

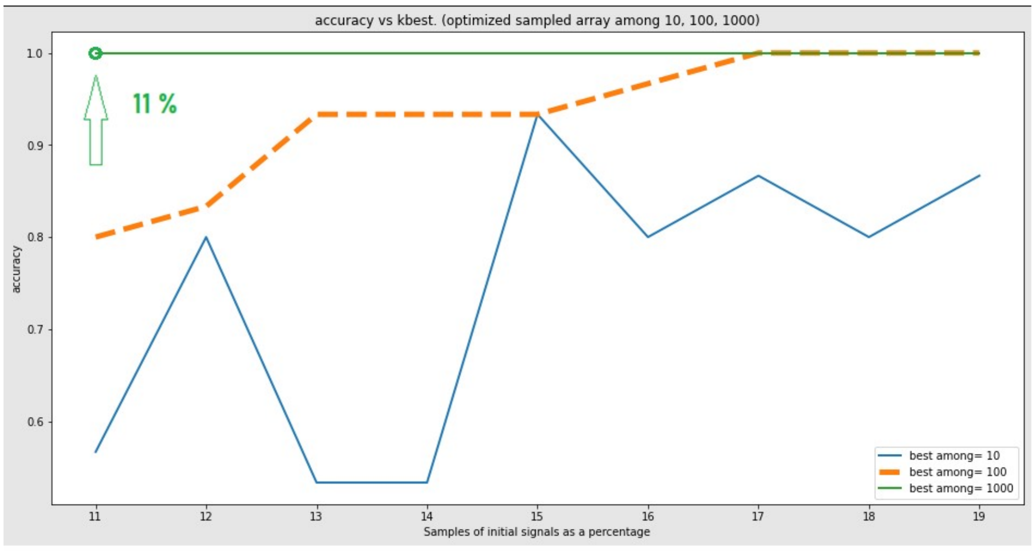


Figure 68: Finding optimum kbest and minimum number m

So, it is at the 11% point for k_best = 200 that we have the maximum accuracy for the first time. This means that even if we take an inertial signal, let's say x, of 534 points in size, we keep 200 of them (which we know exactly which ones) and choose 11% of those as indicated by the optimal sampling matrix

[96,198,164,48,108,107,66,34,177,93,95,136,162,47,139,169,85,52,199,45,97,14]

and we create the compressed signal y with 22 points. We send the y vector to the sink node. There, using the compressed sensing reconstruction technique with the parameters we analyzed, the x_hat vector is reconstructed. The reconstructed x_hat vector is then given to the trained model where it is recognized. This process was repeated for all the signals in the evaluation dataset, and all 30 were recognized. This is a good indicator for us to understand that this point is the answer to our question. By selecting 22 points from an inertial signal with 534 points, the signal is compressed and reconstructed using CS techniques, and the reconstructed signal is recognizable by a machine learning model that was trained with k_best = 200 features of the uncompressed original dataset.

In the summary graph, we can also see that while k_best = 250 achieves maximum system response at every percentage after 10%, choices of k_best = 500 have a smaller response in the area up to 15%. The explanation is again in the sampling matrix. More specifically, when I have k_best = 500, then the strong features are among 500, while in k_best = 250, they are among 250. In the second case, I have twice the chances of finding a more suitable matrix among the 1,000 I'm looking for.

The number of combinations that can be created with a set of 250 elements, taking m of them but one at a time, is given by the binomial coefficient, denoted as C(250, m). This is calculated as follows:

C(250, m) = 250! / (μ! * (250 - μ.)!).

In order to determine the difference, we calculate based on the above formula:

C(250, m) = 1.31e+45

C(500, m)= 1.64e+57

This means that C(500, m) is 1,258,901,483,489 times larger than C(250, m).

We will not go through the process of calculating the optimal sampling matrix for these Kbest values since we found what we were looking for at Kbest=200. Let's not forget that our main goal is to identify the smallest number of elements. Therefore, we will not analyze further; we simply explain what exactly is happening and why this paradox appears.

This page has intentionally been left blank.

6 Conclusions and future work

In the context of conducting this thesis, we thoroughly studied the subject: "exploiting compressed sensing in distributed machine learning". Our goal was to develop a method with the help of which a signal would be captured in a distributed machine learning environment (sink node) and would undergo compressive sampling to the maximum extent possible. At the same time, however, the compression ratio on the original signal would make it capable of being reconstructed when it reaches its destination (leaf node). The reconstruction error of the method should not affect the ability of the machine learning model we have developed to recognize the reconstructed signal. It should be emphasized that the machine learning model, an SVM classifier, is trained on the uncompressed original acquisitions of the signals.

In order to develop the above method, several questions had to be answered, and these questions came with the design and execution of suitable experiments. What made this study particularly demanding was the common application area of two techniques, machine learning and compressive sampling. The configuration of one technique's parameters (e.g., ML) had to be done in conjunction with the impact not only in its area but also in the CS area.

At the machine learning level, the feature selection method proved that gridsearch, with which we maintain the kbest strongest features, is more suitable compared to the removal of those with the highest correlation (Pearson correlation) between them. Although the model in the second case has excellent performance on the evaluation dataset data, when the entire dataset is compressed and reconstructed, the performance is reduced. The SVM proved to be a more effective classifier compared to many others, and the use of the linear kernel proved to be more suitable after the final shaping of the dataset from the selection of features.

At the CS level, it was proven that the most suitable solver was Lasso with the setting of its hyper parameter at 0.037. We saw that with the help of the random subsample sampling matrix, which we adopted, we can achieve very good results. Regarding signals originating from an IMU (accelerometer values in time), we found that the best transformation basis was the DCT compared to wavelets.

Summarizing all of the above and in response to the main research question of this thesis, we can say:

In a distributed machine learning environment, at the leaf node on the extreme edge where an Arduino Nano 33 BLE Sense microcontroller is located, we have a signal captured from its embedded IMU. This signal consists of 534 data points (features). By applying feature selection using the grid search method, we choose the 200 best features. This signal undergoes compressive sampling at an 11% rate, meaning 22 features. This signal is then sent to a Raspberry Pi 4, a sink node, which reconstructs the signal and feeds it to an SVM for classification, and the signal is successfully recognized. To study the performance of the classifier, a set of 30 signals was compressed and reconstructed as described above, then fed to the SVM, and all 30 were successfully recognized.

We will now mention some possible directions for potential future work based on our findings and the limitations we encountered in our research. Our approach has shown many useful results for addressing and managing the research problem we faced. However, there are several points where further investigation could help improve performance and extend the applicability of our approach. The research community has shown increasing interest in this area that we have studied, and we believe that the following directions can contribute to advancing the field. By exploring these directions, we can gain a deeper understanding of the underlying mechanisms at the intersection of machine learning (ML) and compressive sensing (CS). At the same time, we can evaluate the performance of our approach on different signals beyond IMUs (such as images, sounds, etc.) and develop new techniques and applications for the benefit of the broader community.

In the approach of the method we created, we used the traditional signal reconstruction process of CS. More specifically, the reconstruction came with the help of determining the sparsest signal representation of the original signal in the DCT representation basis. This is essentially the solution to a convex optimization problem. How would the findings of the method we created change if a neural network was used for signal reconstruction? The compressed signal that would come to the sink node would feed its input and at the output we would get the reconstructed original. How more accurate would the reconstruction be and, consequently, how many fewer or more points than 22 would I have to take in order to achieve the same results?

We studied the role of the sampling matrix. A subject of study could also be the development of a method for creating a universal sampling matrix that would be adaptable to a variable size of the sampling points from the original signal. That is, can a neural network be created that will accept the original and the compressed signal as input and output the optimal sampling matrix? This method could be generalized to search for a model that would be given the original signal and the number of points and output the compressed signal with the optimal sampling matrix, which would also result in the smallest reconstruction error.

Another interesting point of study that arises from this work is how the results of the current method would be shaped if the machine learning model was not trained only on the compressed signals but on a dataset consisting of the original and reconstructed signals. What would be the generalization of the model? How many would be the minimum points I should take in this case? Generalizing this approach even further, what if the model was trained on the k-most significant coefficients of the DCT representation of the original signal? After representing the signal in a selected basis, feature selection would follow, and then compressive sampling and reconstruction using a neural network, and finally performance on the SVM for recognition.

This page has intentionally been left blank.

Appendix A

Below is a documented part of the jinja code with the help of which we port the svm implementation code to c++.

1. Defining the predict function:

```
{{ dtype|default("int", true) }} predict(float *x) {
   float kernels[{{ sizes.vectors }}] = { 0 };
   {% if sizes.classes > 1 %}
      float decisions[{{ sizes.decisions }}] = { 0 };
      int votes[{{ sizes.classes }}] = { 0 };
   {% endif %}
}
```

The predict function is the primary interface for making predictions with the machine learning model. It accepts a pointer to an array of floats, representing the feature vector of the input data. The function initializes arrays for storing kernel values, decision values, and votes depending on the number of classes in the model.

2. Calculating kernel values:

```
{% for i, w in f.enumerate(arrays.supports) %}
   kernels[{{ i }}] = compute_kernel(x, {% for j, wj in f.enumerate(w) %} {% if j > 0 %},{% endif %} {{ f.round(wj) }} {% endfor %});
{% endfor %}
```

In this part of the code, the kernel values are computed for each support vector using the compute_kernel function. The kernel values are stored in the kernels array.

3. Handling different numbers of classes:

Depending on the number of classes in the model, different prediction mechanisms are used. The Jinja2 template engine generates the appropriate code based on the value of sizes.classes.

```
{% if sizes.classes == 1 %}
   ...
{% elif sizes.classes == 2 %}
   ...
{% else %}
   ...
{% endif %}
```

i) Predicting for a one-class problem:

```
float decision = {{ f.round(arrays.intercepts[0]) }} - ({% for i, coef in f.enumerate(arrays.coefs[0]) %} + kernels[{{ i }}] {% if coef != 1 %}* {{ f.round(coef) }}{% endif %} {% endfor %});
return decision > 0 ? 0 : 1;
```

For a one-class problem, a decision value is calculated using the intercepts and coefficients. The class is determined based on whether the decision value is greater than zero.

ii) Predicting for a two-class problem:

```
float decision = {{ f.round(arrays.intercepts[0]) }}; decision = decision - ({% for i in range(0, sizes.supports[0]) %} + kernels[{{ i }}] * {{ f.round(arrays.coefs[0][i]) }} {% endfor %}); decision = decision - ({% for i in range(sizes.supports[0], sizes.supports[0] + sizes.supports[1]) %} + kernels[{{ i }}] * {{ f.round(arrays.coefs[0][i]) }} {% endfor %});
return decision > 0 ? 0 : 1;
```

For a two-class problem, the decision value is calculated similarly, but with different coefficients and intercepts. The class is determined based on whether the decision value is greater than zero.

iii) Predicting for multi-class problems:

```
{% set helpers = {'ii': 0} %}
{% for i in range(0, sizes.classes) %}
  {% for j in range(i + 1, sizes.classes) %}
    {% set start_i = sizes.supports[:i].sum() %}
    {% set start_j = sizes.supports[:j].sum() %}
    decisions[{{ helpers.ii }}] = {{ f.round(arrays.intercepts[helpers.ii]) }}
    {% for k in range(start_i, start_i + sizes.supports[i]) %}
      {% with coef=arrays.coefs[j-1][k] %}
        {% if coef == 1 %}
          + kernels[{{ k }}]
        {% elif coef == -1 %}
          - kernels[{{ k }}]
        {% elif coef != 0 %}
          + kernels[{{ k }}] * {{ f.round(coef) }}
        {% endif %}
      {% endwith %}
    {% endfor %}
    {% for k in range(start_j, start_j + sizes.supports[j]) %}
      {% with coef=arrays.coefs[i][k] %}
        {% if coef == 1 %}
          + kernels[{{ k }}]
        {% elif coef == -1 %}
          - kernels[{{ k }}]
        {% elif coef %}
          + kernels[{{ k }}] * {{ f.round(coef) }}
        {% endif %}
      {% endwith %}
```

```
        {% endfor %};
        {% if helpers.update({'ii': helpers.ii + 1}) %}{% endif %}
    {% endfor %}
{% endfor %}
{% set helpers = {'ii': 0} %}
{% for i in range(0, sizes.classes) %}
    {% for j in range(i + 1, sizes.classes) %}
        votes[decisions[{{ helpers.ii }}] > 0 ? {{ i }} : {{ j }}] += 1;
        {% if helpers.update({'ii': helpers.ii + 1}) %}{% endif %}
    {% endfor %}
{% endfor %}
int val = votes[0];
int idx = 0;
for (int i = 1; i < {{ sizes.classes }}; i++) {
    if (votes[i] > val) {
        val = votes[i];
        idx = i;
    }
}
```

For multi-class problems, pairwise decisions are made for each pair of classes using the kernel values, intercepts, and coefficients. The code iterates through all possible pairs of classes and computes the decision values, storing them in the decisions array.

After calculating the decision values, the code determines the class with the highest number of votes. The votes array is updated based on the pairwise decisions. The class with the most votes is returned as the final prediction.

This page has intentionally been left blank.

Appendix B

Below is the python source code that implements the main algorithm that finds the optimum kbest and the optimum number m measurements from the initial signal x that forms the compressed vector y.

```python
best_pososto=[]# gia kathe kbest to mikrotero pososto pou petixe to kalitero acc
kalitero_accuracy_gia_kbest=[]
kalitero_pososto_gia_kbest=[]
kalitero_perm_gia_kbest=[]
print("Enarksi epanaliptikis diadikasias evresis kaliterou k best")
for kbest_param in range(100,534,50):
# for kbest_param in range(100,101,100):
    print("-------------------------------------------------------------------------------------------")
    y=data['y']
    limited_X = data.drop(columns = ['y'])
    # generate dataset from leftover features, after Remove collinear features in the dataframe
    selector = SelectKBest(f_classif, k=kbest_param)#
    selector.fit(limited_X, y)
    # # # Get columns to keep and create new dataframe with those only
    cols2 = selector.get_support(indices=False)
    selected_data = limited_X.iloc[:,cols2]
    kbest_evaluation=evaluation_df.iloc[:,cols2]
    # array1=data.columns.values
    selected_data['y']=y
    print('\x1b[6;30;42m' +
'********************************************************************************************************' + '\x1b[0m')
    print("meletame to kbest =",kbest_param)
    print("to shape tou original dataset me kbest = ", selector.k, " einai ", selected_data.shape)
    print("to shape tou evaluation dataset me kbest = ", selector.k, " einai ", kbest_evaluation.shape)
    print('\x1b[6;30;42m' +
'********************************************************************************************************' + '\x1b[0m')
    #pame gia train tou meiomenou dataset
    split = StratifiedShuffleSplit(n_splits = 10,test_size = 0.2,random_state = 1)
    for train_index, test_index in split.split(selected_data,selected_data['y']):
        train_set = selected_data.loc[train_index]
        test_set = selected_data.loc[test_index]
    #dimiourgia train test
    y_train=train_set['y']
    y_test=test_set['y']
    #dimioyrgia test test
    X_train = train_set.drop(columns = ['y'])
    X_test = test_set.drop(columns = ['y'])
    models = []
    models.append(('SVM', SVC()))
    # Test options and evaluation metric
    num_folds = 5
    seed = 7
    scors = ['accuracy']
    results = []
    names = []
    my_res=[]
    for name, model in models:
        print("----------------------------------------------------------")
        for scoring in scors:
            kfold = StratifiedKFold(n_splits=num_folds,shuffle=True, random_state=seed, )
            cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring=scoring)
            results.append(cv_results)
            names.append(name)
            msg = "%s: %s: Cross Validation Score %f (%f)" % (name,scoring, cv_results.mean(), cv_results.std())
            my_res.append(cv_results.mean())
    tuning_data_set=X_train
    c_values = [0.01,0.05,0.08,0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.3, 1.5, 1.7, 2.0]
    gamma=[0.1,0.3,0.7,1,1.2,1.5,1.8,2]
    kernel_values = ['linear', 'poly', 'rbf', 'sigmoid']
    param_grid = dict(C=c_values, kernel=kernel_values,gamma=gamma)
    model = SVC()
    kfold = StratifiedKFold(n_splits=num_folds,shuffle=True, random_state=seed)
    grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring, cv=kfold)
    grid_result = grid.fit(tuning_data_set, y_train)
    means = grid_result.cv_results_['mean_test_score']
    stds = grid_result.cv_results_['std_test_score']
    params = grid_result.cv_results_['params']

model=SVC(C=grid_result.best_params_['C'],gamma=grid_result.best_params_['gamma'],kernel=grid_result.best_params_['kernel
'])
    print(grid_result.best_params_)
    print("kalytero montelo gia k-best =",kbest_param," einai to :" ,model)
    #perform predictions
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    print("Cross Validation Test set accuracy = ",accuracy_score(y_test, predictions))
    #perform prediction on KBEST TRAINED MODEL
    y0=[0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2]
    predictions = model.predict(kbest_evaluation)
    print("Evaluation Dataset accuracy = ",accuracy_score(y0, predictions))
    import warnings
    warnings.filterwarnings('ignore')
    best_perm_list=[]
    best_acc=[]
    # kalitero_pososto=[]
    n=kbest_evaluation.shape[1]
    Psi = dct(np.identity(n)) # Build Psi
    pososta=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.7,0.9]
#      pososta=[0.1]
    y0=[0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2]
    # signals_4_CS_df=kbest_my_signals
    signals=kbest_evaluation.values.tolist()
    print("ta ipo dokimi meiomena exoun shape", kbest_evaluation.shape)
    #     for pososto in pososta:
    times=0#poses fores vrike sinexomeno acc=1
    position=0
    print("-----------------------------------------------------------")
    print("Enarksi meletis pososton")
    #gia kathe pososto
    sinoliko_megisto_pososton=-1 #midenisma gia kathe ena kbest
    while (times<=1 ) and (position<=len(pososta)-1): #thelo an brei dio sinexomena acc=1 na stop...afou ola apo ekei kai
kato 1 tha einai
        print("times=",times)
        pososto=pososta[position]
        position=position+1
        maximum=-11 #to megisto acc kathe posostou
        m = round(n*pososto) # num. random samples
        #enarksi dikias mou gridseach na bro to kalitero perm
        for i in range(1): # edo kano mia grid search na bro t kalitero perm anamesa se 1000 prospathies
            signals_recon=[]
            perm = np.floor(np.random.rand(m) * n).astype(int)
            Theta = Psi[perm,:]       # Measure rows of Psi
            for signal_4_study in signals: #edo pairnei ena ena ta stoixeia tou validation_set
                x=np.array(signal_4_study)
                lasso = Lasso(alpha=0.037) #to eixa brei apo ti meleti
                y = x[perm] #to compressed vector
                lasso.fit(Theta,y) #efarmozo solver
                s_lasso = lasso.coef_ #pairno to S
                s_lasso = np.squeeze(s_lasso)
                xrecon = idct(s_lasso) # reconstruct full signal
                signals_recon.append(xrecon) #ena ena ta anakataskevasmena ta bano se mia lista
            reconstructed_df = pd.DataFrame(signals_recon) #kano ti lista anakataskeuasmenon dataframe
            reconstructed_X=reconstructed_df
            reconstructed_y=y0
            predictions = model.predict(reconstructed_X)
            acc=accuracy_score(y0, predictions) #brisko se poses epese mesa to montelo mou
            if acc > maximum :
                maximum=acc #to kalitero acc sta 1000 gride search
                best_perm=perm #i kaliteri perm list pou dinei to max
                # max_pososto=pososto
            acc=-11 #midenizo kalou kakou
        # kalitero_pososto.append(max_pososto)#macc kathe posostou
        print("===================================================================")
        print("Για ποσοστό  =",pososto," kai kbest =",kbest_param)
        print("max accuracy = " , maximum)
        print("Στοιχεία που απαιτούνται, m= " , m)
        print("best subsample list")
        print(*best_perm,sep=",") #auti edo einai i kaliteri lista
        print("===================================================================")
        best_perm_list.append(best_perm) #bale sti lista best_perm_ti kaliteri perm lista
        best_acc.append(maximum) #best acc gia katheposostο
        if maximum > sinoliko_megisto_pososton:
            sinoliko_megisto_pososton=maximum
            kalitero_pososto=pososto
#         if maximum==1 :
#             times=times+1
    #edo prepei na bro to max accuracy apo to best_acc ALLA to proto pou tha bro giati an px einai to 3
    #megisto acc=1 ola apo ekei kai kato tha einai 1. opote thelo to proto max acc kai tin perm list pou
    #to dinei gia auto to kbest
#     kalitero_accuracy_gia_kbest.append(best_acc)#edo mesa einai to proto kalitero accuracy gia kathe kbest
    kalitero_accuracy_gia_kbest.append(best_acc) #auti i lista periexei kathe posostou to kalitero accuracy
    #diladi sto kalitero_accuracy_gia_kbest[0] tha einai to kalitero accuracy pou petixame sto proto kbest
    #apo kato i antistoixi lista me to kalitero poosοsto pou petixame
    kalitero_pososto_gia_kbest.append(kalitero_pososto)#mia lista me kalitero acc an pososto kai auto gia kathe kbest
    #ka apo kato i antistoixi lista perm pou mou to dinei
#     kalitero_perm_gia_kbest.append(best_perm_list[best_acc.index(max_value)])#edo mesa einai to proto kalitero accuracy
gia kathe kbest
    kalitero_perm_gia_kbest.append(best_perm_list)#edo mesa einai to proto kalitero accuracy gia kathe kbest
    #meta midelizo tis listes gia to epomeno kbest
    best_perm_list=[]
    best_acc=[]
    # kalitero_pososto=[]
    times=0
```

This page has intentionally been left blank.

References

[1] Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, 29(7), 1645-1660

[2] D. L. Donoho, "Compressed sensing," IEEE Transactions on Information Theory, vol. 52, no. 4, pp. 1289-1306, 2006.

[3] E. J. Candès and M. B. Wakin, "An Introduction to Compressive Sampling," in IEEE Signal Processing Magazine, vol. 25, no. 2, pp. 21-30, March 2008, doi: 10.1109/MSP.2007.914731.

[4] Y. -J. Lee, H. -K. Pao, S. -H. Shih, J. -Y. Lin and X. -R. Chen, "Compressed learning for time series classification," 2016 IEEE International Conference on Big Data (Big Data), Washington, DC, USA, 2016, pp. 923-930, doi: 10.1109/BigData.2016.7840688.

[5] X. Liu and S. G. Razul, "Distributed machine learning: A survey," ACM Computing Surveys, vol. 52, no. 2, pp. 1-38, 2019.

[6] P. K. Vica, J. K. Sundararajan, and D. P. Agrawal, "Edge computing: vision and challenges," IEEE Access, vol. 5 , pp. 29,592-29,607, 2017.

[7] X. Chen, S. Park, and M. B. Srivastava, "Efficient distributed machine learning using sketch-based compression," in Proceedings of the International Conference on Machine Learning and Data Mining, pp. 1-16, 2017.

[8] S. Papernot, M. Abadi, Ú. Erlingsson, I. Goodfellow, and N. Carlini, "Deep learning with differential privacy," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 308-318, 2016.

[9] M. E. Hoque and H. K. Nguyen, "Edge-based machine learning using compressive sensing," in Proceedings of the International Conference on Computing, Networking and Communications, pp. 391-397, 2017.

[10] Y. Zhang, Y. Li, and J. Li, "Compressed sensing based distributed machine learning for wireless networks," in Proceedings of the International Conference on Wireless Communications and Signal Processing, pp. 1-6, 2018.

[11] X. Liu and S. G. Razul, "Efficient algorithms for distributed machine learning with compressive sensing," in Proceedings of the International Conference on Machine Learning and Data Mining, pp. 109-124, 2018.

[12] H. Kim and Y. Lee, "Utilizing Energy-Quality Trade-Off for Low-Cost ML-Based Compressive Sensing Reconstruction," 2021 55th Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, CA, USA, 2021, pp. 314-317, doi: 10.1109/IEEECONF53345.2021.9723296.

[13] IoT: Number of Connected Devices Worldwide 2012–2025 | Statista. Available online: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/ (accessed on 21 February 2020).

[14] Vahid Dastjerdi, A.; Buyya, R. Fog Computing: Helping the Internet of Things Realize. IEEE Comput. Soc. 2016, 49, 112–116.

[15] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," in Proceedings of the 4th International Conference on Learning Representations (ICLR 2016), 2016.

[16] Verbraeken Joost, Wolting Matthijs, Katzy Jonathan, Kloppenburg Jeroen, Verbelen Tim, and Rellermeyer Jan S.. 2020. A survey on distributed machine learning. ACM Comput. Surv. 53, 2 (2020), 30:1–30:33

[17] P. Simon, Too Big to Ignore: The Business Case for Big Data. Hoboken, NJ: Wiley, 2013, p. 89. ISBN: 978-1-118-63817-0.

[18] Gunn, S. R. (1998). Support Vector Machines for Classification and Regression. University of Southampton, Technical Report, May.

[19] S. Raschka and V. Mirjalili, Python Machine Learning, 2nd ed. Birmingham, U.K.: Packt, 2017.

[20] N. Cristianini and J. Shawe-Taylor, "Introduction to Support Vector Machines," in Cambridge University Press, 2000. [Online]. Available: http://pzs.dstu.dp.ua/DataMining/svm/bibl/IntroToSVM.pdf.

[21] S. Papadakis, "Machine Learning," lecture notes, HELMEPA, Heraklion, Greece, 2020, postgraduate program "Master of Science in Advanced Manufacturing Systems, Automation and Robotics".

[22] Baldi, Brunak, Chauvin, Andersen and Nielsen, (2000). Assessing the accuracy of prediction algorithms for classification: an overview.

[23] Dhal, P., Azad, C. A comprehensive survey on feature selection in the various fields of machine learning. Appl Intell 52, 4543–4581 (2022). https://doi.org/10.1007/s10489-021-02550-9

[24] I. Iguyon and A. Elisseef, “An introduction to variable and feature selection,” Journal of Machine Learning Research, vol. 3, pp. 1157–1182, 2003.

[25] Murty, M. Narasimha. "Feature selection methods: A review." International journal of software engineering and knowledge engineering 5.3 (1995): 303-327.

[26] I. Iguyon and A. Elisseef, “An introduction to variable and feature selection,” Journal of Machine Learning Research, vol. 3, pp. 1157–1182, 2003.

[27] Talavera, L. (2005). An Evaluation of Filter and Wrapper Methods for Feature Selection in Categorical Clustering. In: Famili, A.F., Kok, J.N., Peña, J.M., Siebes, A., Feelders, A. (eds) Advances in Intelligent Data Analysis VI. IDA 2005. Lecture Notes in Computer Science, vol 3646. Springer, Berlin, Heidelberg.

[28] Candes, E. J., & Tao, T. (2006). Near-optimal signal recovery from random projections: Universal encoding strategies? IEEE Transactions on Information Theory, 52(12), 5406-5425.

[29] Donoho, D. L. (2006). Compressed sensing. IEEE Transactions on Information Theory, 52(4), 1289-1306.

[30] Candes, E. J., & Wakin, M. B. (2008). An introduction to compressive sampling. IEEE Signal Processing Magazine, 25(2), 21-30.

[31] Baraniuk, R. (2007). Compressive sensing. IEEE Signal Processing Magazine, 24(4), 118-121.

[32] Pati, Y. C., Rezaiifar, R., & Krishnaprasad, P. S. (1993). Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In 1993 Conference Record of The Twenty-Seventh Asilomar Conference on Signals, Systems and Computers (Vol. 1, pp. 40-44). IEEE.

[33] Tropp, J. A., & Gilbert, A. C. (2007). Signal recovery from random measurements via orthogonal matching pursuit. IEEE Transactions on Information Theory, 53(12), 4655-4666.

[34] H. J. Landau, "Sampling, data transmission, and the Nyquist rate," in Proceedings of the IEEE, vol. 55, no. 10, pp. 1701-1706, Oct. 1967, doi: 10.1109/PROC.1967.5962.

[35] Kulkarni, A. V. (2004). Frequency Domain Analysis. In Electronic Circuit Design (pp. 95-123). Princeton University Press. Retrieved March 19, 2023, from

[36] Brunton, S., & Kutz, J. (2019). Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control. Cambridge: Cambridge University Press. doi:10.1017/9781108380690

[37] Foucart, S., Rauhut, H.: A Mathematical Introduction to Compressive Sensing. Applied and Numerical Harmonic Analysis. Springer, New York (2013)

[38] Davenport, Mark & Duarte, Marco & Eldar, Yonina & Kutyniok, Gitta. (2012). Introduction to compressed sensing. Preprint. 93. 10.1017/CBO9780511794308.002.

[39] M. A. Iwen and C. V. Spencer, "A note on compressed sensing and the complexity of matrix multiplication," Information Processing Letters, vol. 109, no. 10, pp. 468-471, Apr. 2009, doi: 10.1016/j.ipl.2009.02.014.

[40] Indyk, P. (2010). A tutorial on compressed sensing (or compressive sampling, or linear sketching). MIT. Retrieved from https://people.csail.mit.edu/indyk/CS-tutorial.pdf

[41] K. Okarma, Z. Wei, J. Zhang, Z. Xu, and Y. Liu, "Measurement matrix optimization via mutual coherence minimization for compressively sensed signals reconstruction," Mathematical Problems in Engineering, vol. 2020, Article ID 7979606, Sep. 2020, doi: 10.1155/2020/7979606.

[42] Baraniuk, Richard. “Compressive Sensing [Lecture Notes].” IEEE Signal Processing Magazine 24 (2007): 118-121.

[43] Shirin Jalali, Arian Maleki,From compression to compressed sensing,Applied and Computational Harmonic Analysis,Volume 40, Issue 2,2016,Pages 352-385,ISSN 1063-5203

[44] Elad, M. (2010). Sparse and redundant representation: From theory to applications in signal and image processing. Springer Science & Business Media.

[45] Li, L.; Fang, Y.; Liu, L.; Peng, H.; Kurths, J.; Yang, Y. Overview of Compressed Sensing: Sensing Model, Reconstruction Algorithm, and Its Applications. Appl. Sci. 2020, 10, 5909.

[46] Zonzini, F.; Carbone, A.; Romano, F.; Zauli, M.; De Marchi, L. Machine Learning Meets Compressed Sensing in Vibration-Based Monitoring. Sensors 2022, 22, 2229.

[47] Bao, Y., Tang, Z., & Li, H. (2021). Compressive-sensing data reconstruction for structural health monitoring: a machine-learning approach. Journal of Intelligent Material Systems and Structures, 32(9), 1154-1164

[48] H. Palangi, R. Ward and L. Deng, "Distributed Compressive Sensing: A Deep Learning Approach," in IEEE Transactions on Signal Processing, vol. 64, no. 17, pp. 4504-4518, 1 Sept.1, 2016, doi: 10.1109/TSP.2016.2557301.

[49] Hongpo Zhang, Zhongren Dong, Zhen Wang, Lili Guo, Zongmin Wang,CSNet: A deep learning approach for ECG compressed sensing, Biomedical Signal Processing and Control,Volume 70,2021,103065,ISSN 1746-8094,

[50] D. T. Tran, M. Yamaç, A. Degerli, M. Gabbouj and A. Iosifidis, "Multilinear Compressive Learning," in IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 4, pp. 1512-1524, April 2021, doi: 10.1109/TNNLS.2020.2984831.

[51] Sadeghi, H., & Mohamad, Y. (2016). Sampling rate optimization in inertial measurement unit-based human motion analysis

[52] José Antonio Santoyo-Ramón, Eduardo Casilari, José Manuel Cano-García, "A study of the influence of the sensor sampling frequency on the performance of wearable fall detectors", Volume 193,2022,110945,ISSN 0263-2241

[53] Tova Milo and Amit Somech. 2020. Automating Exploratory Data Analysis via Machine Learning: An Overview. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20). Association for Computing Machinery, New York, NY, USA, 2617–2622.

[54] J. A. Tropp and A. C. Gilbert, "Signal Recovery From Random Measurements Via Orthogonal Matching Pursuit," in IEEE Transactions on Information Theory, vol. 53, no. 12, pp. 4655-4666, Dec. 2007, doi: 10.1109/TIT.2007.909108.