Universidad del País Vasco
Euskal Herriko Unibertsitatea

Informatics Engineering Degree

Software Engineering

Degree's Final Project

Design of classification methods for Brain Computer Interfaces: An application using the OpenViBE software

---

Author

Ibai Díez Oronoz

2015

Abstract

Brain computer Interface (BCI) is a technology which is based on the acquisition of brain signals to be processed and interpreted by a computer, creating a direct communication pathway between the brain and an external device. This type of interfaces stablish a way to interact with the outside world by our thoughts as these interfaces allow transforming them into real actions in our environment. BCIs are often directed at assisting, augmenting, or repairing human cognitive or sensory-motor functions.

This Degree's Final Project is focused on implementing and testing classification algorithms for BCIs because these algorithms are among the most important components of the translation of brain signals. Classification algorithms (CAs) are used in BCIs to predict the human intention from the analysis of the signals, helping the computer to decode the human will from the signals and translate them into commands.

Regarding the implementation and evaluation of these CAs, OpenViBE will be used. OpenViBE is a free software platform which is dedicated to designing, testing and using BCIs and give us all the tools needed to test these algorithms. Moreover, it is a modular and multi-platform software which supports a vast selection of hardware electroencephalography (EEG) devices. In addition, it has different application fields such as medical, multimedia, robotics and all other domains related to BCIs and real-time neuroscience.

Keywords: classification algorithms, BCI, OpenViBE, brain signal analysis.

Contents

List of Figures

List of Tables

PART 1: PROJECT MANAGEMENT

CHAPTER 1.

Objectives of the project

1.1 Introduction

Nowadays, the knowledge of how the brains works is indispensable in our society, because it permits the development of many applications to improve life quality, like providing the capacity of communication and control to people with muscular or neural disabilities.

This fact explains why BCIs [100, 76] were created, because they are used to detect and quantify characteristics of brain signals, translate them into the desired commands and provide a feedback to the user. Moreover, BCIs have also found application in domains such as gaming [75], virtual reality environments [91], and space applications [98].

Brain computer interfaces uses electroencephalography (EEG), magnetoencephalography (MEG) data, etc., to extract the features and then translate them into commands. Usually, classification algorithms (CAs) are used to predict the human intention from the analysis of the signals [19]. This is the reason why there are different types of CAs depending on the BCI paradigm and the type of recorded data.

Therefore, it is required a decoding component to translate brain signals into commands. This explains why signal processing is one of the most important topics of BCIs, where the goal is to interpret brain electrical signals to translate them into the desired command in an effective, accurate and fast way using classification algorithms.

The main purpose of this thesis is to investigate about how can features from EEG be

extracted and translated using the different classification algorithms and paradigms. Another purpose of this thesis is to use this knowledge to develop 4 classification algorithms, including and testing them with OpenViBE software.

This project is focused on contributing to the development of assistance technologies providing communication and control to people with severe muscular or neural disabilities. The goal is designing and implementing several classification algorithms that work with brain signals and classify these signals efficiently and accurately.

To evaluate the results of our implemented classification algorithms we will use the free software OpenViBE which supports the possibility to create scenarios to work with the brain signals, add our algorithms, and test them comfortably.

1.2 Description

Design of classification methods for Brain Computer Interfaces: An application using the OpenViBE software is the name given to this project proposed by Roberto Santana as Degree's Final Project.

The initial idea of this project was to develop classification methods based on evolutionary algorithms and to add them to OpenViBE. During the execution of the project, and due to the characteristics of the available data, we focus on other types of classifiers not yet implemented in OpenViBE.

The brain signal classification problem is a topic of today and much studied by the scientific community. The classification techniques used in this project are not the only ones that can be applied and not necessarily the most optimal. For this reason, we are going to explain different types of techniques that are also used in BCIs in the next sections. This will help to understand which are the reasons because we decided to use them.

1.3 Contents of the thesis

This thesis is divided into three parts, the project management, the technical development, and the validation and conclusions. Moreover, it is divided in chapters which have the following structure:

- The first chapter describes the objectives of the project and the resources used to

work on it. This part also contains the analysis of the background and the feasibility of the project. Finally it presents the initial and real planning, explaining the reasons of the temporary deviation taken.

- The second chapter contains the analysis of the state of the art explaining the basic ideas related to BCIs, focusing on the concepts that help to understand this document.

- The third chapter includes the analysis of the technical aspects of our object of study, in addition to the description of the different classification algorithms.

- The fourth chapter contains an explanation of the implementation of the classification algorithms and how the implementation has been inserted into the OpenViBE software.

- The fifth chapter describes the tests that have been made, the evaluation of the different algorithms and the experiments.

- The sixth and last chapter presents the conclusions of our work and a list of the different lines of investigation that have been left open.

1.4 Background analysis

The background analysis of the project shows the necessity of knowing more about how the brain works. Nowadays, this knowledge is still in its infancy. Hans Berger started to study the human brain, discovering electrical activity on the human brain and developing the electroencephalography (EEG) in 1924. This discovery [113] was the start of the BCIs and opened a world of possibilities and doubts. However, it had to wait until 1970 to start the first investigation of BCI devices in the University of California which was focused on medical objectives like the implantation of neuronal prosthesis to restore hearing, sight or mobility damaged in an human [67]. The first prostheses were developed in 1990 and in the XXI century the investigation was expanded to other areas like video-games [55].

As can be seen, this knowledge has a lot to bring to our society. That is why the technology which works with brain signals is still young and that brings the possibility of carrying out researches and developments to improve life quality. This fact makes this subject of study a relevant area, worth for research.

1.5 Feasibility analysis

Before starting to work with this thesis we made a previous study to see if it was feasible. In spite of being a subject of study that has not a long life we could see that the completion of this project was feasible because of the following reasons:

- **Software availability:** In the research of the available software to develop this project we could find several different platforms [18] as BCI2000[9], OpenViBE [79], TOBI Common Implementation Platform (CIP)[112], BCILAB[10], etc. After the research, we decided to use OpenViBE because it is the one which brings us the possibility of using the algorithms that we will develop and test them in a comfortable way.
- **BCI data availability:** Although the availability of data in the OpenViBE format was limited, we considered it was sufficient to test the CAs.
- **BCI classifiers:** In the previous research of BCI classifiers we could find many different types of classification methods with their mathematical explanations. These mathematical explanations will be useful to develop the classifiers in c++.

1.6 Project initial objectives

The objective of this project is the study of BCIs focusing on the study of the classification algorithms. The goal is to implement different classification algorithms and test them using the OpenViBE software. We will also compare these results with the others algorithms that are implemented in this software.

Some additional objectives will be to implement more classification algorithms, some of them based on optimization methods. Also, to incorporate experiments with other subjects or experiments with other paradigms such as the P300 paradigm for example, which will require to create another scenario.

1.7 Project planning

First of all, a time project estimation was performed. To get this estimation, a list with all the activities to develop was made. Also the times of these activities and the dependencies

between them were estimated. Moreover, an estimation about the necessary resources for the development of the work was made, including the management of the different tasks. Finally, a task monitoring and a correction of the deviations on the first planning was done during the development of the project.

1.7.1 Initial planning

After detailing the project and its requirements, we proceeded to break all work in different tasks. Subsequently, a time breakdown of the activities was performed. In the Gantt chart (Figure 1.1), the Temporary Initial planning is shown. We can observe in this figure the performed tasks that were planned for the development of this project, their duration and dependencies.

Summarizing this table, we first see the task name, start date, end date, and duration in hours. The first idea was to make the necessary initial study in November, October, December and January, to then start with the implementation and test our classification algorithms the first weeks of April. Finally, the writing of the memory would be done in April and May.

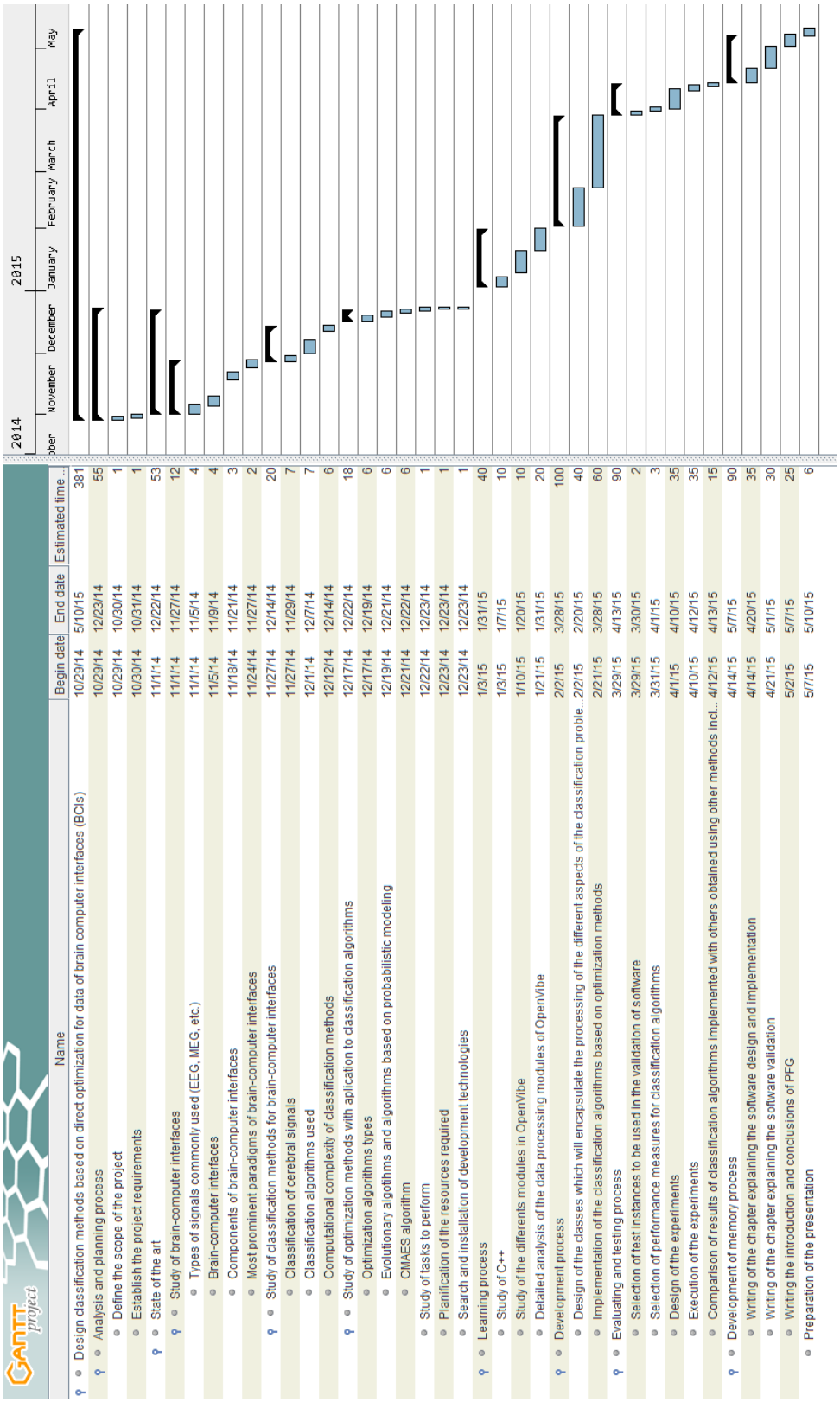

Figure 1.1: Initial Planning

1.7.2 Actual planning

The initial planning (Figure 1.1) was useful as a guide during the development of the project. However there were some difficulties which generated some delays, like inexperience in the programming language and difficulties to understand the new programming environment and personal problems. All these changes are reflected on the new Gantt chart (Figure 1.2).

The phases that have suffered more changes are the implementation, the testing and the writing phases. The implementation phase has been stretched because of some difficulties that appeared during the learning and developing of the different algorithms. Some of these difficulties are, for example, the learning of the particularities of OpenViBE implementation and lack of proper OpenViBE documentation, the limitation of the unknown data format and the difficulty of the implementation of an evolutionary classification algorithm being new in the subject. For this reason, all the following planning changed and new purposes were chosen.

Moreover, we can see in Figure 1.2 that the main purpose of the work changed and two different implementation steps appears. One of them is the implementation of classification algorithms and the other one consists on developing a improvement of one of the previously developed algorithms. Also, the first part of the work was written in January and the rest at the end of the work.

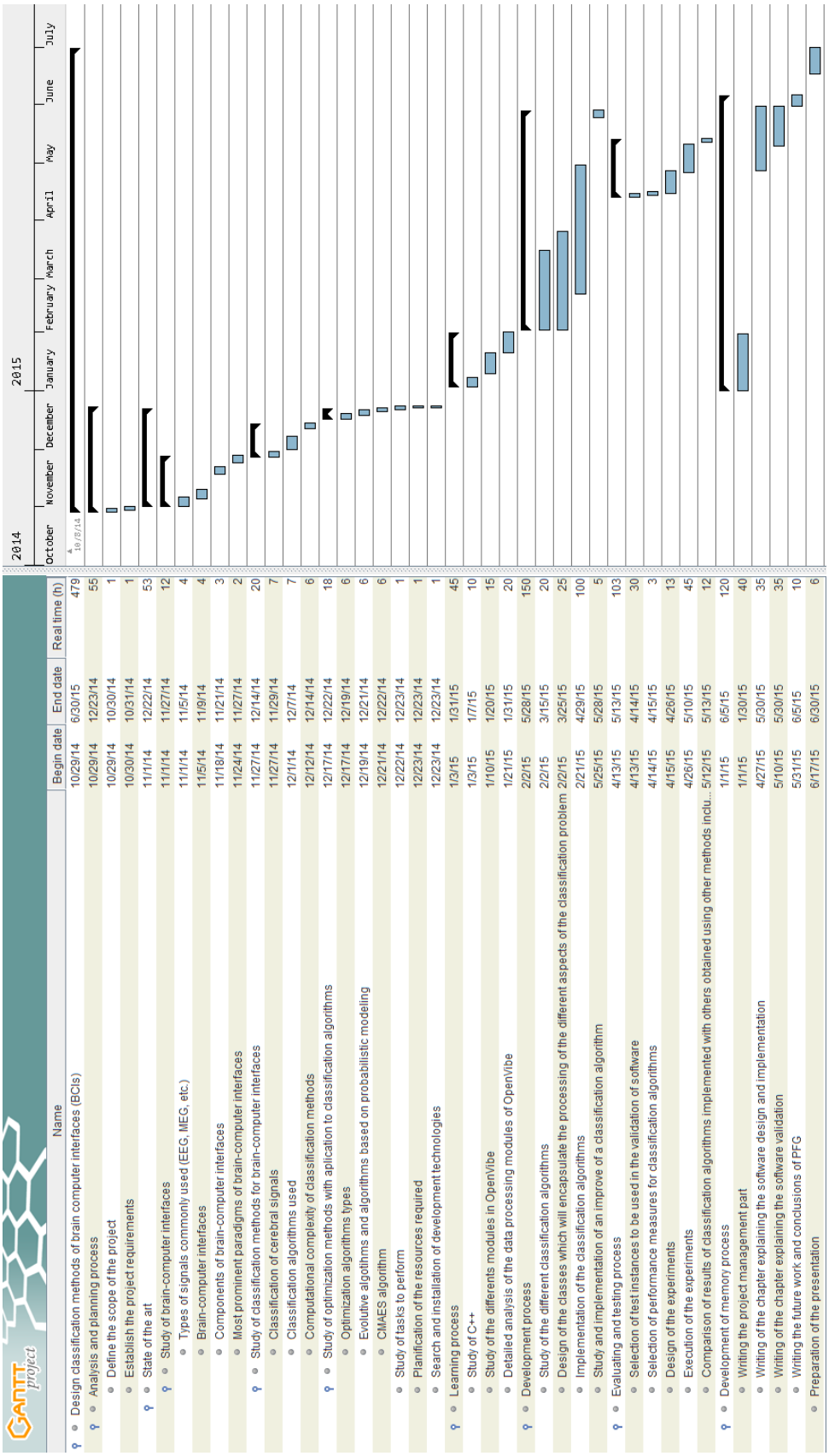

Figure 1.2: Actual Planning

1.8 Resources

This section has the aim of bringing to the reader all the necessary information about the different tools and devices that are used in this work.

1.8.1 Operating system

This work could have been developed on Windows or Linux but Linux was chosen because the compilation and installation steps where faster and more comfortable.

That is why the algorithms and the installation of the OpenViBE software have been developed on Ubuntu operating system [114] which is based on Linux, and is distributed as free and open source software. It is directed to amateur and experienced users because it brings an easy usability and it has not limits for experienced users.

1. Requeriments:
    - 700 MHz processor (about Intel Celeron or better)
    - 512 MiB RAM (system memory)
    - 5 GB of hard-drive space (or USB stick, memory card or external drive but see LiveCD for an alternative approach)
    - VGA capable of 1024x768 screen resolution
    - Either a CD/DVD drive or a USB port for the installer media
    - Internet access is helpful
2. Installation: In case of not having installed Ubuntu is necessary to follow some easy steps which are explained in the annex A.1.

1.8.2 OpenViBE

We used OpenViBE which is a BCI software implemented in C++, which is dedicated to designing, testing and using BCIs [79]. This is why this software has been chosen for the development of this project, because it brings us the capacity to create, implement and test our algorithms in a comfortable way. However, a detailed description of OpenViBE is left for Chapter 4 after we had presented the relevant concepts related to brain signal

analysis and classification methods, since these concepts are required for understanding how the different components of OpenViBE interact.

Installation

In case of not having installed it is necessary to follow some easy installation steps which are explained in the annex A.2.

CHAPTER 2.

State of the art: Classification methods for BCIs

2.1 Introduction

In this section the research area where the project is located is presented to the reader. Below, the main BCI concepts and the different steps that are needed to obtain useful data from the brain signals, will be explained. The way in which classification algorithms are used is also explained in detail.

2.2 Feature extraction

First of all, to be able to process brain signals it is necessary to know about the principles of signal processing that transform the signals to an understandable format representation for a computer. These principles consist in amplifying and digitizing the signals using an analog-to-digital converter [63] which converts the signal into binary numbers. Also signal-processing theory is rooted in Fourier analysis [107], which transforms a time-domain signal into its equivalent frequency-domain representation making easier to evaluate an individual sinusoidal components independently. In addition, a digital filter [90] is commonly used to modify the frequency content of a digital signal by attenuating some frequencies and amplifying others.

The feature extraction process can be separated in three steps: signal conditioning, fea-

ture extraction and conditioning, that reduce the noise of the signal as we can see in the diagram below (Figure 2.1):

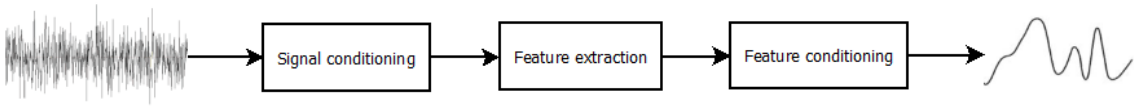

Figure 2.1: Feature extraction workflow

2.2.1 Signal conditioning

Signal conditioning is used to reduce noise and to enhance relevant aspects of the signals. Frequency-range prefiltering is used to eliminate frequencies that lie outside the frequency range of the brain activity most relevant to the application. Also if the signal is digitized at a rate that is higher than the rate required, the signal values are usually decimated and normalized.

We can see in Figure 2.2 the difference between a unconditioned and conditioned signal.

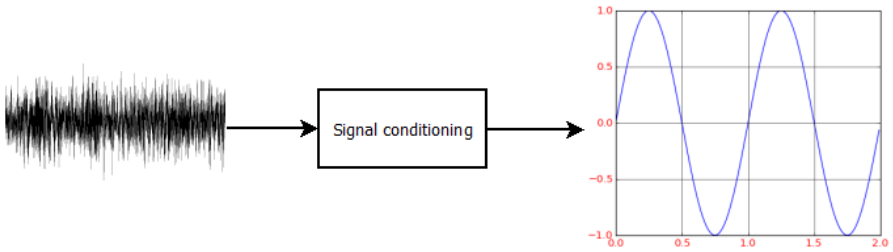

Figure 2.2: A signal before and after being conditioned

There are several factors that must be taken into account to reduce noise and get more clean data. Among these factors are: detection and removal of environmental interferences and biological artefacts, spatial filtering [60], Laplacian filtering [115], interference of eye blinks, etc.

2.2.2 Feature extraction

As suggested by the section title, this process consists in extracting the chosen features from the optimized signal obtained in the previous step.

In this process it is important to correctly select the method that is going to be used because it must be adequate to the specific signal type. That is why the choice of the translation algorithm may affect the choice of the feature-extraction method [62], and vice versa.

Furthermore, it is desirable for the processing to occur in real time. Prior to feature extraction, the incoming signal samples are usually segmented into consecutive, overlapping, sample blocks creating a feature vector with these sample blocks.

Moreover, there are different types of features that are going to be explained below [126]:

- **Time (Temporal Features):** This type of features use to be extracted by Peak-Picking and Integration methods [51]. These methods determine the minimum or maximum value of the signal samples in a specific time block and use these values as the feature for the block. Also this value can be integrated when the responses to the stimulus are known to vary in latency. Moreover it is usually studied the similarity of a response to a predefined template using this template as a filter.

- **Frequency (Spectral) features:** Much brain activity manifests itself as continuous amplitude and frequency-modulate oscillations. To take advantage of this type of feature there are some methods like Band Power which consists on tracking amplitude modulations at a particular frequency to isolate the frequency of interest with a bandpass filter. Also, the Fast Fourier Transform (FFT) [119] method is used to extract this kind of features. The FFT represents the frequency spectrum of a digital signal with a frequency resolution of a sample-rate. Moreover, there is an alternative to FFT method, Autoregressive Modeling (AR) [66] which also computes the frequency spectrum. However, AR assumes that the signal being modeled was generated by passing white noise through an infinite impulse response filter.

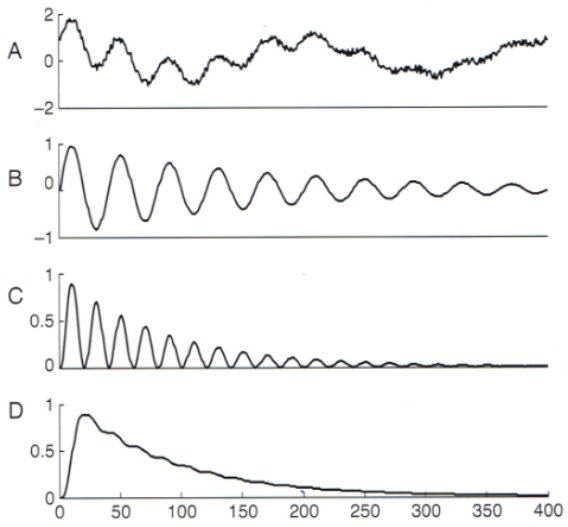

Figure 2.3: The procedure for extracting bandpower from a signal

In Figure 2.3 we can see the procedure for extracting band power from a signal.(A) The original signal. (B) The result after applying a bandpass filter to the data A. (C) The result after squaring the amplitude of B. (D) The final band power result after smoothing C with a low-pass filter.

- **Time-Frequency features (Wavelets):** Wavelet analysis [57] is based on the power spectrum, that is, that the temporal and spectral resolution of the resulting estimates are highly dependent on the selected segment. This method produces a time-frequency representation of the signal and solves the problem that FFT and AR methods only produce one frequency bin that represents these fluctuations at the respective frequency.

- **Similarity features:** Brain activity that occurs synchronously across multiple channel can be relevant to BCI applications. Measures of interaction like the correlation [42], the mutual information [102], and the coherence [49] have been used. The phase locking value (PVL) [110] is a measurement of the level of phase coupling that occurs between two signals occupying the same narrow frequency ranges. Also it is useful to use the coherence, which measures the correlation between the amplitudes of two narrow-band signals. Moreover, features can be defined by measuring the similarity between certain signal features. This is the reason why Mahalanobis distance [27] is used, because it accounts for the covariance among features and is scale invariant.

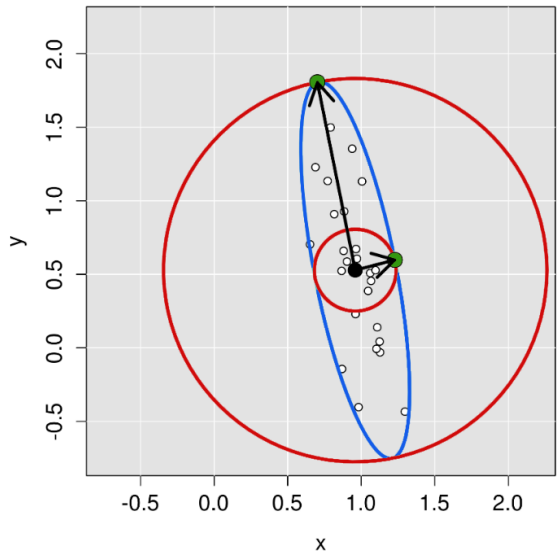

Figure 2.4: The representation of Euclidean and Mahalanobis distances

As we can see in Figure 2.4, the Euclidean distance is circular while the Mahalanobis distance (M) is elliptical. The blue ellipse (M) represents a line of equidistant points that scales data taking account the variance in each direction. The M between green points and the center are equal. However, the Euclidean distance between each green point and the center is different, as shown by the pink circles. In Section 3.1.4 we will introduce in detail these distances that are used by some of the classification approaches described in the thesis.

2.2.3 Feature conditioning (post-processing)

Finally, extracted features can be post-processed using different procedures like normalization, log-normal transforms, feature smoothing and principal component analysis (PCI) and independent component analysis (ICA) methods [126].

2.3 Feature translation

Ideally, these extracted features would be in a form that could directly show the user's intent. However, because the features represent indirect measurements of the user's intent, they must be translated into appropiate device commands which represent that intent. A translation algorithm is a mathematical procedure which accepts the feature vector at a given time instant and processes it to obtain a set of commands that the application device can recognize [64, 118].

2.3.1 Model selection

One of the most important choice to obtain a successful translation algorithm is selecting an appropriate model. This selection is directly related with the requirements of the BCI application. Moreover, there are some general principles that must be followed to choose the right model. We can find different types of model classes according to whether their inputs are discrete or continuous dimensions. Also they can be discriminant functions or regression functions.
Discriminant functions [35] are useful in BCIs that produce simple "yes/no" outputs, whereas regression functions [65] are well suited for BCIs that must provide continuously graded outputs in one or more dimensions. Moreover, model parameters are normally based on a body of previous observations. This method assumes that future data will be similar to the training data. However, biological data in general, normally display considerable data variations. Another issue in the design of BCI translation algorithms [127] concerns whether or not the parameterization of the model algorithm is supervised.

2.3.2 Classification algorithms

A classification algorithm (CA) is a procedure which is used to select a hypothesis from a set of alternatives that best fits with the set of observations. Generally, a classification algorithm needs some components to work with as a Training Set which contains a set of labeled examples that is used to obtain an induced hypotesis. This hypothesis is used with the Testing Set, an unlabeled set of examples, to obtain the score. As we can see in the diagram below (Figure 2.5), it explains how a classification algorithm works with the components that we have explained before.

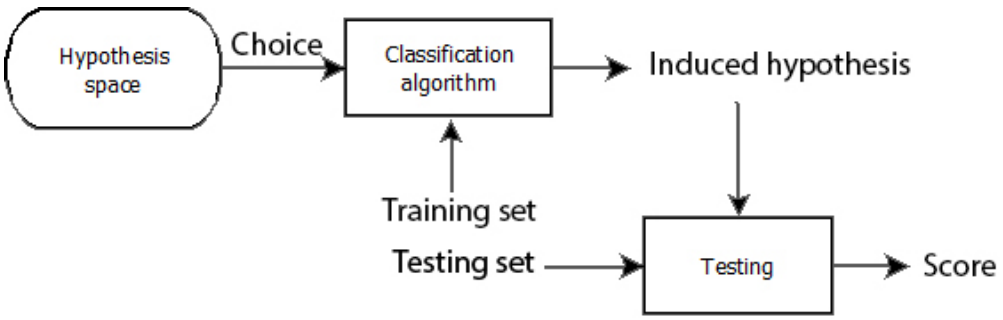

Figure 2.5: Block diagram of a classification algorithm

There are a variety of different classification algorithms and for this reason they can be grouped into five different categories [59]:

1. **Linear classifiers:** Linear classifiers are discriminant algorithms that calculate linear functions to separate classes. In this group we can distinguish two main kinds of linear classifiers, Linear Discriminant Analysis (LDA) [105] and Support Vector Machine (SVM) [109].

   (a) **Linear Discriminant Analysis (LDA):** LDA classifier also known as Fisher's LDA uses vectors to separate data of different classes [31, 48]. For example, for a two-class problem, the class of a feature vector depends on which side of the vector is, as we can see in Figure 2.6 in which the first class (circles) and second class (triangles) are separated by the calculated equation $w_0 + w^T x = 0$. If the calculation of the unlabeled vector with this equation is greater than 0 it is a circle, otherwise, it is classified as a triangle.

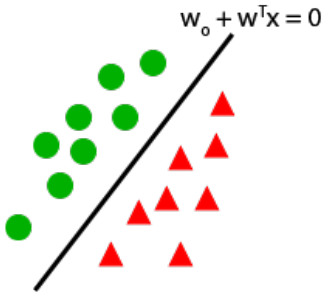

Figure 2.6: A hyperplane which separates two classes: the "circles" an the "triangles"

LDA uses a normal distribution of the data for both classes and the separating vector is obtained by searching the line that maximizes the distance between the two classes means and with the minimum variance [48]. This classifier is simple to use and usually provides good results. For this reason, LDA has been used in a several BCI implementations and is one of those with which we are going to compare our new implemented classifiers.

(b) **Support Vector Machine (SVM):** An SVM also uses a discriminant vector to distinguish between classes as LDAs do [20, 11]. However, the selected vector is the one which maximizes the margins. For example in Figure 2.7, we can see that it uses supporting vectors to obtain the optimal vector to distinguish the features. Also, maximizing the margins we can increase the generalization capabilities [20, 11].

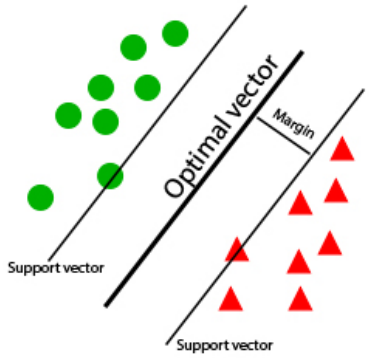

Figure 2.7: SVM finds the optimal hyperplane for generalization

SVMs have some advantages as having good generalization properties thanks to the margin maximization and the regularization term. Also it can be insensitive to over-training and it has some parameters that have to be defined by hand. However, to obtain these advantages we have to sacrifice the speed of execution.

2. **Neural networks:** NNs are, together with linear classifiers, the most used CA in BCIs [46, 2]. The field of NNs studies artificial neurons which produce nonlinear decision boundaries [14]. In this group we can find the Perceptron [97], MultiLayer Perceptron [99], and other neural network architectures [125].

   (a) **MultiLayer Perceptron (MLP):** An MLP contains a multiple layers of neurons, one or more input layers and an output layer [14]. Each of the neuron

inputs are connected with the output of the previous layer's neurons while the neurons of the output layer define the class of the input feature vector.

An MLP without hidden layers is known as a Perceptron and it is equivalent to LDA, for this reason, it has been used for BCIs [123, 23].

(b) **Other Neural Network architectures:** Within this section the most interesting architecture used in BCIs is the Gaussian classifier [29] which exceeds MLP on BCI data, being more efficient and getting better successful results.

Apart from the Gaussian classifier we can find other classifiers as Learning Vector Quantization (LVQ) Neural Network [86], RBF Neural Network [47], Bayesian Logistic Regression Neural Network (BLRNN) Neural Network [41], etc.

3. **Nonlinear Bayesian classifiers:** This group contains two Bayesian classifiers used for BCI: Naive Bayes (NB) classifier [58] and Hidden Markov Model (HMM) [33]. All these classifiers generate nonlinear decision boundaries. Moreover, these classifiers enable to perform more efficient solutions of uncertain samples than discriminative classifiers.

   (a) **Naive Bayes classifiers:** These classifiers [121] are probabilistic classifiers based on applying the Bayes theorem [117] with important independence assumptions between the features. All NB classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. This assumption is called class conditional independence and it is used to simplify the computation involved. This classifier is one of the algorithms developed in this thesis that will be explained in more detail in Section 3.4 .

   (b) **Hidden Markov Model (HMM):** HMM classifiers are dynamic and are used in the field of speech recognition [56]. An HMM provides a probability of observing a feature vector. Also it is combined with algorithms for the classification of time series.

   HMM classifier can be improved to the Input-Output HMM (IOHMM) [21]. The most important advantage of IOHMM is that it can discriminate several classes while HMM needs more operations to obtain the same results.

4. **Nearest neighbor classifiers:** This type of classifiers are relatively simple because they do not involve using a difficult process of class selection. They consist in as-

signing a feature vector to a class according to its nearest neighbor(s). This calculated neighbor can be a feature vector from the training set or a class prototype. Moreover, they are discriminative nonlinear classifiers.

(a) *K* **Nearest Neighbors (KNN):** KNN classifiers [130] aim is to assign the dominant class among its *k* nearest neighbors inside the training set. These nearest neighbors are obtained using a metric distance as Euclidean distance [24], Kolmogorov-Smirnov distance [61], Cosine similarity [73], etc. KNN can be similar to any function which produces nonlinear decision boundaries with a high value of *k* and enough training samples.

KNN algorithms are not very used in the BCIs because they are very sensitive to the curse of dimensionality [25], which made them fail in a variety of scenarios. However, if it is used in BCIs with low dimensional feature vectors KNN can be efficient. KNN classifier is also one of the classifiers developed in this thesis and will be explained in more detail in Section 3.1.

(b) **Nearest CentroID:** This classifier has some similarities with the KNN classifier regarding the search of the nearest neighbour. In this case however, it computes one centroid per class, using the mean of the training data points in that class and then compares these mean values with the testing set, choosing the minimum value to assign the correct class.

As we can see in Figure 2.8 the Rocchio classification is shown, in which a new modified points are calculated. Finally the class is selected computing the distances like the nearest centroID classifier does. For this reason, Rocchios and nearest centroID classifiers are very close.

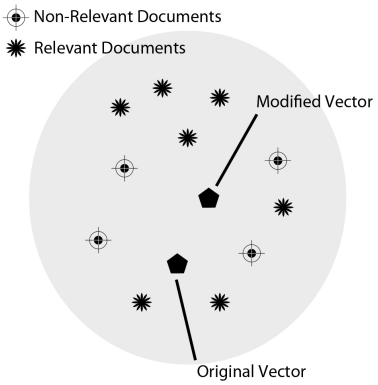

Figure 2.8: Rocchio Classification. Source: Wikipedia

By the fact that nearest centroID classifier obtains good results with most of the experiments, it has found applications in the medical domain, specifically in the classification of tumors [111, 83]. For this reason, this is one of the classifier that has been developed in this project and that is going to be explained in Section 3.3.

5. **Combinations of classifiers:** A new trend is to use some classifiers and aggregate them in different ways. Some of the most commonly used classifier combination strategies are:

   (a) **Boosting:** Boosting consists in using some classifiers in cascade, each classifier focusing on the errors committed by the previous ones [31].

   (b) **Voting:** Whereas using Voting, some classifiers are being used, each of them assigns the input feature vector to a class. The final class will be the majority of them [48].

   (c) **Stacking:** Stacking consists in using some classifiers, each of them classifying the input feature vector. The output of each classifier is given as input to a meta-classifier which makes the final decision [128].

   The most important advantage of this combination is that better results are obtained because the variance and the classification error is reduced.

2.4 BCI paradigms

First of all there is not a definition of BCI paradigms and for this reason we will explain the concept of paradigm and then give our idea of BCI paradigms.

A paradigm is a structure of what direction research should take and how it should be performed. Also, it contains the different theories set or experiments to get this goal. For this reason, a BCI paradigm is the set of theories and experiments that helps to obtain brain data and test them in the computer with the algorithms that are implemented. The most famous paradigms of BCIs are Event related de/synchronization (ERD) [85], Steady State Evoked Potentials (SSEP) [92], P300 [36] and Motor Imagery [16]. This project is focused on P300 and Motor Imagery paradigms which are implemented and used later.

2.4.1 P300

The P300 paradigm [36] is called that way because it is based on the P300 wave. This wave is an evoked potential that can be recorded using ERP as a positive deflection voltage. A positive peak occurs over central and parietal cortex 300 ms after the target stimulus is presented. The presence, magnitude and duration of this signal is used in the measurement of the cognitive function of the decision-making processes. It is usually elicited using the oddball paradigm [8], in which low-probability target items are mixed with high-probability non-target items. The primary advantage of P300 is that it can be parameterized for a new user in a few minutes and requires a minimal user training. For these reasons, P300 based BCIs are the most amenable ones to independent home usage by people with disabilities.

One of the most used methods to obtain brain signals using P300 paradigm consists on flashing on a screen different letters, numbers or symbols and recording the brain signals in the process. This procedure is illustrated in Figure 2.9.

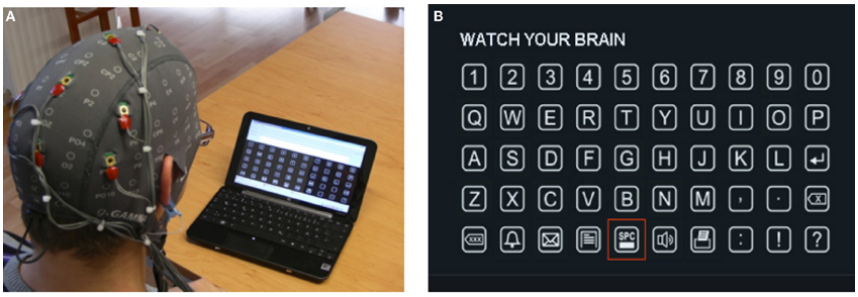

Figure 2.9: P300 paradigm example to obtain brain signals

Moreover, we can observe in Figure 2.10 the graphical representation of the signal recorded and the brain area where the signal originates.

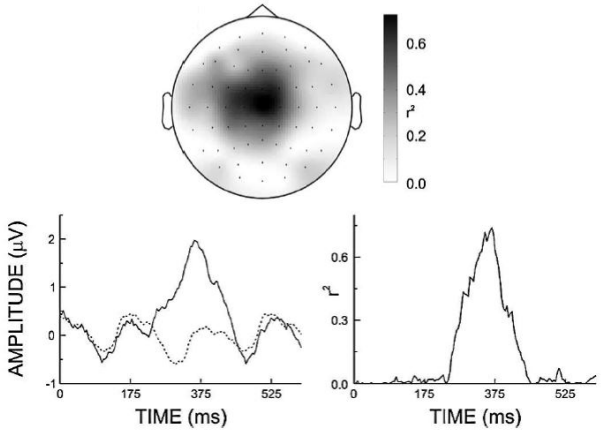

Figure 2.10: A graphical representation of a signal and where the signal originates with P300 paradigm

2.4.2 Motor Imagery

The Motor Imagery [87] paradigm is a mental process by which a subject rehearses or simulates a given action. This mental process has a strong similarity to real executed movements but with a minor intensity. One part of EEG-based BCI is based on the recording and classification of circumscribed and transient EEG changes during different types of motor imagery such as, e.g., imagination of left-hand, right-hand, or foot movement. These changes are detected in the signals and transformed into control signals for external devices.

We can see in Figure 2.11 an example of motor imagery capture in which the subject has to imagine the movement of right and left hand. Also the Figure shows the intensity of the activity in the areas that the brain is using to make this imaginary movement.

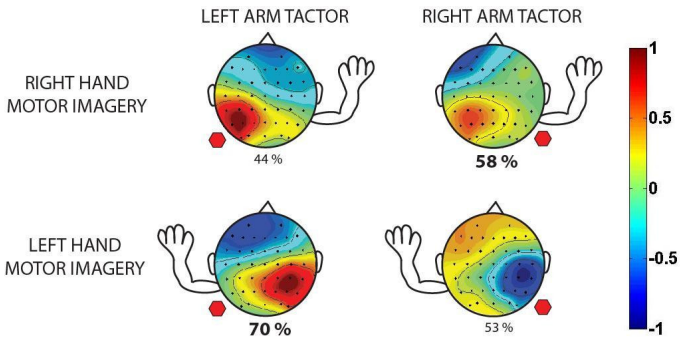

Figure 2.11: Right and left hand motor imagery paradigm example. Source: Traumatrucos

2.5 Optimization algorithms

We also investigate classification methods based on optimization algorithms because they try to improve the accuracy of the traditional classification methods. Some of the most important categories that we can find on optimization algorithms subject are Combinatorial Optimization (CO) [81], Dynamic Programming (DP) [101], Evolutionary Algorithms (EAs) [6], Gradient Methods (GMs) [77] and Stochastic Optimization (SO) [45]. In this work we will talk about EAs, Estimation of distribution algorithm (EDA) [133] and Covariance Matrix Adaptation Evolution Strategy (CMAES) [12], which have been previously applied in BCIs [103, Astigarraga et al., 104].

2.5.1 Evolutionary algorithms

EAs [28] are based on the search of the optimal solutions based on methods inspired on the biological evolution, such as reproduction, mutation, recombination, and selection. They operate on a population of solutions applying the principle of survival of the fittest to obtain better and better approximations to a solution. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness. This process leads to the evolution of populations of individuals that are better suited to their environment than the individuals that they were created from, just as in natural adaptation. In Figure 2.12 we can observe a general workflow of EAs that we have explained before.

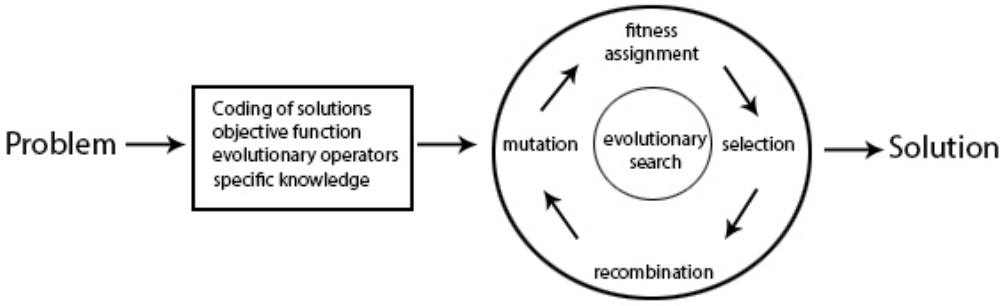

Figure 2.12: General evolutionary algorithm workflow

2.5.2 Estimation of distribution algorithm

EDAs [52, 53] are a relatively new branch of evolutionary algorithms. They replace search operators that are used in EAs with the estimation of the distribution of selected individuals and sampling from this distribution. The aim is to avoid the use of arbitrary operators (mutation, crossover) in favour of explicitly modelling and exploiting the distribution of promising individuals.

The main difference between EDAs and most used EAs is that evolutionary algorithms generate new candidate solutions using an implicit distribution defined by one or more variation operators, whereas EDAs use an explicit probability distribution encoded by a Bayesian network [84], a multivariate normal distribution [106], or another model class. Also the experimental results demonstrate that EDA is better than other EAs in terms of the solution quality and the computational cost [132, 30]. Algorithm 1 presents a pseudocode of the basic EDA steps described in [53].

Figure 2.13: The basic steps of an estimation of distribution algorithm

Inputs : A representation of the solutions
An objective function f
$P_0 \leftarrow$ Generate initial population according to the given representation
$F_0 \leftarrow$ Evaluate each individual x of P_0 using f
$g \leftarrow 1$
while termination criteria not met **do**
$S_g \leftarrow$ Select a subset of P_{g-1} according to F_{g-1} using a selection mechanism
$p_g(x) \leftarrow$ Estimate the probability of solutions in S_g
$Q_g \leftarrow$ Sample $p_g(x)$ according to the given representation
$H_g \leftarrow$ Evaluate Q_g using f
$P_g \leftarrow$ Replace Q_g in P_{g-1} according to F_{g-1} and H_g
$F_g \leftarrow$ Update F_{g-1} according to the solutions in P_g
$g \leftarrow g+1$
end while
Output: The best solution in P_{g-1}

One advantage of EDAs is that they can provide an optimization practitioner with a series of probabilistic models that provide additional information about the problem being solved [108]. This information can be used to create an efficient computational model of the problem.

2.5.3 CMAES algorithm

The CMAES [43] is an evolutionary algorithm that can be applied to difficult non-linear non-convex black-box optimization problems in continuous domain. In an evolution strategy, new candidate solutions are sampled according to a multivariate normal distribution. Recombination amounts to selecting a new mean value for the distribution. Mutation amounts to adding a random vector, a perturbation with zero mean. Pairwise dependencies between the variables in the distribution are represented by a covariance matrix. The covariance matrix adaptation (CMA) is a method to update the covariance matrix of this distribution. This is particularly useful if the function f is ill-conditioned.

Adaptation of the covariance matrix [124] amounts to learning a second order model of the underlying objective function similar to the approximation of the inverse Hessian matrix [13] in the Quasi-Newton method in classical optimization. In contrast to most classical methods, fewer assumptions on the nature of the underlying objective function are made. Only the ranking between candidate solutions is exploited for learning the sample distribution and neither derivatives nor even the function values themselves are required by the method.

Algorithm 2 presents a pseudocode of the basic CMAES steps described in [7, 103].

Figure 2.14: The basic steps of CMAES

1. Generate an initial random solution.

2. The λ offspring at $g + 1$ generation are sampled from a Gaussian distribution using covariance matrix and global step size computed at generation g.

$$x_k{}^{g+1} = z_k,\ z_k = N\ (x_\mu{}^{(g)},\ \sigma^{(g)^2}\ C^{(g)}),\ k = 1,...,\lambda$$

where $x_\mu{}^{(g)} = \sum_{i=1}^{\mu} x_i{}^{(}g)$ with μ being the selected best individuals from the population.

3. The evolution path $P_c{}^{(g+1)}$ is computed as follows:

$$P_c{}^{(g+1)} = (1\text{- } c_c).P_c{}^{(g)} + \sqrt{c_c(2-c_c)}.\frac{\sqrt{\mu}}{\sigma^{(g)}}G$$

$$C^{(g+1)} = (1\text{-}c_{cov}).C^{(}g) + c_{cov}.(\frac{1}{\mu}F + (1-\frac{1}{\mu})\frac{1}{\mu}\sum_{i=1}^{\mu}\frac{1}{\sigma^{(g)^2}}HI)$$

where $G = x_\mu{}^{(g+1)} - x_\mu{}^{(g)}$, $F = (P_c{}^{g+1})P_c{}^{g+1^T})$, $H = (x_i{}^{(g+1)} - x_\mu{}^{(g)})$, and $I = (x_i{}^{(g+1)} - x_\mu{}^{(g)})^T$ The strategy parameter $c_{cov} \in [0, 1]$ determines the rate of change of the covariance matrix C.

4. Adaptation of global step size $\sigma^{(g+1)}$ is based on a conjugate evolution path $P_c{}^{(g+1)}$.

$$P_c{}^{(g+1)} = (1-c_\sigma).P_\sigma{}^{g} + \sqrt{c_\sigma(2-c_c)}B^{(g)}(B^{(g)})^{-1}B^{(g)}\frac{\sqrt{\mu}}{\sigma^{(g)}}G$$

the matrices $B^{(g)}$ and $D^{(g)}$ are obtained through a principal component analysis:

$$C^{(g)} = B^{(g)}(D^{(g)})^2(B^{(g)})^T$$

where the columns of $B^{(g)}$ are the normalized eigen vectors of $C^{(g)}$, and $D^{(g)}$ os the diagonal matrix of the square roots of the eigen values of $C^{(g)}$. The global step size $\sigma^{(g+1)}$ is determined by

$$\sigma^{(g+1)} = \sigma^g exp(\frac{c_\sigma}{d}(\frac{\|P_c{}^{(g+1)}\|}{E(\|N(0,I)\|)}) - 1)$$

5. Repeat Steps 2-4 until a maximum number of function evaluations are reached.

PART 2: TECHNICAL DEVELOPMENT

CHAPTER 3.

Algorithm development

We will explain in this chapter the theoretical concepts that have been used for the project and the particularity of the implementation for each algorithm developed by us.

3.1 k-Nearest Neighbors algorithm

3.1.1 Description

The k-Nearest Neighbors algorithm (KNN) is a method which is considered non-parametric in pattern recognition. It is used for classification and regression but we will focus our explanation on the classification algorithm because it is the one that we have used.

KNN is a type of instance-based learning because it compares new problem instances with instances of the training set which have been stored in memory. For this reason, all the computation is done in the classification and this also makes it a slow algorithm. Moreover, the Nearest Neighbor rule (NN) is considered the simplest form of KNN when $k = 1$. The main advantages of KNN are its simplicity, intuitiveness and competitive classification performance in many domains. Furthermore, there are some improvements of KNN as distance-weighted KNN rule (WKNN) where the close neighbors are weighted to become them heavier according to their distance [32]. Nevertheless, the number of samples is usually too small to obtain a good performance, which usually produces an

important degradation of the classification accuracy, especially in cases with outliers [38] and in problems with a large number of variables [37].

3.1.2 Algorithm

As we have said, in the KNN algorithm all the data computation is made in the classification step. For this reason, it could be implemented directly in one step but to follow the standards of classification algorithm implementation, it is implemented in two steps, the training and the classification.

- **Training:** In this step, we have a training set of labeled feature vectors. We have l classes, $C_1, C_2, ..., C_l$ and each vector is n-dimensional, $X = \{x_1, x_2, ..., x_n\}$. Then it only stores the feature vectors and class labels of the training set in memory to be used in the classification step.

- **Classification:** In this step, the k has been previously chosen by the user. The selection of this parameter is a bit controversial and for this reason it will be discussed in more detail in Subsection 3.1.3. Moreover, a test set $Y = \{y_1, y_2, ..., y_n\}$, which is an unlabeled vector, is used to obtain the nearest k vectors computing the distances between Y and the training set vectors using for that a metric distance method. The developed methods for this classifier will be explained in more detail in Subsection 3.1.4. After obtaining the k nearest vectors, Y is classified taking into account the most frequent class among the k nearest vectors.

Now, we will explain an example of KNN classification with 2 types of labels and with a training set of 2-dimensional vectors to understand how this classifier works. Also, the distance method used in this example is the Euclidean distance. In Figure 3.1 the test sample (green circle) must be classified to the first class of blue squares or to the second class of the red triangles. If $k = 3$ (solid line circle) the class assigned to the test sample is the second class because there are 2 triangles and only 1 square inside the circle. If $k = 5$ (dashed line circle), it is assigned to the first class because there are 3 squares and 2 triangles inside the circle.

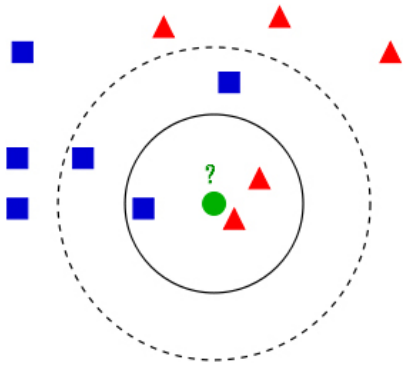

Figure 3.1: Example of KNN classification

3.1.3 Parameter selection

One of the biggest problems of this algorithm that has not been solved yet, is the selection of the neighborhood size k. This factor which can have an important impact on the performance of KNN classifiers [129]. It has been found that the performance of KNN is affected by the selection of the neighborhood size k. If k is very small, the estimation tends to be very bad to the data sparseness and the noisy and ambiguous points. We can try to increase k and take more neighbors but unfortunately a large value of k can easily make the estimate over-smoothing and get worst classification results. Some authors have tried to search for a solution to this problem but the debate is still open [32, 70, 131, 120].

3.1.4 Computation of distances

The definition of the neighborhood relationship between the variables depends on the chosen distance. The most used distance is the Euclidean distance [24], but the Kolmogorov-Smirnov distance [61] and the Cosine similarity [73] are also used. Since the type of distance used may have a strong impact in the behavior of KNN, we have considered the distance as a parameter of the algorithm and implemented three different distances that are described below:

- **Euclidean distance:** It is defined as the square root of the sum of the squares of the differences between the vector points [24]. For example, in two dimensional geometry, the Euclidean distance between two vectors $p = (p_1, p_2)$ and $q = (q_1, q_2)$ can be seen in Formula 3.1:

$$D_e(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2} \tag{3.1}$$

In the same way, we can extend this formula to a n-dimensional geometry as we can see in Equation 3.2:

$$D_e(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_i - q_i)^2 + ... + (p_n - q_n)^2} \tag{3.2}$$

- **Kolmogorov-Smirnov distance:** It is a nonparametric method and also one of the most useful distances to compare two samples and quantify the distance. It can be modified to bring results as a goodness of fit test. However, some studies have found that, it is less powerful for testing than Shapiro-Wilk or Anderson-Darling tests [93].

  As we can see in Equation 3.3, it accumulates the values of the samples and choose the maximum distance of the difference between them:

$$D_{ks}(p,q) = \max_{1 \leq i \leq n} (\sum_{i=1}^{n} p_i - \sum_{i=1}^{n} q_i) \tag{3.3}$$

- **Cosine similarity distance:** It is a measure of similarity between two vectors that measures the cosine of the angle between them.

  The cosine similarity, $cos(\theta)$, is represented using a dot product of the two vectors and the product of the Euclidean distance of each vector as we can see in Equation 3.4.

$$S_c(p,q) = cos(\theta) = \frac{\sum_{i=1}^{n} (p_i * q_i)}{\sqrt{\sum_{i=1}^{n} (p_i)^2} * \sqrt{\sum_{i=1}^{n} (q_i)^2}} \tag{3.4}$$

  Cosine distance is a term often used to measure, however, it is important to say that it is not a good distance metric because it does not have the triangle inequality property and violates the coincidence axiom [74]. As we can see in Equation 3.5 it is just a simple subtraction to obtain the cosine distance:

$$D_c(p,q) = 1 - S_c(p,q) \tag{3.5}$$

3.2 k-Nearest Neighbors Equality algorithm

3.2.1 Description

The k-Nearest Neighbors Equality algorithm (KNNE) [68, 69] is a improvement of the KNN method that we have explained in Section 3.1. For this reason, it has the same characteristics that KNN has, like being a non-parametric method in pattern recognition and being used for classification and regression. The purpose of this algorithm is to decrease the influence of outliers and it includes all classes as candidates in the classification process. This algorithm was developed because there are situations where cases from a class can add noise to the surrounding cases. This is why it tries to solve this problem diminishing the importance of outlier cases in the classification process. Moreover, the improvements of this algorithm are more visible in multi-class problems in which the number of classes bigger than 2.

3.2.2 Algorithm

As we have said, the KNNE algorithm is an extension of the KNN and for this reason all the data computation is made in the classification step. Although it looks that it has a bigger computational cost than the KNN they have the same cost. For this reason, it could be implemented directly in one step but to follow the standards of classification algorithm implementation, it is implemented in two steps, the training and the classification.

- **Training:** In this step, we have a training set of labeled feature vectors. We have l classes, $C_1, C_2, ..., C_l$ and each vector is n-dimensional, $X = \{x_1, x_2, ..., x_n\}$. Then it only stores the feature vectors and class labels of the training set in memory to be used in the classification step.

- **Classification:** In this step, the k has been previously chosen by the user and has been discussed in more detail in Section 3.1.3. Also, one of the three distance methods developed for this algorithm and explained in Section 3.1.4 has been chosen too. Moreover, a test set $Y = \{y_1, y_2, ..., y_n\}$, which is an unlabeled vector, is used

to obtain the nearest k vectors for each class C_l computing the distances between Y and the training set vectors using for that a metric distance method. After obtaining the nearest k vectors, the mean distance of each class is computed and Y is classified taking into account the minimum mean distance between all the class mean distances. Equation 3.6 summarizes this explanation in which with a given k, we search for the k nearest neighbor cases to Y in each class $C_1, C_2, ..., C_l$ and classify the case Y to the class C_l whose k nearest neighbor cases have the minimum mean distance to Y. That is, considering individuals $x_{1m}, ..., x_{im}$ are the k NN for X in class C_l, $l = 1, ..., L$,

$$classify\ x\ in\ C_l\ if\ \min_{l-1..L}\{\frac{1}{k}\sum_{0..k} d(Y, x_{im})\} = \frac{1}{k}\sum_{l} d(Y, x_{im}) \tag{3.6}$$

In Figure 3.2 we present an example of a classification problem with 3 types of labels and with a 2-dimensional training set. We apply the KNNE algorithm to illustrate how it works. The test sample (green circle) must be classified to the first class of blue squares, the second class of the red triangles and the third class of purple circles. In the training step, the mean distance of the $k = 2$ samples of each class is computed. In this example, the first class mean corresponds to the light blue square, the second class to the dark red triangle and the third class to the light purple. The minimum mean distance corresponds to the dark red triangle and in this case the test set is a red triangle. As we can see, all the classes have the same number of opportunities to participate in the classification.

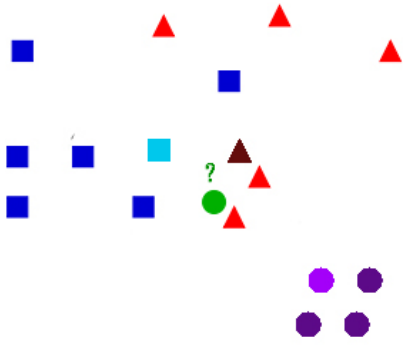

Figure 3.2: Example of KNNE classification

3.3 Nearest CentroID

3.3.1 Description

The Nearest CentroID is an efficient classification algorithm because it has the capacity of implementing nonconvex or disconnected partitions which are formed by a small number of samples.

Nearest CentroID has some similarities with the KNN classifier because both of them search for the nearest neighbours. This classifier uses the mean (centroID) of each class of the training set vectors to obtain the closest to the test set vector and assign the correct class. When it is used to classify text, the nearest centroid classifier is known as the Rocchio classifier because of its similarity to the Rocchio algorithm [71, 50].

Moreover, it has been proved that this algorithm has an error rate that is at most twice the Bayes error rate [39], no matter the distance used in the classification [31].

In addition, this classifier is fast because in the training step it minimizes the quantity of data to work with in the classification step. This is one of the reasons why its extended version has been used in the medical domain, specifically in the classification of tumors [111].

3.3.2 Algorithm

The nearest centroID also has two steps, the training and classification, and it works at follows:

- **Training:** In this first step, the labeled training set $X = \{x_1, x_2, ..., x_n\}$ with l class labels $C_1, C_2, ..., C_l$ is used to compute the per-class means, which are also called "centroids". Equation 3.7 shows the mathematical way in which these centroids are calculated, where C_l is the set of indices of samples that belong to class l .

$$\vec{\mu} = \frac{1}{|C_l|} \sum_{i \in C_l} \vec{x}_i \tag{3.7}$$

- **Classification:** In this step, the closest vector to the test set unlabeled vector $Y = \{y_1, y_2, ..., y_n\}$ is computed. To obtain these closest vectors, previously computed

centroids are used. In this step, nearest centroID and KNN classification algorithms are similar because they compute the distances between two vectors. For this reason, the Euclidean distance [24], Kolmogorov-Smirnov distance [61] and Cosine similarity [73] are developed in this classifier too. These methods have been explained before in KNN Subsection 3.1.4. Finally, the minimum value of this computation assigns the class to this unlabeled vector. Equation 3.8 describes the way the class with the minimum distance is assigned, being $\vec{\mu}$ the centroid of each class, $\vec{y}$ the test set vector and $\hat{L}$ the assigned class.

$$\hat{L} = argmin_{l \in C_l} \|\vec{\mu}_l - \vec{y}\| \tag{3.8}$$

In Figure 3.3 we present an example of a classification problem with 2 types of labels and with a 2-dimensional training set. We apply the Nearest CentroID algorithm to illustrate how it works. The test sample (green circle) must be classified to the first class of blue squares or to the second class of the red triangles. In the training step, the mean (centroID) of each class is computed. In this example, the first class mean corresponds to the black square and the second class to the purple triangle. Now, in the step of classification the distance between the test set and the new means is calculated. The minimum distance corresponds to the purple triangle and in this case the test set is a red triangle.

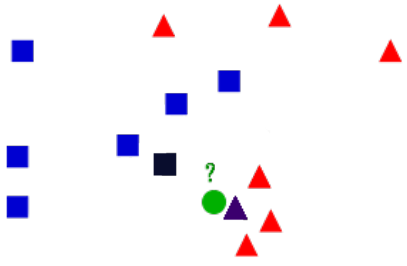

Figure 3.3: Example of Nearest CentroID classification

3.4 Naive Bayes algorithm

3.4.1 Description

Naive Bayes classifiers are a class of simple probabilistic classifiers based on using the Bayes' theorem [117] with independence assumptions between the features. They can predict class membership probabilities like the probability that a given sample belongs to a particular class.

These models assign a class label to a unlabeled vector of feature values. All the naive Bayes classifiers assume that a value of a feature is independent of the value of other feature given a class variable. For example, a fruit might be considered to be an apple if it is red, round and 4" in diameter. This classifier considers each feature to independently contribute to the probability of this fruit being an apple. This assumption is called class conditional independence [26] and it makes the computation involved more simple and for this reason is why they are considered "naive".

Some of the advantages of classifiers with probabilistic outputs are the reject option, changing utility functions, combining models, etc [72].

3.4.2 Bayes' Theorem

We have a sample $X = \{x_1, x_2, ..., x_n\}$ which values are a set of n features. Also we have a hypothesis H, like X belongs to a class C. For classification problems, we have to determine $P(H|X)$, the probability that sample X belongs to class C.

For example, suppose that our samples have features: *age* and *ill*, and the sample X is a 35 year patient that is ill. H is the hypothesis that he has flu. Then $P(H|X)$ is the probability that patient X will has flue given his age and if he is ill.

For our example, $P(H)$ is the probability of H. The next probability $P(H|X)$ is based on more information than the $P(H)$ probability, which is independent of X.

According to Bayes' theorem, the probability we want to compute with Equation 3.9 and these probabilities may be estimated from the given data.

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)} \tag{3.9}$$

3.4.3 Algorithm

The naive Bayes classifier works as explained below in the two common steps, training and classification:

- **Training**: In this step, we have a training set of feature vectors, each with their class labels. There are l classes, $C_1, C_2, ..., C_l$. Each sample is a n-dimensional vector, $X = \{x_1, x_2, ..., x_n\}$. Then we calculate the mean vector of each class and the variances of each vector value of each class.

- **Classification:** Given an unlabeled test set X, in this step the classifier will predict that class X belongs to. It will be selected by the highest probability conditioned on X. This test set belongs to the class C_i if and only if

$$P(C_i|X) > P(C_j|X) \quad for\ 1 \leq j \leq m, j \neq i. \tag{3.10}$$

  Thus we obtain the class that maximizes $P(C_i|X)$. The class C_i for which $P(C_i|X)$ is maximized is the Bayes's theorem 3.11.

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)} \tag{3.11}$$

  $P(X)$ is the same for all classes, then only $P(X|C_i)P(C_i)$ must be maximized. It is assumed that the classes are equally distributed, $P(C_1) = P(C_2) = ... = P(C_l)$, and we will maximize then $P(X|C_i)$.

  It would be very expensive to compute $P(X|C_i)$ with many attributes. In order to improve the computation, the naive assumption of class conditional independence is made. Mathematically it means that

$$P(X|C_i) \approx \prod_{k=1}^{n} P(x_k|C_i). \tag{3.12}$$

  The probabilities $P(x_1|C_i), P(x_2|C_i), ..., P(x_n|C_i)$ can be estimated from the training set. This x_k refers to the value of attribute A_k for sample X.

1. If A_k is categorical, then $P(x_k|C_i)$ is the number of samples of class C_i in the training set having the value x_k for attribute A_k, divided by frequency of $P(C_i, T)$, the number of class C_i in the training set.
2. In our case A_k is continuous-valued, then we assume that the values have a Gaussian distribution with a mean μ and a standard deviation σ defined by Equation 3.13:

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} exp - \frac{(x-\mu)^2}{2\sigma^2} \tag{3.13}$$

The classifier predicts that the class label of X is C_i only if it is the class with maximum probability value.

CHAPTER 4.

OpenViBE software

In this chapter we will explain the most important features of OpenViBE, presenting a general overview of how OpenViBE is implemented, its structure, and describing its functionality. Also, we will focus on the features that we have used.

4.1 General aspects

OpenViBE is a software platform dedicated to designing, testing and using brain-computer interfaces [79]. This software is conceived for real-time neuroscience because it give us the possibility to process brain signals in real-time. Also, it can be used to acquire, filter, process, classify, and visualize the brain signals in real time too. Figure 4.1 shows a diagram of the OpenViBE components.

The most important application fields of OpenViBE are medical [116], multimedia [22], robotics [40, 134] and all other application fields related to BCIs and real-time neurosciences [4]. For example, it can be useful for assistance to disabled people, real-time biofeedback, neurofeedback, real-time diagnosis, virtual reality, video games, etc.

Furthermore, OpenViBE is oriented to users who can be programmers or people not familiar with programming like medical doctors, video game developers, researchers in signal-processing or robotics, etc.

Moreover, OpenViBE is a free and open source software which is compatible with Win-

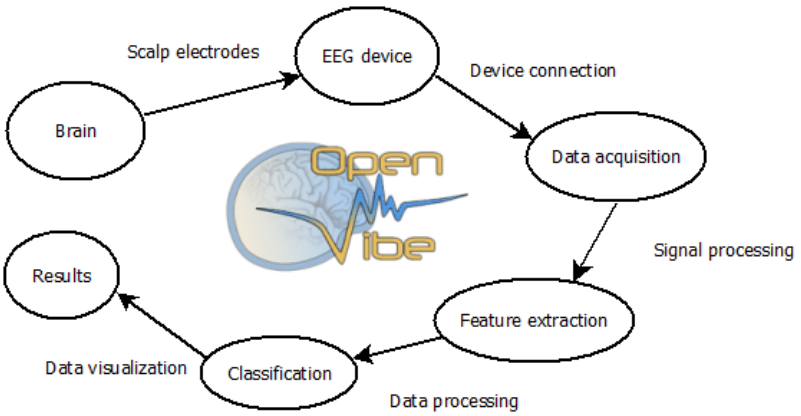

Figure 4.1: Workflow of the OpenViBE software

dows and Linux operating systems and can use a vast selection of hardware EEG devices [80].

4.2 OpenViBE structure and features

As we have mentioned before, OpenViBE is a software that allows to process brain signals and gives us the possibility of acquisition, filtering, processing, classification and visualization of these brain signals. This is the reason why the structure of this software was clearly implemented in different modules.

This modular structure helps to understand and utilize the different possibilities that OpenViBE provides, making the implementation and addition of new different elements, algorithms or boxes, easier. OpenViBE is organized in folders, each folder contains information about a different module. The folder structure is useful to distinguish where the elements must be saved and what files should be changed to rebuild the program with the new code added by the user. OpenViBE is able to process data in real-time and organizes the processing of data in different threads which makes this software more useful.

Furthermore, it is necessary to explain some basic concepts of OpenViBE to make the reading more understandable. The most basic concepts of OpenViBE are the next ones:

1. **Scenario:** An OpenViBE scenario is the set of boxes which are linked between them to get a bigger purpose. For example, if we link different boxes in a scenario we can obtain a maximum signal scenario like in Figure 4.2. This scenario could be tested to evaluate how it works and observe the results. Moreover, the two scenarios

that have been used in this work will be explained in more detail in Section 4.3 and 4.4.

2. **Box:** An OpenViBE box is the most basic element that we can find in this software. It handles a specific task, e.g. classification task. Also we can link different boxes between them to create a sequence of distinct tasks.

3. **Modules:** OpenViBE contains some modules which clearly distinguish the purpose of each box as we will explain after in Section 4.2.1. These modules, each of which encapsulates a phase of data analysis, are divided in data acquisition, signal processing, feature extraction, classification and visualization.

Figure 4.2 represents a scenario with boxes whose goal is to obtain and visualize the maximum signal that is created by sinus oscillator and time signal boxes. This scenario uses the Sinus Oscillator box to produce a 4-channel sinusoidal signal and a 1-channel linear signal using the Time Signal box. Then, the Signal Merger box will merge both inputs to produce a 5-channel signal in which the last channel is the time signal. We have to explain that the In|Out|Set of the Signal Merger box reference to the input, output and settings of this box. To do this work correctly, the 2 sampling boxes must generate a signal with the same sampling frequency and number of samples per buffer. Finally, the Signal Maximum is computed and displayed as shown in the Figure 4.3. Once the time signal is above the maximum amplitude of the generated sinusoids, its samples are naturally chosen as the maximum.

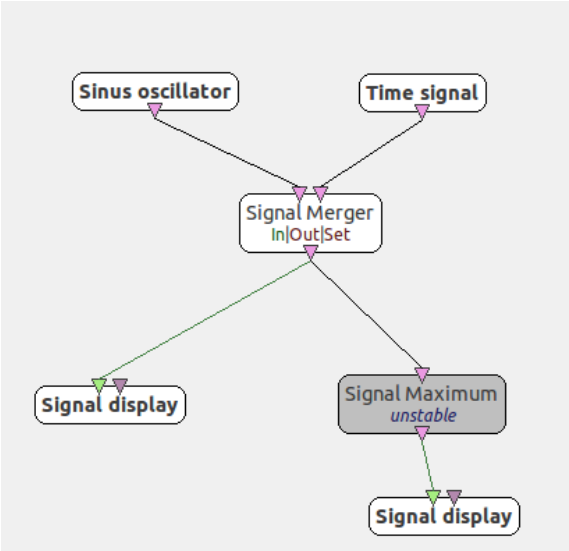

Figure 4.2: Maximum signal scenario

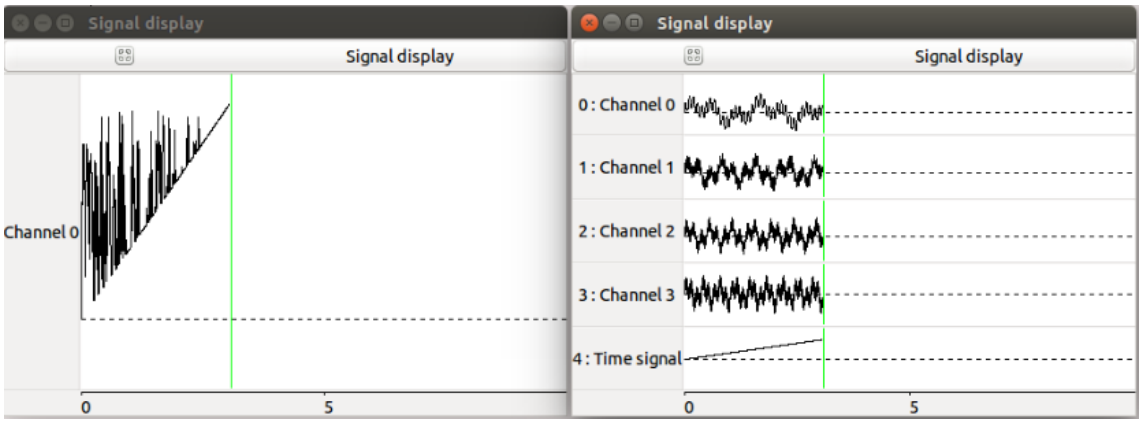

Figure 4.3: Maximum signal scenario display

4.2.1 OpenViBE modules

The modules in which OpenViBE is divided reflect the different steps that have to be made to acquire, filter, process and classify the brain signals as it was explained before. That is why each module encapsulates a phase of data analysis. The most important modules that can be found in OpenViBE are:

- **Data acquisition:** This module acquires brain signal information from EEG devices like helmets with sensors, transforming brain signals into understandable data for the computers. There are different ways to obtain this data into OpenViBE. They can be obtained from local files, online servers, connected devices, etc. Also it has different tools which bring the possibility of creating a connection between different devices to generate a feedback and creating useful prototypes. Moreover, OpenViBE is implemented to be able to read data in real-time.

- **Signal processing:** The data analysis phase of this module is related with the phase of pre-conditioning the brain data before the feature extraction. In this module, the input brain signals are processed by different methods that have been explained in Section 2.2, as filtering and data conditioning, and transform them into a clear and useful data ready for the feature extraction. Some of these methods help to reduce noise for environmental interferences and biological artifacts, filter the signal by spatial or temporal filtering, reduce interferences, etc.

- **Feature extraction:** As the name indicates, this module includes the phase of analysis of the feature extraction. It is responsible of the features extraction method

and this method must be adequate for the classification algorithm that is going to be used for the feature translation. Therefore, the processed data on the signal processing module must be conditioned using the chosen method. These data can be filtered, divided in chunks, modified by applying mathematical methods like log-normal transforms, normalization, etc. This transformations prepare them for the classification method that will be used in the next module.

- **Classification:** This is the module that contains the phase of feature translation that was explained in the state of the art section. This is the most important module in this work because it will contain the different CAs that will be implemented. This does not mean that the other modules will not be modified in case of the algorithm requires a different preprocessing of the data. Now is time to translate these features and to obtain a classification result. In this step OpenViBE gives us different options that must be chosen correctly and will be explained in more detail in Subsection 4.2.2.

- **Visualization:** The visualization module contains the tools to visualize the signals or the classification results that are obtained after these processes. These tools are really useful to validate the results. There are different types of tools for writing in a textfile or just visualize it on the screen.

4.2.2 Classification module

As mentioned before, the classification module encapsulates the phase of feature translation. The main box which we are going to use to test our new CAs is the classifier trainer. The behavior of this box is simple, it collects a number of feature vectors which are labeled depending on the input they arrive on. When a stimulation arrives, a training process is triggered and the box generates a configuration file with the classification. As we said this box needs three inputs:

1. **Stimulations set:** This set is obtained from the data acquisition which reads the input database. Moreover, it does not need any type of transformation before using it in the classifier trainer.

2. **Features for classes:** These two sets of features contain the set of samples obtained after the extraction of features phase divided by class. Then, we have two different

sets, the features for class 1 and 2, which are the two other inputs to make the classifier trainer work.

But this is not all that we need to get a successful classification results, other configuration values must be correctly chosen as we can see in the picture below (Figure 4.4):

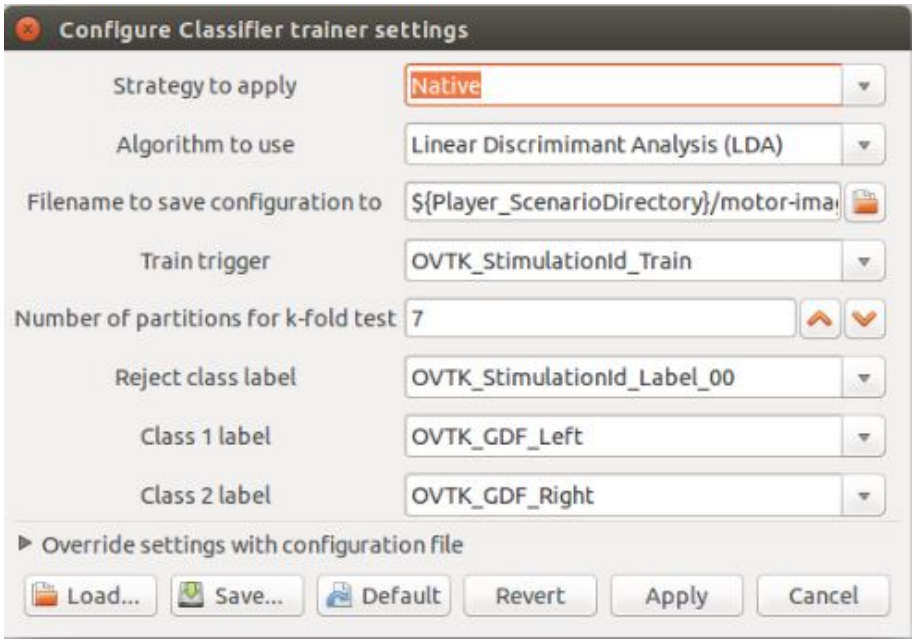

Figure 4.4: Classifier trainer configuration layout

As we can see in the picture, there are some values that must be selected before running our scenario.

- **Strategy to apply:** In this label there are three types of strategies: Native, OneVsAll and OneVsOne strategy [89] that must be adequately chosen by the algorithm that will be used.
- **Algorithm to use:** This is the label in which our developed algorithms will appear and the different algorithms that are implemented in OpenViBE by default are Linear Discriminant Analysis, Shrinkage LDA and its variants, and Support Vector Machine.
- **Filename to save configuration to:** As the name indicates, this label is used to select the filename in which the configuration created by the classifier trainer will be saved, and that will be usable online by the classifier processor box.

- **Train trigger:** This is the stimulation to consider to trigger the training process. In our case, we will use the OVTK_StimulationId_Train trigger because it is the one we need to read the stimulation set input type.

- **Number of partitions for *k*-fold test:** In this one we perform a *k*-fold test, where we should enter something else than 0 or 1 here to get it work. *K*-fold cross validation [94] method generally gives better estimation results than just a native testing. This method consists on dividing the set of feature vectors in a number of partitions and train the classification algorithm on some of the partitions, and its accuracy is tested on the others.

- **Reject class label:** This label is for CAs that support rejection, we can choose a stimulation that reflects the feature vector could not be classified. In our case we will choose the OVTK_StimulationID_Label_00 label which works with the stimulations that we have in our input.

- **Class 1 and 2 labels:** This is the stimulation to send when the classifier algorithm detects a class 1 or 2 feature vector. In our case as we are working with stimulous to right and left imagery movements we will choose OVTK_GDF_Left and OVTK_GDF_Right options.

Finally, after running our scenario, a classified data is generated which can be visualized or used it again with another scenario configuration.

4.3 Motor Imagery BCI with CSP

In this section, we will explain in more detail one scenario that we have used in this work to test our developed classification algorithms. As it has been explained in Section 2.4 there are different types of paradigms that can be used to generate a scenario. In this section, we will explain in detail the scenarios that use the Motor Imagery paradigm which has been described in more detail in Section 2.4.2. OpenViBE provides two different scenarios based on Motor Imagery, in one of them the Common Spatial Pattern (CSP)[15] is used and in the other one the Surface Laplacian spatial filter (SL) [54] is applied. We will use the Motor Imagery with CSP scenario because it obtains better results and data processing is more efficient.

4.3.1 Scenarios

This set of scenarios develop the Graz BCI [88] which is based on motor imagery of the hands. It calculates the spatial filters that efficiently discriminate the signal using CSP, which obtains better performances. For the correct knowledge of these scenarios, in what follows we are going to explain them:

- **Signal monitoring:** This scenario checks the signal quality of the acquisition device and should be always used prior to anything and in the background. It helps to make sure that the EEG acquisition runs correctly.
- **Acquisition:** First of all, some data must be acquired in order to train the classifier that will discriminate the imagined hand movements. We can configure the settings of the LUA stimulator to get a customized database, for example, we can modify the number of trials, timings, etc.
- **Train:** This scenario computes Common Spatial Pattern to produce a spatial filter that maximizes the difference between the signal of the two classes. This filter and its box will be explained in more detail in Section 4.3.2.
- **Classifier:** This scenario trains our implemented classifiers based on the previous acquisition session. Here it is important to say that the signal processing pipeline may be changed according to the type of data acquired. This scenario will be explained in more detail in Section 4.3.3 because it is the one we will primary use to test our classifiers.
- **Online:** This scenario adds real-time feedback to the visualization, using our trained classifier. The signal processing may be tuned again to customize it to our measure. For example, the signal processing pipeline may be changed according to the type of data acquired.
- **Replay:** This scenario is based on the online scenario, but the input signal is coming from a file. In this version, classifier performance tools are used to show the confusion matrix of the classifier and its global performance during the session.

In Figure 4.5 we can see the output using these scenarios. Particularly this picture is taken from one execution of the replay scenario in which the LDA classifier results are shown.

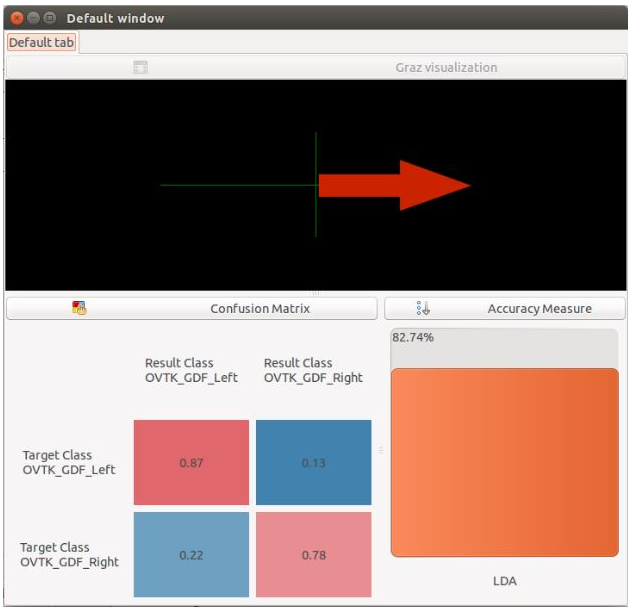

Figure 4.5: The motor imagery BCI with CSP: offline performance computations

4.3.2 Common Spatial Pattern

Common spatial patterns (CSP) [122] method was firstly used for classification of EEG with multiple channels during imagined hand movements. The main idea is to use a linear transform to show the EEG data into a low-dimensional spatial space with a projection matrix. The rows of this matrix are the weights for channels and this transformation can maximize the variance of two class signal matrices. CSP is based on the diagonalization of the covariance matrices of both classes.

Now the details of the algorithm are going to be described [3] using an example in which we classify Left and Right hands movements. X_L and X_R are the preprocessed EEG matrices under two classes (left and right) with $N \times T$, where N is the number of channels and T is the number of samples per channel. The normalized spatial covariance is computed in Equation 4.1, where X^T is the transpose of X and trace(A) computes the sum of the diagonal elements of A.

$$R_L = \frac{X_L {X_L}^T}{trace(X_L {X_L}^T)} \qquad R_R = \frac{X_R {X_R}^T}{trace(X_R {X_R}^T)} \tag{4.1}$$

After computing the spatial covariance, the averaged normalized covariance $\bar{R_L}$ and $\bar{R_R}$ are calculated by averaging all the trials of each group. Now the composite spatial convariance

can be computed as in Equation 4.2, where U_0 is the matrix of eigenvectors and $\sum$ is the diagonal matrix of eigenvalues.

$$R = \bar{R}_L + \bar{R}_R = U_0 \sum {U_0}^T \tag{4.2}$$

The whitening transformation matrix (Equation 4.3):

$$P = \sum -1/2 {U_0}^T \tag{4.3}$$

transforms the average covariance matrices as Equation 4.4:

$$S_L = P\bar{R}_L P^T \qquad S_R = P\bar{R}_R P^T \tag{4.4}$$

S_L and S_R have common eigenvectors and the sum of them for the two matrices will be always one as we can see in Equation 4.5:

$$S_L = U \sum_L U^T \qquad S_R = U \sum_R U^T \qquad \sum_L + \sum_R = I \tag{4.5}$$

The largest eigenvalues for S_L have the smallest eigenvalues for S_R and vice versa. The transformation of whitened EEG onto the eigenvectors is optimal for separating variance in two signal matrices. The projection matrix W of Equation 4.6 is:

$$W = U^T P \tag{4.6}$$

With this projection matrix the original EEG can be transformed into uncorrelated components as Equation 4.7 shows:

$$Z = WX \tag{4.7}$$

Finally Z can be seen as EEG source components but the original EEG X can be reconstructed by Equation 4.8, where W^{-1} is the inverse of the matrix W.

$$X = W^{-1} Z \tag{4.8}$$

4.3.3 Classifier scenario

This scenario will be used to train our classifiers to detect left and right movements. In this scenario the signal processing steps that have been explained before in Chapter 2 can be clearly identified.

- **Data acquisition and preprocessing:** First the *GenericStreamReader* box has to be configured to select the file we have recorded with the *Acquisition* scenario. Moreover, a temporal filter and the CSP Spatial filter are used to preprocess the input data. This can be seen in Figure 4.6:

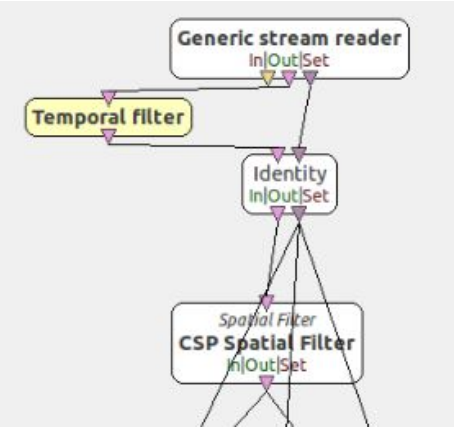

Figure 4.6: Motor-Imagery classifier scenario: Data acquisition and preprocessing

- **Feature extraction:** Now the feature extraction part comes in which the signal is filtered in a large alpha/beta [8 30] Hz range. Then the *StimulationBasedEpoching* box is used to select four seconds of signal half a second after the instruction was shown to the user ($OVTK\_GDF\_Left$ and $OVTK\_GDF\_Right$). After this, the signal is then splitted in blocks of 1 second every 16th second and the logarithmic band power is computed using the two *SimpleDSP* and the *SignalAverage* boxes. Finally the matrices are converted in feature vectors ready for the classification. Figure 4.7 shows all the steps involved:

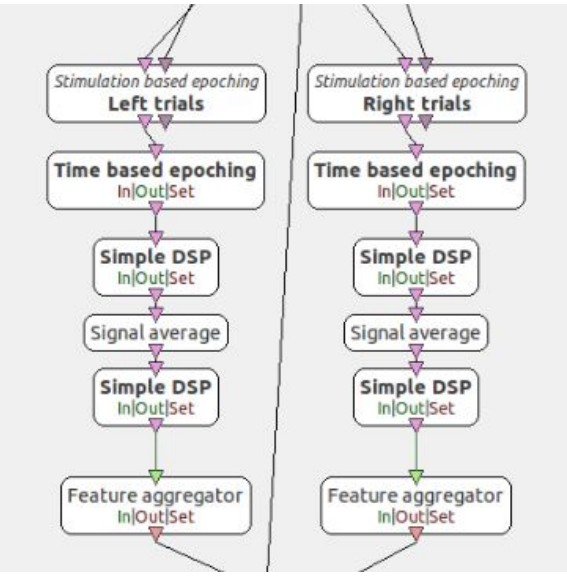

Figure 4.7: Motor-Imagery classifier scenario: Feature extraction

- **Feature translation:** In this last step, the *ClassifierTrainer* box that has been explained in Section 4.2.2 is used to produce a configuration file at the end of the experiment which will be used during the online sessions. When the training process is completed, the box outputs a stimulation *OVTK_StimulationId_TrainCompleted* to stop the scenario through a *PlayerController*.

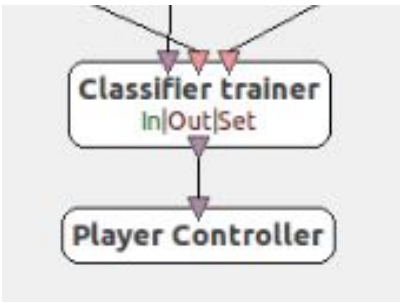

Figure 4.8: Motor-Imagery classifier scenario: Feature translation

At the end of the training, we obtain an estimation of the classifier performance in the console. Generally if the performance is lower than 65% the results are not valid and a new data session must be run to have better results.

4.4 P300 BCI Speller

In this section, we will explain in more detail the different scenarios that use the P300 paradigm that has been described in Section 2.4. OpenViBE provides different scenarios based on P300, in one of them the Magic Card experiment [78] is used, and in the other two the Speller experiment with and without xDAWN spatial filter [96] is used. We will use the P300 Speller scenario because it obtains the most controversial values for the comparison of the classification algorithms.

4.4.1 Scenarios

This set of scenarios develop the P300 Speller BCI which is a good pattern in the brainwaves. The Speller flashes randomly a grid of letters, and the instructed user has to focus on the letters he wants to spell out. After some repetitions, a vote choose which letter the BCI system selects. For the correct knowledge of these scenarios, in what follows we are going to explain them:

- **Acquisition:** First of all, some data must be acquired in order to train the classifier that will classify the P300 brainwaves. The default training session is made of 10 trials in which the instructed user focuses on a particular letter which is marked with blue color. After 12 repetitions we move on to the next letter. These repetitions are 12 flashes of each row and column, so 24 flashes for each letter on the grid. We can configure the settings of the LUA stimulator to get a customized database, for example, we can modify the number of letters, the colors, etc.

- **Classifier:** This scenario trains our developed classification algorithms based on the previous acquisition session that try to discriminate the two classes: P300 or not. This scenario will be explained in more detail in Section 4.4.2 because it is the one we will use to test our classifiers.

- **Online:** This scenario allows the user to spell some text. He can choose to spell the predefined letters or any other letter. However, if the online session is not well-formed it will not work properly. After 12 repetitions, the system displays the letter it has chosen as the best candidate. As in the acquisition scenario we can configure the settings of the LUA stimulator.

In Figure 4.9 we can see how this paradigm works using these scenarios. Particularly this picture is taken from one execution of the classifier scenario in which the flashing letters are shown using the *P300SpellerVisualization* box.

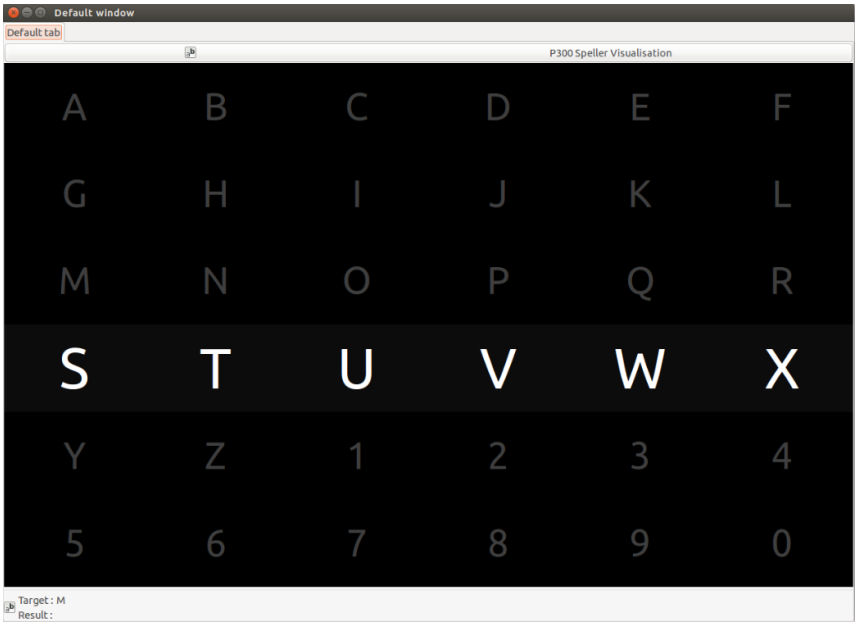

Figure 4.9: The P300 Speller BCI visualization using the classifier scenario

4.4.2 Classifier scenario

This scenario will be used to train our classifiers to detect if it is a P300 or not. In this scenario the signal processing steps that have been explained before in Chapter 2 can be identified.

- **Data acquisition, preprocessing and visualization:** First the *GenericStreamReader* box has to be configured to select the file we have recorded with the *Acquisition* scenario. Moreover, a temporal filter and signal decimation are used to pre-process the input data. Also, the reading of the database can be visualized with the *P300SpellerVisualization* box. This can be seen in Figure 4.10:

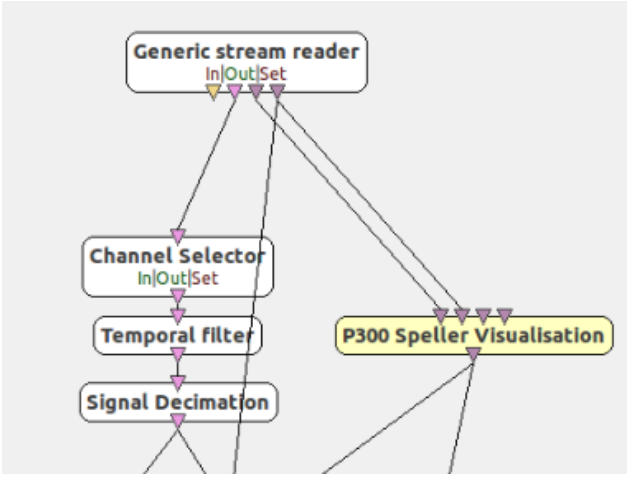

Figure 4.10: P300-Speller classifier scenario: Data acquisition, preprocessing and visualization

- **Feature extraction:** Now the feature extraction part comes in which the signal is divided by class and we obtain its average with *Epochaverage* box. Finally the matrices are converted into feature vectors ready for the classification. Figure 4.11 shows the steps involved:

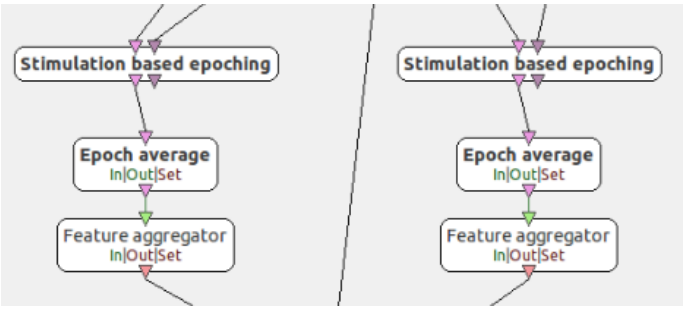

Figure 4.11: P300-Speller classifier scenario: Feature extraction

- **Feature translation:** In this last step, the *ClassifierTrainer* box that has been explained in Section 4.2.2 is used. It uses the stimulus of the database and the extracted signal data.

Figure 4.12: P300-Speller classifier scenario: Feature translation

At the end of the training, we obtain an estimation of the classifier performance in the console. Generally if the performance is lower than 65% the results are not valid and a new data session must be run to have better results.

4.5 OpenViBE data features

In this section we will explain the different ways to obtain data features in OpenViBE that are:

1. One of the ways to obtain data features to use them with our algorithms is using a device which records brain signals and generates our own data. As we do not have a device to generate our own data this option was directly discarded for this work.
2. Other way to get data features is using the different files that are available online in OpenViBE Datasets and then read them with the CSV reader boxes that OpenViBE brings us. But it was discarded too because the CSV reader boxes were unstable boxes and they did not generate a useful data file.
3. The last way to obtain this data is using the default data that OpenViBE provides with the installation. This is the data that we was used due to the impossibility of obtaining new data. This data will be explained in more detail in Section 5.2.

Moreover, this data are composed of different channels that contain the recorded data of each device sensor. For this reason, the configuration of each scenario and data processing depends directly on the device recording type. Also, these channels can be combined into clusters to be able to work more efficiently with the data.

PART 3: VALIDATION AND CONCLUSIONS

CHAPTER 5.

Results

In this chapter, the algorithms that we have developed for this project are going to be analyzed. We have used OpenViBE software to test our algorithms and obtain the necessary data for the analysis of the algorithms.

5.1 Introduction to experiments

The main goal of this research consists in determining if the predictions that our algorithms produce are accurate and competitive to the algorithms that are implemented in OpenViBE. For this reason, the following sections comprise a comparison between the algorithms and also a study of the results obtained by each algorithm.

The results of the algorithms that we will study in the next sections have been obtained using a Ubuntu 14.04.1 LTS. The computer used has an INTEL 3.40GHz processor with 8GB of RAM. Also, the OpenViBE version used is the 0.18.0 stable version.

We will investigate the performance of the classification algorithms we have developed for this project, the default classification method included in OpenViBE, and a modification of this algorithm. We have opted for including several algorithms in the comparison so we could make a better validation of our developed algorithms. Table 5.1 shows the different algorithms that will be used in these experiments with a reference to its author as appears in the code, and the indication *"implemented in this thesis"* otherwise. Some of these algorithms are tested with different variants of their parameters that determine im-

portant differences in the results (e.g. difference distances for KNN, KNNE and Nearest CentroID).

Algorithm	**Author**	**Ref.**
Linear Discriminant Analysis classifier	Y. Renard/F. Lotte	[95]
Linear Discriminant Analysis classifier with CMAES	R. Santana	[103]
Support Vector Machine classifier	L. Bougrain/B. Payan	[34]
k-Nearest Neighbors classifier	I. Díez	
k-Nearest Neighbors Equality classifier	I. Díez	
Nearest CentroID classifier	I. Díez	
Naive Bayesian classifier	I. Díez	

Table 5.1: Classification algorithms implemented in the OpenViBE framework and used for the experiments

5.2 Characteristics of the BCI data used for validation

Now we will explain the characteristics of the data we are going to use with our algorithms.

- **Motor-Imagery data:** This dataset includes records of left and right hand motor imagery. It includes 11 brain channels: C3, C4, Cz, FC3, FC4, C5, C1, C2, C6, CP3 and CP4. In Figure 5.1 we can see which are these brain channel areas. These channels are recorded in common average mode and Cz can be used as a reference if needed. Moreover, the signal is sampled at 512 Hz and was recorded with the Mindmedia NeXus32B amplifier. Also, this data was recorded from the same user and in the same day.

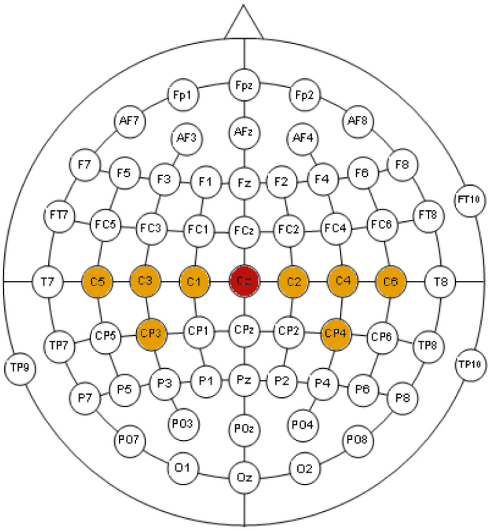

Figure 5.1: Data acquisition brain channel areas for Motor-imagery dataset

As we have explained in Section 4.3.3, these datasets are preprocessed before using them on the classification trainer. These 11 channels are transformed into a two dimensional data. Moreover, this input data is composed by a feature vector set which contains a feature vector and each class label. For this reason, Table 5.2 shows all the characteristics of these feature vector set.

- **P300 data:** This dataset includes records of the P300 speller. It includes 3 brain channels: O1, Oz and O2. In Figure 5.2 we can see which are these brain channel areas. These signals were recorded with the Mindmedia NeXus32BB amplifier.

These signals were amplified and digitized with OpenViBE default rate of 512 Hz and a sample count of 32 per block. Moreover, the training session is made of 10 trials in which the instructed user focuses on a particular letter. After 12 repetitions (12 flashes of each row and column, so 24 flashes for each letter on the grid) we move on to the next letter [44].

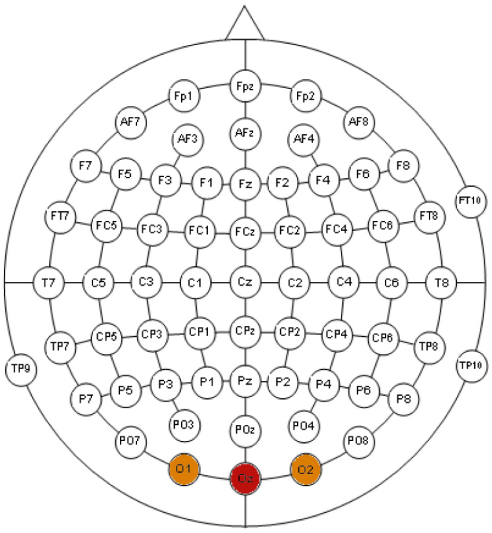

Figure 5.2: Data acquisition brain channel areas for P300 dataset

As we have explained in Section 4.4.2, these data are preprocessed and divided by class before using them on the classification trainer. For this reason, Table 5.2 shows all the characteristics of these feature vector sets.

Paradigm	**No. of feature vectors**	**No. of features**	**No. of Class 1 cases**	**No. of Class 2 cases**
Motor-Imagery	1764	2	931	833
P300	1440	228	240	1200

Table 5.2: Characteristics of the databases used with classification algorithms

5.3 Experiments methodology

Regarding the methodology used for the validation, we have used the cross-validation method to estimate the accuracy of the classifiers. The number of folds has been 5 and the number of repetitions of the cross-validation experiments has been 15. In the case of Motor-Imagery database, for each subset of the cross-validation, the class distribution is approximately 50%. However, in the case of the P300 database the class distribution is approximately 83.33% for Class 2 and 16.66% for Class 1. For this reason and to ensure that our algorithms work properly, we will research the sensitivity and specificity [82, 1] of each classifier.

Feature vectors will be divided in 5 folds. Since the number of cases for the different classes is not the same, the proportions explained before are maintained in each stratified $k-fold$. As we can see in Figure 5.3, the process to generate the subsets for the cross-validation which clearly shows how the previous subsets are used for the training in the following $k-fold$ steps.

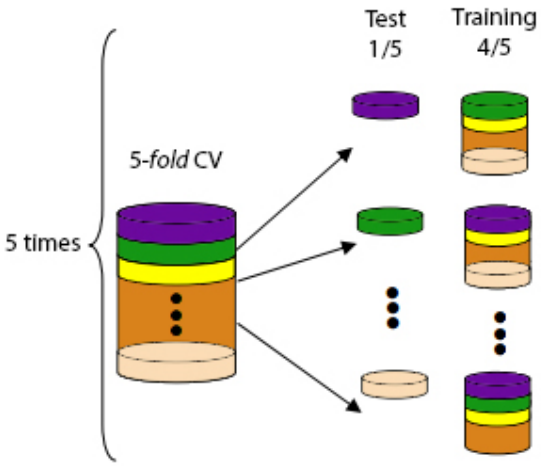

Figure 5.3: Subsets generation process for 5 cross-validation experiments

Moreover, in the case of the k value for KNN and KNNE algorithm, the optimal value, which will be analyzed later, will be computed by $\sqrt{number of features}$.

5.4 Motor-Imagery experiment 1: k-nearest neighbors

In this experiment we will study the particularities of the k-Nearest Neighbors which has the possibility of calculating the distance between the cases in the training set and the test set with three different distance measures. Also, the value of k can be changed and we will investigate this issue later in Section 5.4.3. Moreover, we have used the default dataset explained in Section 5.2 and also we will follow the experimental methodology explained in Section 5.3.

5.4.1 Accuracies

In this section, we will investigate the accuracies obtained using the different distance methods developed for KNN. All these accuracies have been computed with the k value explained in Section 5.3 and without cross-validation, and we can see them in Figure 5.4.

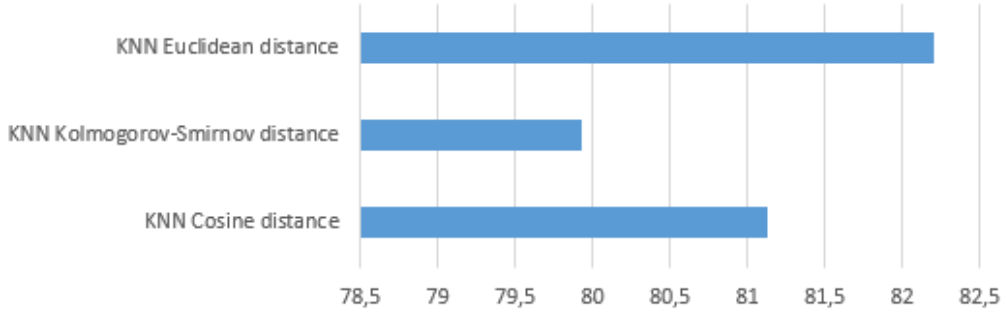

Figure 5.4: KNN accuracies using the three different distances without cross-validation in Motor-Imagery scenario

As it can be appreciated in Figure 5.4, the best result of the KNN algorithm is achieved using the Euclidean distance. With this distance KNN obtains a 82.2% accuracy which is a really good value. Moreover the Cosine distance obtains a 81.12% and the Kolmogorov-Smirnov distance 79.93%, which is the smaller accuracy but still is a really good value. These results are not completely valid because we are training and classifying the algorithm with the same data values. For this reason, we are going to pay more attention to the results obtained with cross-validation because they bring us more valid results.

5.4.2 Performances and variance

Now, we will show the mean accuracy prediction made by the cross-validation along the 15 executions for each distance method. Table 5.3 shows the mean of the variances.

	Euclidean	**Kolmogorov-Smirnov**	**Cosine**
Mean accuracy	80.66	79.37	80.59
Mean variance	1.66	1.14	1.19

Table 5.3: KNN algorithm performances with the different distances in Motor-Imagery scenario (in percentages)

In Table 5.3, we can observe that the mean accuracy and the mean variance of each distance do not have a big difference between them. For this reason, we have used Kruskal Wallis statistical test [17] to determine that the differences between these distances are statistically significant.

First of all, we have compared the Euclidean and Cosine distances. With the adjusted hypothesis $H = 0.097$ and 15 values of each distance, we obtain a $P = 0.7557 > 0.05$. For this fact, we can say that there is not a significant difference between them.

Secondly, we have compared the Kolmogorov-Smirnov distance with the Euclidean and Cosine distances separately and we have obtained similar results for both. With the adjusted hypothesis $H = 21.389$ and 15 values of each distance, we obtain a $P = 3.749\,52\text{E}{-6} < 0.01$. For this fact, we can say that there is a highly significant difference between them.

In conclusion, we could say that the only distance which generates a significant difference in accuracy terms is the Kolmogorov-Smirnov distance comparing it with the other ones. Also we can affirm that the Euclidean and Cosine distance are the best distances to this algorithm.

5.4.3 Study of k value

In this section we will study the accuracies of the KNN algorithm using the Euclidean method changing the value of k. For this study we will start from $k = 7$ because of the way we classify the same values of the training set. With a small k value we obtain a accuracy close to 100% and it is not a real valid solution. In fact, if we use $k = 1$, the algorithm will always find the same testing sample in the training set and the accuracy will be 100%. In Figure 5.5 we can see how the accuracies of the KNN algorithm change with the value of k.

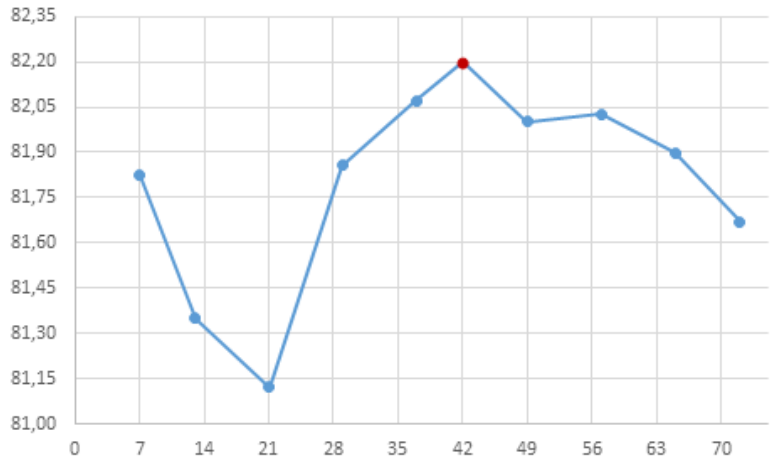

Figure 5.5: KNN accuracies along the variation of k value in Motor-Imagery scenario

As we can see in Figure 5.5, the first 7 values that we have discarded should be the best option for k value but as we have mentioned before these accuracies are not correct. For this reason, we have studied the following values and chosen the optimal that in this case is $k = 42$ computed as $\sqrt{number of features}$. The k values that we have been tested and their accuracies can be seen in Table 5.4.

k	7	13	21	29	37	42	49	57	65	72
Accuracy	81.82	81.34	81.12	81.85	81.86	82.2	82	82.03	81.9	81.67

Table 5.4: The accuracies of KNN for different values of k in the Motor-Imagery scenario (in percentages)

5.4.4 Sensitivity and specificity

In this section, we will investigate the sensitivity and specificity of the KNN algorithm using cross-validation with the three developed distances using the optimal k value. The results of this investigation are shown in Table 5.5, it will help to better evaluate if the algorithm classifies properly the data of the Motor-Imagery database. As mentioned before this data is distributed in a 50.77% and 47.22% for Class 1 and Class 2, respectively.

	Correctly classified	**Wrongly classified**	**Sensitivity**	**Specificity**
Euclidean	80.66	19.34	86.78	77.16
Kolmogorov-Smirnov	79.37	20.63	86.98	72.03
Cosine	80.59	19.41	83.33	78.63

Table 5.5: Sensitivity and specificity of the KNN classifier with cross-validation in Motor-Imagery scenario (in percentages)

the proportion of negatives which are correctly identified as such As we can see in Table 5.5, the sensitivity for the three distances are high values which indicate that these three distances have a high proportion of Classes 1 which are correctly identified as such. In the case of the specificity, we get slightly lower values but they are still high and indicate that they have a high proportion of Classes 2 which are correctly identified as such.

In conclusion, we could say that the KNN is a good option to classify this Motor-Imagery database because it obtains a high sensitivity and specificity with the three distances.

5.4.5 An illustrative example of the algorithm

In this section we will show an example and explain in detail how the algorithm classifies a test data using the dataset of 1764 features. Figure 5.6 presents an example of the classification process for a given unlabeled feature vector. The figure shows the representation of the 1764 samples. There are 931 samples of class 1 (Light blue circles) and 833 samples of class 2 (Orange circles). The KNN algorithm computes the nearest neighbors to the test set (Green circle) selecting in this case 42 neighbors, Class 1 neighbors (Dark blue circles) and Class 2 neighbors (Red circles). The class assigned to the unlabeled case is the class with more number of neighbors, that in this case is Class 1. Moreover, we have added in Table 5.6 the corresponding data for the complete understanding of the example.

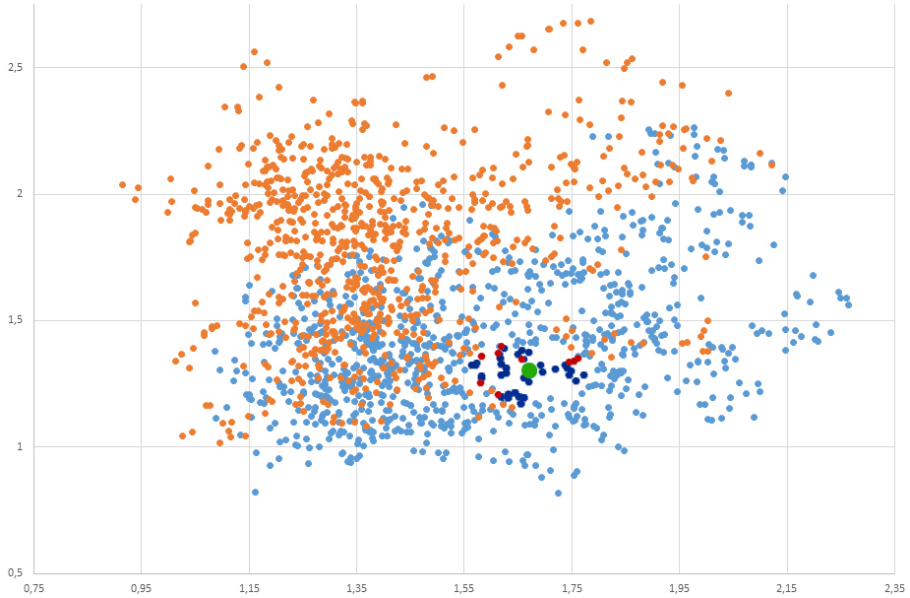

Figure 5.6: KNN real classification example with 1764 data with 2 classes in the Motor-Imagery scenario

Testing data	**Feature 1**	**Feature 2**
	1.67088	1.30327
No. Neighbors Class 1	35	
No. Neighbors Class 2	7	

Table 5.6: The data used in the KNN classification example with 1764 data with 2 classes in the Motor-Imagery scenario

5.5 Motor-Imagery experiment 2: k-Nearest Neighbors Equality

In this experiment we will study the performance of the k-Nearest Neighbors Equality in Motor-Imagery scenario. k-Nearest Neighbors Equality is an improvement of KNN algorithm to solve a problem with outliers. It offers the possibility of computing the distance between the cases in the training set and the test set with three different distance measures. Also, the value of k can be changed and we will investigate this issue later in Section 5.5.3. Moreover, we have used the default dataset explained in Section 5.2 and also we will follow the experimental methodology explained in Section 5.3.

5.5.1 Accuracies

In this section, we will investigate the accuracies obtained using the different distance methods developed for KNNE. All these accuracies have been computed with the k value explained in Section 5.3 and without cross-validation. The accuracies are shown in Figure 5.7.

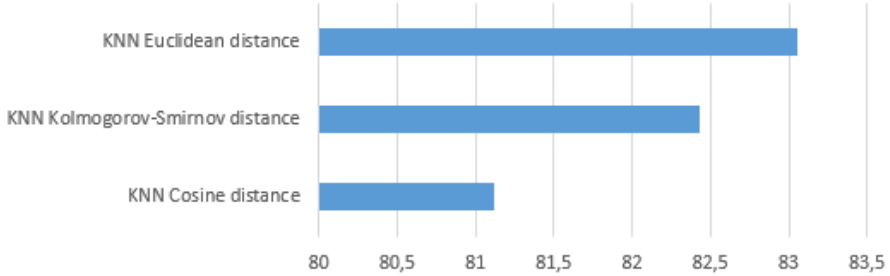

Figure 5.7: KNNE accuracies using the three different distances without cross-validation in Motor-Imagery scenario

As it can be appreciated in Figure 5.7, the best choice for the KNNE algorithm is using the Euclidean distance. With this distance KNNE obtains a 83.05% accuracy which is a high value. Moreover the Cosine distance obtains a 81.12% and the Kolmogorov-Smirnov distance a 82.43% which are very close to the best accuracy obtained with the Euclidean distance. These results are not completely valid because we are training and classifying the algorithm with the same data values. For this reason, we are going to pay more attention to the results obtained with cross-validation because they bring us more valid results.

5.5.2 Performances and variance

We computed the mean accuracy prediction obtained using the cross-validation along the 15 executions for each distance method. Table 5.7 shows the mean variance and the mean of the predicted accuracies of each distance.

	Euclidean	**Kolmogorov-Smirnov**	**Cosine**
Mean accuracy	80.67	79.8	80.63
Mean variance	1.98	2.14	1.98

Table 5.7: KNNE algorithm performances with the different distances in Motor-Imagery scenario (in percentages)

In Table 5.7, we can observe that the mean accuracy and the mean variances of each distance have a slightly difference between them. For this reason, we have used Kruskal Wallis statistical test [17] to determine that these distances are statistically significant or not.

First of all, we have compared the Euclidean and Cosine distances. With the adjusted hypothesis $H = 0.19$ and 15 values of each distance, we obtain a $P = 0.663 > 0.05$. From this result, we can say that there is not a significant difference between them.

Secondly, we have compared the Kolmogorov-Smirnov distance with the Euclidean and Cosine distances separately and we have obtained similar values for both. With the adjusted hypothesis $H = 6.054$ and 15 values of each distance, we obtain a $P = 0.014 < 0.05$. From this result, we can affirm that there is a significant difference between them.

In conclusion, we could say that the only distance which generates a significant difference in the accuracies is the Kolmogorov-Smirnov distance comparing it with the other ones, but the difference in this case is not as high as in the KNN. Also we can affirm that the Euclidean and Cosine distances are the best option for this algorithm.

5.5.3 Study of k value

In this section we will study the accuracies of the KNNE algorithm using the Euclidean distance changing the value of k. For this study we will start from $k = 7$ because of the way we classify the same values of the training set. With a small k value we obtain a accuracy close to 100% and it is not a real valid solution. In fact, if we use $k = 1$, the algorithm will always find the same testing sample in the training set and the accuracy will be 100%. In the Figure 5.8 we can see how the accuracies of the KNNE algorithm change with the number of k.

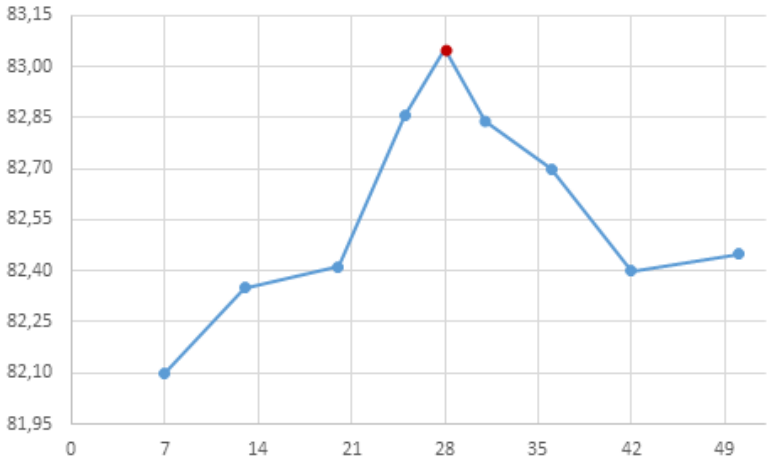

Figure 5.8: KNNE accuracies along the variation of k value in Motor-Imagery scenario

As we can see in Figure 5.8, the first 7 values do not produce real correct accuracies and for this reason we have discarded them. For this reason, we have studied the following values and chose the optimal that is $k = 28$. Moreover, the k values that we have tested and their accuracies can be seen in the table 5.8.

k	7	13	20	25	28	31	36	42	50
Accuracy	82.1	82.35	82.41	82.86	83.05	82.84	82.7	82.4	82.45

Table 5.8: The accuracies of KNNE for different values of k in the Motor-Imagery scenario (in percentages)

5.5.4 Sensitivity and specificity

In this section, we will investigate the sensitivity and specificity of the KNNE algorithm using cross-validation with the three distances implemented and with the optimal k value. The mean results of this investigation are shown in Table 5.9 and it will help to better evaluate if the algorithm classifies properly the data of the Motor-Imagery database. As mentioned before this data is distributed in a 50.77% and 47.22% for Class 1 and Class 2, respectively.

	Correctly classified	**Wrongly classified**	**Sensitivity**	**Specificity**
Euclidean	80.67	19.33	86.67	78.99
Kolmogorov-Smirnov	79.8	20.2	87.95	76.23
Cosine	80.63	19.37	82.17	79.93

Table 5.9: Sensitivity and specificity of the KNNE classifier with cross-validation in Motor-Imagery scenario (in percentages)

As we can see in Table 5.9, the sensitivity for the three distances are high values which indicates that these three distances have a high proportion of Classes 1 which are correctly identified as such. In the case of the specificity, we get slightly lower values but they are still high and indicate that they have high proportion of Classes 2 which are correctly identified as such. In this case, we can also see that this database has not several outliers because the differences between the KNN and KNNE are not significant.

In conclusion, we could say that the KNNE is a good option to classify this Motor-Imagery database and for equally distributed databases.

5.5.5 An illustrative example of the algorithm

In this section we will show an example and explain in detail how the algorithm classifies a testing data using the dataset of 1764 features. Figure 5.9 presents an example of the classification process for a given unlabeled feature vector. The figure shows the representation of the 1764 samples. There are 931 samples of class 1 (Light blue circles) and 833 samples of class 2 (Orange circles). Moreover, the KNNE algorithm computes the nearest neighbors to the testing test (Green circle) selecting in this case 28 neighbors of each class and computing the mean distance of each class, Class 1 neighbors (Dark blue circles) and Class 2 neighbors (Red circles). The class assigned to the unlabeled case is the class with

more number of neighbors, that in this case is Class 1. Moreover, we have added in Table 5.10 the corresponding data for the complete understanding of the example.

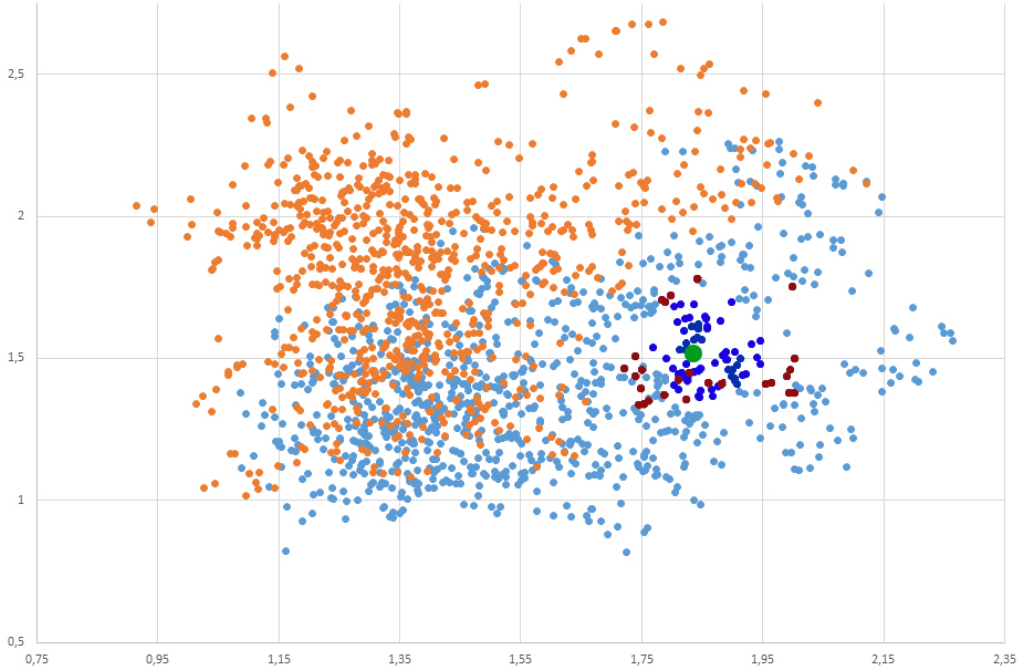

Figure 5.9: KNNE real classification example with 1764 data with 2 classes in the Motor-Imagery scenario

Testing data	**Feature 1**	**Feature 2**
	1.83439	1.51567
Mean distance of Class 1	0.0572112	
Mean distance of Class 2	0.161118	

Table 5.10: The data used in the KNNE classification example with 1764 data with 2 classes in the Motor-Imagery scenario

5.6 Motor-Imagery experiment 3: Nearest CentroID

In this experiment we will study the performance of the Nearest CentroID which can compute the distance between the centroID and the test set using three different distances. We have used the motor-imagery dataset explained in Section 5.2 and also we will follow the experiment methodology explained in Section 5.3.

5.6.1 Accuracies

The accuracies obtained using the different distance methods developed for this algorithm and without cross-validation are shown in the Figure 5.10.

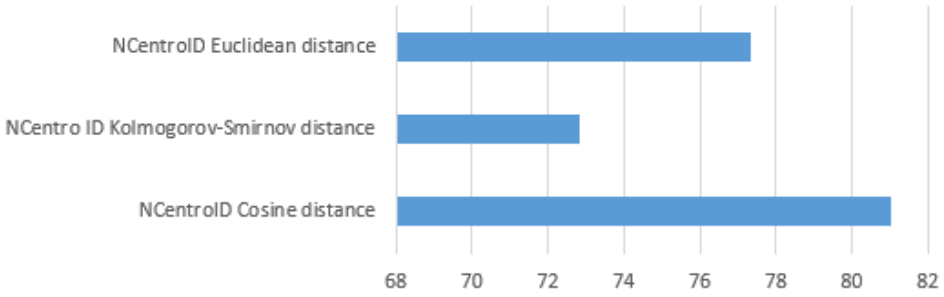

Figure 5.10: Nearest centroID accuracies using the distance methods in Motor-Imagery scenario without cross-validation

As we can see in Figure 5.10, the best result of the Nearest CentroID algorithm is obtained using the Cosine distance. It obtains an accuracy of 81.01% on this dataset. The Euclidean distance obtains a 77.32% and the Kolmogorov-Smirnov distance a 72.85%. An important result is that there is a higher variability due to the distance measure for Nearest CentroID than for KNN. Although they get a lower accuracy than other algorithms they are not discarded because a 65% accuracy is considered a good value in this scope. These results are not completely valid because we are training and classifying the algorithm with the same data values. For this reason, we are going to pay more attention to the results obtained with cross-validation because they bring us more valid results.

5.6.2 Performances and variance

Coming up next, we will investigate for each distance the mean accuracy prediction obtained using the cross-validation along the 15 executions and the mean of the variance. The data can be seen in Table 5.11.

	Euclidean	**Kolmogorov-Smirnov**	**Cosine**
Mean accuracy	77.45	72.84	81.05
Mean variance	1.23	1.22	1.35

Table 5.11: Nearest CentroID algorithm performances with the different distances in Motor-Imagery scenario (in pertentages)

In Table 5.11, we can observe that the mean accuracies and the mean variances of each distance have a bigger difference than in the KNN and KNNE case. However, we have used Kruskal Wallis statistical test [17] to determine whether the results of the classifiers for these distances are statistically significant.

In this case we have compared the three algorithm with Kruskal Wallis and with the adjusted hypothesis $H = 41.8$ and 15 values of each distance, we obtain a $P = 8.38\text{E}{-10} < 0.01$. From this fact, we can affirm that there are significant differences between the distances.

In conclusion, we can determine that the Cosine distance is the most accurate distance for this algorithm and for this dataset. This does not mean that the Cosine will always be the best choice for this algorithm because with other datasets and signal type this algorithm will obtain different results.

5.6.3 Sensitivity and specificity

In this section, we will investigate the sensitivity and specificity of the Nearest CentroID algorithm using cross-validation with the three developed distances and with the optimal k value. This investigation, which the obtained data are shown in Table 5.12, will determine whether the algorithm classifies properly the data of the Motor-Imagery database which is distributed in a 50.77% and 47.22% for Class 1 and Class 2, respectively.

	Correctly classified	**Wrongly classified**	**Sensitivity**	**Specificity**
Euclidean	77.45	22.55	80.75	73.47
Kolmogorov-Smirnov	72.84	27.16	77.12	68.15
Cosine	81.05	18.95	81.53	80.41

Table 5.12: Sensitivity and specificity of the Nearest CentroID classifier with cross-validation in the Motor-Imagery scenario (in percentages)

As we can see in Table 5.12, the sensitivity of the three distances are good values which indicates that these three distances have a high proportion of Classes 1 which are correctly identified as such in most cases. In the case of the specificity we get slightly lower values but they still are high and indicate that they have high proportion of Classes 2 that are correctly identified as such, except in the Kolmogorov-Smirnov case that gets a worst value.

In conclusion, we could say that the Nearest CentroID is a good option to classify this Motor-Imagery database but in this work we will present better options.

5.6.4 An illustrative example of the algorithm

In this section we will show an example and explain in detail how the Nearest CentroID algorithm classifies a test set using the dataset of 1764 features. Figure 5.11 presents an example of the classification process for a given unlabeled feature vector. The figure shows the representation of the 1764 samples. There are 931 samples of class 1 (Light blue circles) and 833 samples of class 2 (Orange circles). The nearest centroID algorithm computes the centroIDs for each class. Then, with the centroID of class 1 (Dark blue circle) and the centroID of class 2 (Red circle), we compute the distance between the unlabeled case (Green circle) and each centroID using in this case the Cosine distance. As we can see in Figure 5.11, the distance D_1 is smaller than the distance D_2 and for this reason we can conclude that the test sample will be assigned a Class 1 (Light blue

circle). Moreover, we have added in Table 5.13 the corresponding data for the complete understanding of the example.

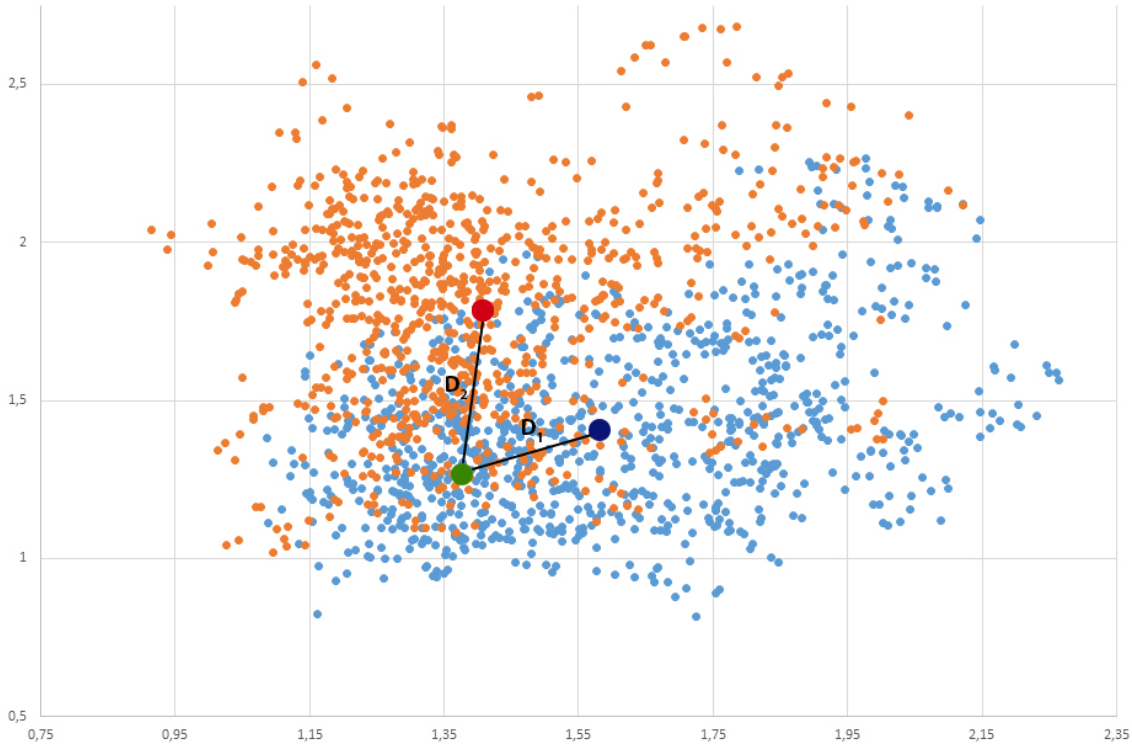

Figure 5.11: Nearest centroID real classification example with 1764 data with 2 classes in the Motor-Imagery scenario

	Feature 1	**Feature 2**
Testing data	1.37526	1.26575
Class 1 mean	1.58056	1.40684
Class 2 mean	1.40364	1.79298
Distance 1	0.00013846	
Distance 2	0.0131975	

Table 5.13: The data used in the Nearest centroID classification example with 1764 data with 2 classes in the Motor-Imagery scenario

5.7 Motor-Imagery experiment 4: Naive Bayes

In this experiment we will study the characteristics of the Naive Bayes classifier which calculates the probability of the occurrence of the testing sample in each class. We have used the default dataset explained in Section 5.2 and also we will follow the experiment methodology explained in Section 5.3.

5.7.1 Performances and variance

This section presents the mean accuracy prediction obtained using the cross-validation along the 15 executions and also the mean variance. For this database, the Naive Bayes has obtained a 80.37% accuracy with 1.76% variance which we consider a really good value. These results are not completely valid because we are training and classifying the algorithm with the same data values. For this reason, we are going to pay more attention to the results obtained with cross-validation because they bring us more valid results.

5.7.2 Sensitivity and specificity

In this section, we will investigate the sensitivity and specificity of the Naive Bayes algorithm using cross-validation. This investigation, which the obtained data are shown in Table 5.14, will determine whether the algorithm classifies properly the data of the Motor-Imagery database which is distributed in a 50.77% and 47.22% for Class 1 and Class 2 respectively.

Correctly classified	Wrongly classified	Sensitivity	Specificity
80.37	19.94	85.05	75.63

Table 5.14: Sensitivity and specificity of the Naive Bayes classifier with cross-validation in the Motor-Imagery scenario (in percentages)

As we can see in Table 5.14, the sensitivity accuracies values are high which indicates that it has a high proportion of Classes 1 which are correctly identified as such in most cases. In the case of the specificity has a high proportion of Classes 2 which are correctly identified as such, making a few mistakes.

In conclusion, we could say that the Naive Bayes classifier is a good option to classify this Motor-Imagery database.

5.7.3 An illustrative example of the algorithm

In this section we will explain in detail how the algorithm classifies a test set using the dataset of 1764 features. Figure 5.12 presents an example of the classification process for a given unlabeled feature vector. The figure shows the representation of the 1764 samples. This dataset contains 931 samples of class 1 (Light blue circles) and 833 samples of class 2 (Orange circles). Moreover, the Naive Bayesian algorithm computes the probability of the occurrence of the test set for each class using the mean value and the variance. Also we can see a black circle with surrounds a nearest area. This area is only used to make more understandable and see how the probability of the testing set of being a type class 1 (P_1) is bigger than being a type class 2 (P_2). For this reason, the test set is of the type class 1 and we can see it in the table 5.15 which contains the obtained data for this example.

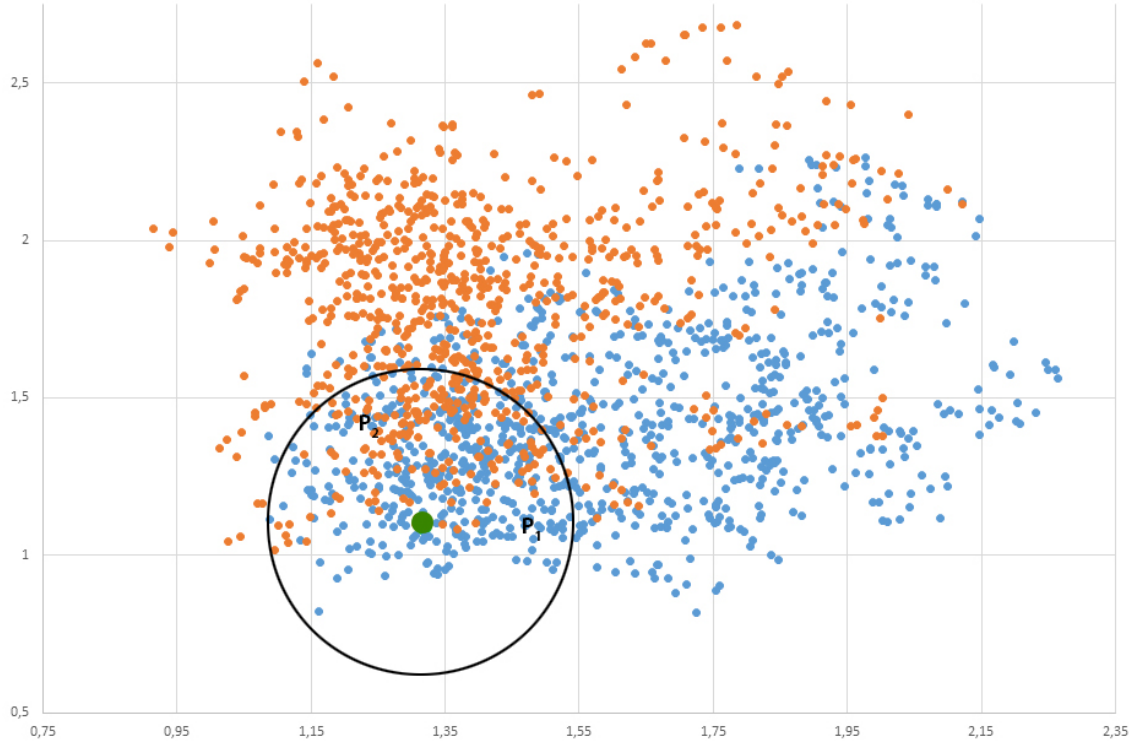

Figure 5.12: Naive Bayes real classification example with 1764 data with 2 classes in the Motor-Imagery scenario

Testing data	**Feature 1**	**Feature 2**
	1.31697	1.09772
Class 1 mean	1.58056	1.40684
Class 2 mean	1.40364	1.79298
Variance 1	0.262085	0.271708
Variance 2	0.218629	0.334132
Probability of Class 1	70.5588%	
Probability of Class 2	23.114%	

Table 5.15: The data used in the Naive Bayes classification example with 1764 data with 2 classes in the Motor-Imagery scenario

5.8 Motor-Imagery experiment 5: Classifier comparisons

In this experiment we will make a comparison between the different algorithms that have been developed in this project in terms of their accuracy. The LDA-CMAES algorithm described in [103], the LDA algorithm that is the default OpenViBE algorithm and the SVM algorithm that is also included in OpenViBE.

For this experiment the motor-imagery dataset has been used without cross-validation. In the case of the KNN, KNNE and Nearest CentroID algorithms we can choose different ways to calculate the distance but now we will show the best accuracy. For this reason, we will show the KNN and KNNE accuracy obtained with the Euclidean distance and with the optimal k value, that in this case is calculated by $\sqrt{number of features}$. For the Nearest CentroID algorithm we will present the results achieved using the Cosine distance.

Figure 5.13 shows the accuracies of each algorithm for the default dataset.

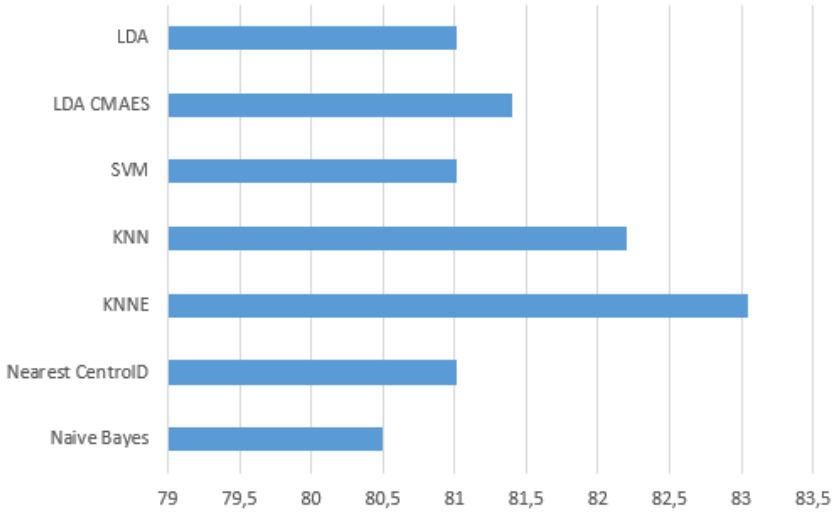

Figure 5.13: Accuracies achieved by the classification algorithms in Motor-Imagery scenario

As we can see in Figure 5.13, the accuracies of the algorithms developed for this project are competitive with the LDA, LDA-CMAES and SVM algorithms that had been previously developed. Moreover, we can see that the KNNE and KNN with the optimal configuration gets the best accuracy value beating the LDA-CMAES algorithm for this dataset. Also, the KNNE obtains better results than the KNN because it improves the KNN solving the outliers problem. In this case, the worst classifiers are the Nearest CentroID and

Naive Bayesian but none of them has an accuracy lower than 80.5%. Also, we can observe that the LDA-CMAES algorithm gets a slightly better accuracy than the LDA algorithm. Furthermore, the SVM algorithm gets the same accuracy as the LDA, that is 81%. These results are not completely valid because we are training and classifying the algorithm with the same data values. For this reason, we are going to pay more attention to the results obtained with cross-validation because they bring us more valid results.

Now we will compare the accuracy predictions obtained using cross-validation for all the algorithms explained before with their optimal configuration. In Figure 5.14 we can observe how the predicted accuracies change along the 15 executions and also the mean values of each algorithm are shown in Table 5.16.

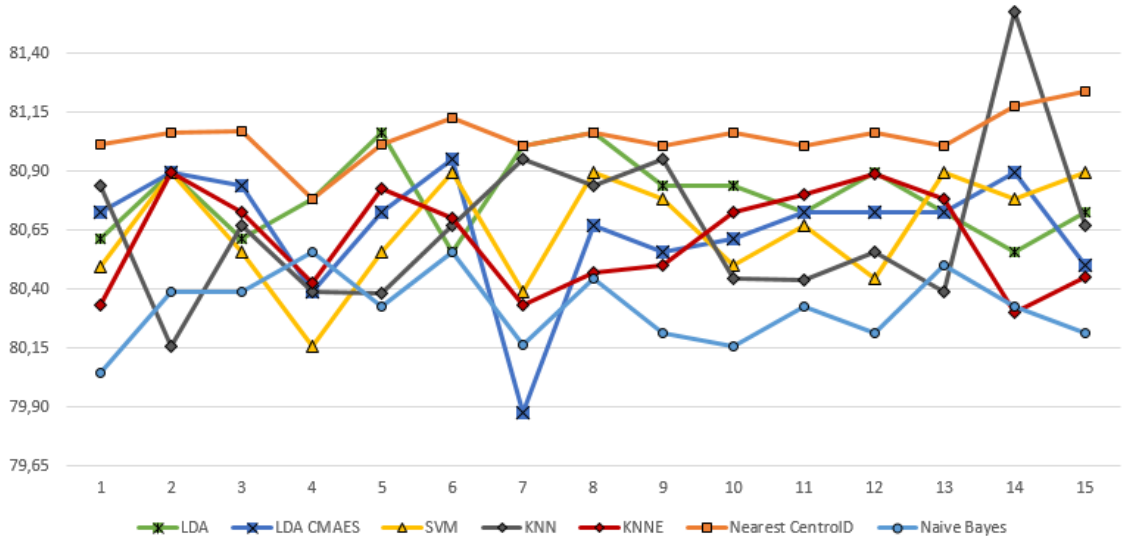

Figure 5.14: Predicted accuracies of the algorithms using 5 cross-validation in the Motor-Imagery scenario

	LDA	LDA-CMAES	SVM	KNN	KNNE	NCentroID	Naive Bayes
Mean accuracy	80.79	80.65	80.65	80.66	80.67	81.05	80.37
Mean variance	1.3	1.47	1.45	1.66	1.98	1.35	1.76
Mean sensitivity	80.3	82.2	83.7	86.78	86.67	81.53	85.05
Mean specificity	78.5	80.05	78	77.16	78.99	80.41	75.63

Table 5.16: Mean accuracies, variances, sensitivity and specificity of 15 executions of the classification algorithms using 5 cross-validation in the Motor-Imagery scenario (in percentages)

As we can observe in Figure 5.14 and Table 5.16, the most stable algorithm is the Nearest CentroID which obtains the highest cross-validation accuracy prediction in most of cases.

Moreover, we can see that the accuracy predictions of the other algorithms are more unstable but they still obtain accuracy prediction values higher than 79.75%. In the case of the KNN obtains the highest accuracy considering each trial independently with 81.58% but it is because of its instability. Also, we have to remind that the accuracy of the algorithms heavily depends on the characteristics of the datasets.

We have used Kruskal Wallis statistical analysis to determine whether that our algorithms have significant differences with the OpenViBE algorithms. For this reason, we have compared the LDA, LDA-CMAES, SVM, KNN and KNNE algorithms and with an adjusted hypothesis $H = 5,875$ we obtain $P = 0.209 > 0.05$ which indicates that there are not a statistically significant difference between them. This also can be understood as if our algorithms were competitive with the OpenViBE algorithms. Also, we have compared the Nearest CentroID and Naive Bayes classifier with the other algorithms and with an adjusted hypothesis $H = 43.72$ we obtain $P = 8.40\text{E}{-8} < 0.01$ which indicates that there is a highly significant difference between them. Moreover, we can see that all the algorithms have a high sensitivity and specificity proportions. For this reason, we can affirm that they are good algorithms to classify the Motor-Imagery database.

In this case, we can say that the Nearest CentroID obtains better results than the other algorithms so it is also competitive with the other algorithms. However, the Naive Bayes classifier obtains worst results than the others.

In conclusion, we can determine that all the developed algorithms for this project obtain a high accuracy for this dataset, and are competitive with the other algorithms already implemented in OpenViBE.

5.9 P300 experiment 1: *k*-nearest neighbors

In this experiment we will study the characteristics of the KNN in the P300 scenario. This algorithm has the possibility of computing the distance between the cases in the training set and the testing set with three different distance measures and this is one of the questions that we will investigate. Also, the value of k can be changed and we will investigate this issue later in Section 5.9.3. We have used the default dataset explained in Section 5.2 and also we will follow the experimental methodology explained in Section 5.3.

5.9.1 Accuracies

In this section, the behavior of KNN for the different distances will be investigated. All of these accuracies have been computed with the k value explained in Section 5.3, and they are shown in Figure 5.15.

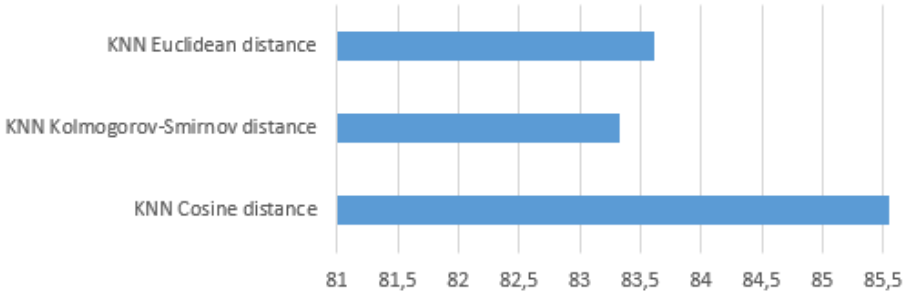

Figure 5.15: KNN accuracies using the three different distances in P300 scenario

As it can be appreciated in Figure 5.15, the best accuracy of the KNN algorithm is obtained using the Cosine distance with a 85.56%. These results are not completely valid because we are training and classifying the algorithm with the same data values. For this reason, we are going to pay more attention to the results obtained with cross-validation because they bring us more valid results.

5.9.2 Performances and variance

In this section, we will show the mean accuracy prediction made by the cross-validation and the mean variance along the 15 executions. This data is shown in Table 5.17.

	Euclidean	**Kolmogorov-Smirnov**	**Cosine**
Mean accuracy	83.41	83.33	84.28
Mean variance	1.33	1.33	1.22

Table 5.17: KNN algorithm performances with the different distances in P300 scenario (in percentages)

In Table 5.17, we can observe that the mean accuracy and the mean variance of each distance do not have a big difference between them. For this reason, we have used Kruskal Wallis statistical test [17] to determine if the KNN results obtained with these distances are statistically significant.

First of all, we have compared the Euclidean and Kolmogorov-Smirnov distances. With the adjusted hypothesis $H = 0.233$ and 15 values of each distance, we obtain a $P = 0.63 > 0.05$. For this fact, we can say that there is not a significant difference between them.

Secondly, we have compared the KNN results achieved with the Cosine distance with the other distances. With the adjusted hypothesis $H = 11.175$ and 15 values of each distance, we obtain a $P = 0.000829 < 0.01$. For this fact, we can say that there is a highly significant difference between them.

In conclusion, we could say that there are significant differences between KNN with Cosine and KNN with any of the two other distances. Also, between the Euclidean and Kolmogorov-Smirnov distances there are no significant differences. We can affirm that the Cosine distance is the best choice for this algorithm.

5.9.3 Study of k value

Now, we will study the accuracies of the KNN algorithm using the Euclidean method changing the value of k. For this paradigm we will start from $k = 1$ because in this case we obtain real correct solutions for this value. Figure 5.16 shows how the accuracies of the KNN algorithm change with the number of k.

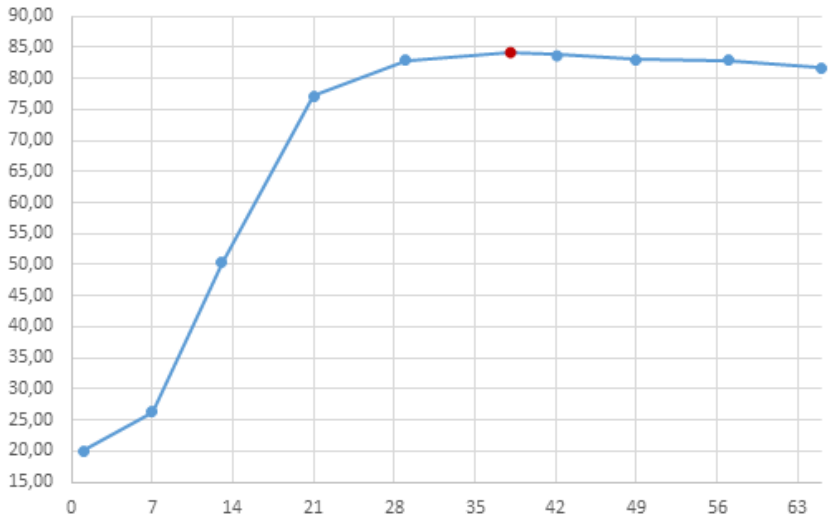

Figure 5.16: KNN accuracies along the variation of k value in P300 scenario

As we can see in Figure 5.16, the accuracy increases considerably for the first 28 values of k. Then this value starts becoming more stable and our investigation concludes that the optimal accuracy is for $k = 38$. The k values have been tested and their accuracies are shown in Table 5.18.

k	1	7	13	21	29	37	42	49	57	65
Accuracy	20.01	26.33	50.35	77.12	82.86	84.28	84	83	82.9	81.67

Table 5.18: The accuracies of KNN for different values of k in the P300 scenario (in percentages)

5.9.4 Sensitivity and specificity

In this section, we will investigate the sensitivity and specificity of the KNN algorithm using cross-validation with the three developed distances and with the optimal k value. This investigation, with the obtained data are shown in Table 5.19, will determine if the algorithm classifies properly the data of the P300 database which is distributed in a 16.66% and 83.33% for Class 1 and Class 2, respectively. Since our accuracies are around 83.33% it could be happening that our algorithm was classifying always for the dominant class with which we will obtain also a 83.33% accuracy.

	Correctly classified	**Wrongly classified**	**Sensitivity**	**Specificity**
Euclidean	83.41	16.59	1.67	100
Kolmogorov-Smirnov	83.33	16.67	0.42	100
Cosine	84.28	15.72	16.67	99.33

Table 5.19: Sensitivity and specificity of the KNN classifier with cross-validation in P300 scenario (in percentages)

As we can see in Table 5.19, the sensitivity accuracies of the Euclidean and Kolmogorov-Smirnov are nearly 0%, which indicates that these two methods are not sensitive because they have a low proportion of Classes 1 which are correctly identified as such. In the case of the sensitivity the best result is obtained with the Cosine distance which obtains a 16.67% and it is still a low value of sensitivity. The specificity of the three methods are close to 100% showing that they have a high proportion of Classes 2 which are correctly identified as such classify.

In conclusion, we could say that the KNN is not a good option for classify this P300 database because the sensitivity is low. Also, if the database was more balanced KNN would obtain bad results. For these reasons, we could determine that KNN is not sensitive to close data.

5.10 P300 experiment 2: k-Nearest Neighbors Equality

In this experiment we will study the characteristics of the KNNE in the P300 scenario. This algorithm has the possibility of computing the distance between the cases in the training set and the testing set with three different distance measures and this is one of the questions that we will investigate. Also, the value of k can be changed and we will investigate this issue later in Section 5.10.3. We have used the default dataset explained in Section 5.2 and also we will follow the experimental methodology explained in Section 5.3.

5.10.1 Accuracies

In this section, the different distances developed for KNNE will be investigated. All of these accuracies have been computed with the k value explained in Section 5.3 and we can see them in Figure 5.17.

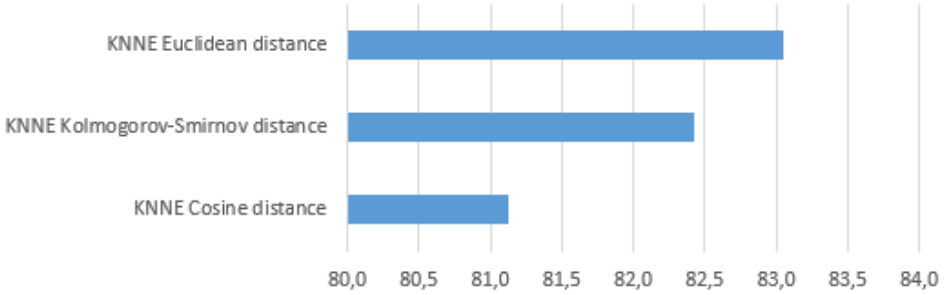

Figure 5.17: KNNE accuracies using the three different distances in P300 scenario

As it can be appreciated in Figure 5.17, the most optimal configuration of the KNNE algorithm is using the Euclidean distance with a 96.39%. Moreover, the Cosine distance obtains a good accuracy with 91.81% and the worst is the Kolmogorov-Smirnov distance. These high accuracies that have been computed without cross-validation can be wrong because they can be overfitted. Since we have not more databases, we can not determine if they are overfitted. Also, these results can be incorrect because we have an unbalanced database. From these facts, we are going to pay more attention to the results obtained with cross-validation.

5.10.2 Performances and variance

In this section, we will show the mean accuracy prediction and the mean variance made by the cross-validation along the 15 executions for each distance method. The data is shown in Table 5.20.

	Euclidean	**Kolmogorov-Smirnov**	**Cosine**
Mean accuracy	83.54	83.33	84.52
Mean variance	2.29	3.03	2.59

Table 5.20: KNNE algorithm performances with the different distances in P300 scenario (in percentages)

In Table 5.20, we can observe that the mean accuracy and the mean variance of each distance do not have a slightly difference between them. For this reason, we have used the Kruskal Wallis statistical test [17] to determine that these distances are statistically significant or not.

First of all, we have compared the Kolmogorov-Smirnov and Euclidean distances. With the adjusted hypothesis $H = 1.073$ and 15 values of each distance, we obtain a $P = 0.3 > 0.05$. From this fact, we can say that there is not a significant difference between them.

Secondly, we have compared the Cosine distance with the Kolmogorov-Smirnov and Euclidean distances separatelly and we have obtained the similar values for both. With the adjusted hypothesis $H = 9.988$ and 15 values of each distance, we obtain a $P = 0.001576 < 0.01$. From this fact, we can say that there is a highly significant difference between them.

In conclusion, we could say that there are significant differences between KNNE with Cosine and KNNE with any of the two other distances. Also, between the Kolmogorov-Smirnov and Euclidean distances there are no significant differences.

5.10.3 Study of k value

Now, we will study the accuracies of the KNNE algorithm using the Euclidean method changing the value of k. For this paradigm we will start from $k = 7$ because with the previous values we do not obtain real correct solutions for this algorithm. Figure 5.18 shows how the accuracies of the KNNE algorithm change with the number of k.

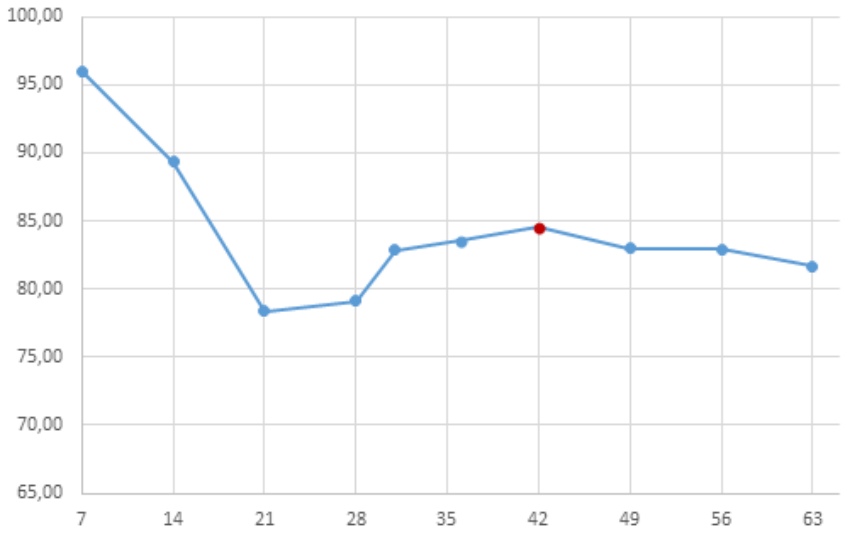

Figure 5.18: KNNE accuracies along the variation of k value in P300 scenario

As we can see in Figure 5.18, the accuracy decreases for the firs 28 values of k. Then the value starts becoming more stable and our investigation concludes that the optimal accuracy is for $k = 42$. Although the best values of k could be others we considerate that the better value of k for all type of databases should be $\sqrt{number of features} = \sqrt{240} = 42$. In this algorithm we have opted to compute this k value using the number of features with less quantity. Moreover, the k values that we have tested and their accuracies can be seen in the table 5.21.

5.18.

k	7	14	21	28	31	36	42	49	56	63
Accuracy	96.02	89.33	78.34	79.12	82.85	83.56	84.52	83	82.9	81.67

Table 5.21: The accuracies of KNNE for different values of k in the P300 scenario (in percentages)

5.10.4 Sensitivity and specificity

In this section, we will investigate the sensitivity and specificity of the KNNE algorithm using cross-validation with the three developed distances and with the optimal k value. This investigation, with the obtained data are shown in Table 5.22, will determine if the algorithm classifies properly the data of the P300 database which is distributed in a 16.66% and 83.33% for Class 1 and Class 2 respectively. Since we can determine only with the accuracy because of distribution if it classifies correctly the samples, we will investigate this topic.

	Correctly classified	**Wrongly classified**	**Sensitivity**	**Specificity**
Euclidean	83.54	16.46	78.33	100
Kolmogorov-Smirnov	83.33	16.67	14.17	100
Cosine	84.52	15.48	63.33	99.75

Table 5.22: Sensitivity and specificity of the KNNE classifier with cross-validation in P300 scenario (in percentages)

As we can see in Table 5.22, the sensitivity of the Euclidean and Cosine distances are 78.33% and 63.33%, which indicates that these two methods have a high proportion of Classes 1 which are correctly identified as such. Moreover, the specificity of the three methods are close to 100% showing that they have a high proportion of Classes 2 which are correctly classified as such.

In conclusion, we could say that the KNNE is a good option to classify this P300 database and also with databases in which the samples are so close each other and also if they have outliers. Moreover, the best choice would be the Cosine distance, although the Euclidean distance also obtains really good results. We could determine that the KNNE is a good improvement of the KNN algorithm and makes it sensitive to data.

5.11 P300 experiment 4: Nearest CentroID

In this experiment we will study the characteristics of the Nearest CentroID which has the possibility of calculating the distance between the centroID and the testing set in three different ways. We have used the P300 dataset explained in Section 5.2 and also we will follow the experiment methodology explained in Section 5.3.

5.11.1 Accuracies

The accuracies obtained using the different distance methods developed for this algorithm are shown in Figure 5.19.

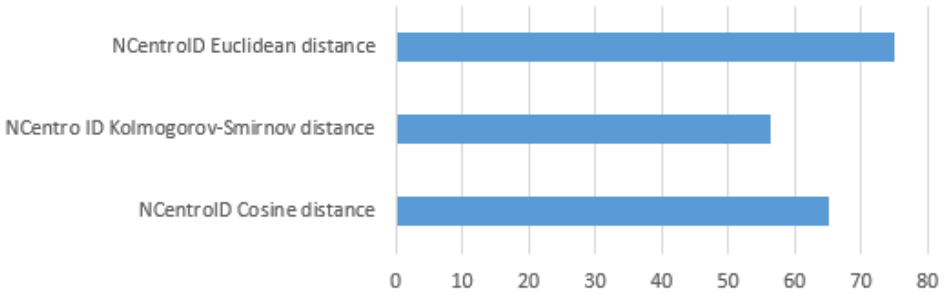

Figure 5.19: Nearest centroID accuracies using the distance methods in P300 scenario

As we can see in Figure 5.19, the optimal configuration of the nearest centroID algorithm is using the Euclidean distance which obtains a 75.07% accuracy. The other two accuracies decrease considerably with the Kolmogorov-Smirnov distance with a 56.25% accuracy and the Cosine distance with a 65.14% accuracy. Also, the outliers could make imprecise the centroID, moving it away from the testing sample. These results are not completely valid because we are training and classifying the algorithm with the same data values. For this reason, we are going to pay more attention to the results obtained with cross-validation because they bring us more valid results.

5.11.2 Performances and variance

Now, we will show you the mean accuracy prediction obtained using the cross-validation along the 15 executions for each distance method. Table 5.23 shows the variance and the predicted accuracy of each distance.

	Euclidean	**Kolmogorov-Smirnov**	**Cosine**
Mean accuracy	74.45	55.87	65.84
Mean variance	2.54	2.89	1.89

Table 5.23: Nearest CentroID algorithm performances with the different distances in the P300 scenario (in percentages)

In Table 5.23, we can observe that the mean accuracy and the mean variances of each distance have a slightly difference between them. Although, we can see that the difference between these distances are statistically significant we will use Kruskal Wallis statistical test [17].

We have compared the three distances at the same time and with the adjusted hypothesis $H = 11$ and 15 values of each distance, we obtain a $P = 0.000911 < 0.01$. For this fact, we can ensure that there are significant differences between them.

In conclusion, we can affirm that Euclidean distance is the best choice for this algorithm and dataset but this does not mean that it will be the best choice for other datasets and signal type.

5.11.3 Sensitivity and specificity

In this section, we will investigate the sensitivity and specificity of the Nearest CentroID algorithm using cross-validation with the three developed distances. This research will determine if the algorithm classifies properly the data of the P300 database which is distributed in a 16.66% for Class 1 and 83.33% for Class 2. The data obtained in this investigation is shown in Table 5.24.

	Correctly classified	**Wrongly classified**	**Sensitivity**	**Specificity**
Euclidean	74.45	25.55	78.24	74.42
Kolmogorov-Smirnov	55.87	44.13	57.92	55.88
Cosine	65.84	34.16	16.67	99.33

Table 5.24: Sensitivity and specificity of the Nearest CentroID classifier with cross-validation in P300 scenario (in percentages)

As we can see in Table 5.24, the sensitivity of the Cosine is low which indicate that the proportion of Classes 1 which are correctly classified as such is low. Also, the Kolmogorov-Smirnov distance obtains a better proportion but it is not as good as the proportion of the Euclidean distance which obtains a 78.24%. Moreover, the Cosine obtains the best specificity proportion which is close to 100% but the Euclidean also obtains a good proportion which indicate that the proportion of Classes 2 are correctly classified as such.

In conclusion, we could determine that Nearest CentroID algorithm using the Euclidean distance would be the best choice for the most databases because of its high values in sensitivity and specificity.

5.12 P300 experiment 4: Naive Bayes

In this experiment we will study the characteristics of the Naive Bayes classifier which calculates the probability of the occurrence of the testing sample in each class. We have used the default dataset explained in Section 5.2 and also we will follow the experiment methodology explained in Section 5.3.

5.12.1 Performances and variance

This section presents the mean accuracy prediction obtained using the cross-validation along the 15 executions and also the mean variance. For this database, the Naive Bayes has obtained a 73.25% accuracy with 2.24% variance. These results are not completely valid because we are training and classifying the algorithm with the same data values. For this reason, we are going to pay more attention to the results obtained with cross-validation because they bring us more valid results.

5.12.2 Sensitivity and specificity

In this section, we will investigate the sensitivity and specificity of the Naive Bayes algorithm using cross-validation. This investigation will determine whether that the algorithm classifies properly the data of the P300 database which is distributed in a 16.66% for Class 1 and 83.33% for Class 2. The results from this investigation are shown in Table 5.25.

Correctly classified	Wrongly classified	Sensitivity	Specificity
73.25	26.75	82.92	72.48

Table 5.25: Sensitivity and specificity of the Naive Bayes classifier with cross-validation in the P300 scenario (in percentages)

As we can see in Table 5.25, the sensitivity accuracy is high with a 82.92% which indicates that it has a high proportion of Classes 1 which are correctly identified as such. Moreover, it obtains a high specificity accuracy with a 72.48% that indicates that it has a high proportion of Classes 2 which are correctly identified as such.

In conclusion, we could determine that this algorithm will obtain better results with more balanced databases but it is a really good choice for all type of databases because of its sensitivity and specificity good values.

5.13 P300 experiment 5: Classifier comparisons

In this experiment we will compare the different algorithms that have been developed in this project in terms of their accuracy. The LDA algorithm that is the default OpenViBE algorithm and the SVM algorithm that is also included in OpenViBE. In this experiment we will not use the LDA-CMAES algorithm [103] because it is very sensitive to the number of variables, while the number of variables increases more slow the algorithm is and in this case it will stay executing several hours. For this reason, it has been discarded for this database and scenario.

For this experiment the P300 dataset has been used without cross-validation. In the case of the KNN, KNNE and Nearest CentroID algorithms we will show the best accuracies of the best distance. For this reason, we will show the KNN accuracy obtained with the Cosine distance and with the optimal k value to this dataset, that in this case is $\sqrt{number of features}$. As the same way, we will use the Euclidean distance for KNNE and the optimal k value that we will discuss later. Moreover, for the Nearest CentroID algorithm we will present the results achieved using the Euclidean distance.

Figure 5.20 shows the accuracies of each algorithm for the default dataset.

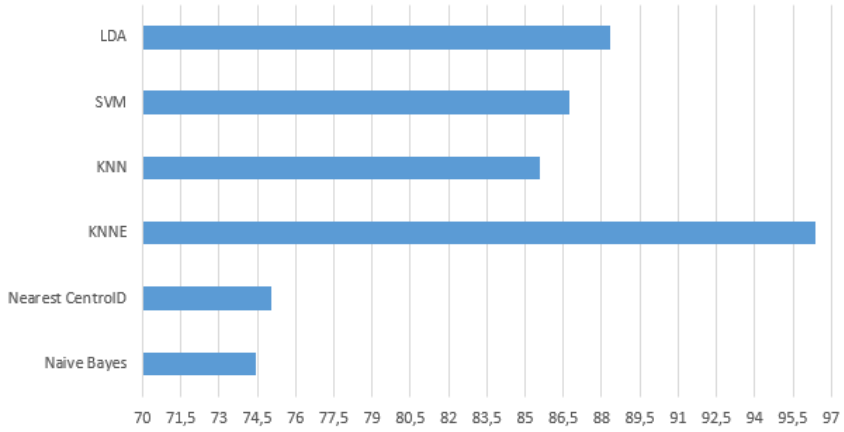

Figure 5.20: Accuracies achieved by the classification algorithms in P300 scenario

As we can see, the accuracies of the algorithms developed for this project are good but the most competitive in this database in accuracy term is the KNNE which obtains an accuracy of 96.39%. This value can be overfitted because of the unbalanced database. For this reason, we are going to pay more attention to the accuracies obtained with cross-validation. Also the KNN is a competitive algorithm with a 85.56% accuracy. We can see

that the LDA and SVM obtains the second and third best accuracies with a 88.33% and 86.18% respectively. In this case the worst classifiers are the Nearest CentroID and Naive Bayesian with 75.07% and 74.44% respectively. These results are not completely valid because we are training and classifying the algorithm with the same data values. For this reason, we are going to pay more attention to the results obtained with cross-validation because they bring us more valid results.

Now we will compare the accuracy predictions made for the cross-validation for all the algorithms explained before with the optimal configuration. In Figure 5.21 we can observe how the predicted accuracies for the cross-validation change along the 15 executions and also the mean values of each algorithm in Table 5.26.

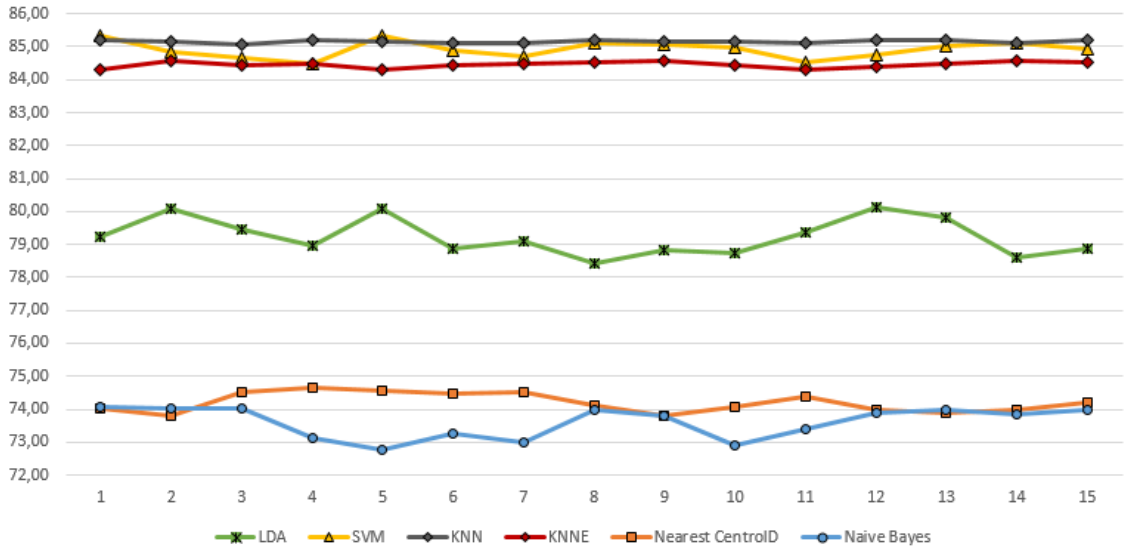

Figure 5.21: Predicted accuracies of the algorithms using 5 cross-validation in P300 scenario

	LDA	SVM	KNN	KNNE	NCentroID	Naive Bayes
Mean accuracy	79.24	84.92	84.28	84.52	74.45	73.25
Mean variance	1.8	1.67	1.22	2.59	2.54	2.24
Mean sensitivity	87.9	19.6	16.67	63.33	78.24	82.92
Mean specificity	88.4	99.5	99.33	99.75	74.42	72.48

Table 5.26: Mean accuracies, variances, sensitivity and specificity of 15 executions of the classification algorithms using 5 cross-validation in P300 scenario (in pertentages)

As we can observe in Figure 5.21 and Table 5.26, the cross-validation accuracies and the accuracies with the database showed in Figure 5.20 are different because without cross-

validation we compare all the database and with cross-validation we obtain a prediction that in this case is more precise. For this reason we will pay more attention to results with cross-validation.

The most stable algorithms are the KNN and KNNE which in the case of KNN obtains one of the best cross-validation accuracy predictions. Moreover, we can see that the accuracy prediction of the SVM is the highest accuracy but it is more unstable than the KNN. Also, the other algorithms obtain worst accuracies with cross-validation than without it but in they case these changes are not very significant. These results reveal that this unbalanced database is difficult.

Moreover, we have used Kruskal Wallis statistical analysis that our algorithms have a significant differences with the OpenViBE algorithms. For this reason, we have compared the LDA, SVM, KNN, KNNE, Nearest CentroID and Naive Bayes algorithms and with an adjusted hypothesis $H = 81.93$ we obtain $P = 3.31\text{E}{-16} < 0.01$ which indicates that there are highly significant differences between them. Also, we have compared all of them in pairs and there are also highly significant differences between them. For this reason, we can determine that KNNE obtain the best of the results of all the algorithms. Also, we can affirm that KNN is worst than the LDA and SVM but has not a high significant difference with them. Moreover, the Nearest CentroID and Naive Bayes are the worst algorithms but they still get good results.

In conclusion and taking account the sensitivity and specificity of the algorithms, we can determine that the LDA, Naive Bayes, Nearest CentroID and KNNE are the most sensitive an specific algorithms for all type of databases and they will obtain better results if the database is balanced. However, the SVM and KNN obtain good results for this database but in a balanced database they will obtain bad results.

CHAPTER 6.

Conclusions and future work

Once analyzed the different classification algorithms and obtained the corresponding results, we can expose the conclusions of the project and the personal experience. Moreover, we are going to discuss other possible topics that could expand this work.

6.1 Conclusions about the objectives

The initial idea of this project was to develop classification methods based on evolutionary algorithms and their incorporation to OpenViBE. This initial idea changed because of the developing difficulty of this type of classification algorithm implementation. Then, we focused on classification algorithms not yet implemented in OpenViBE. These new goals were reached developing 4 classification algorithms for OpenViBE.

Also, a validation of our algorithms has been made with our experiments which shows the strengths and weaknesses of our algorithms and demonstrates that we have got competitive algorithms.

In addition, another strength of this work is the modularity of our algorithms that can be used for any number of features inputs and has been demonstrated with our experiments.

To summarize, the main achieving of this thesis has been the following:

- We have reviewed work on BCI, analyzed in detail the Motor-Imagery and P300 paradigms, and studied different methods for EEG data feature extraction.

- We have studied the design principles of the OpenViBE software.
- We have studied different classification algorithms.
- Four different classifiers have been implemented and added to OpenViBE. The implementations include the possibility of using different parameters like the number of neighbors and the type of distances.
- A comparison between the implemented classifiers and those included in OpenViBE has been conducted.

6.2 Personal experience

First time I heard about the topic of this project I had not a concrete idea what it was about. But once I began reading, searching and working on it I saw the advantages that the topic of the project has and also how it could fit in my future employment. This made me get more interested on this subject and made me work harder to get this project done. Although the way to get this project finished had not been easy, the problems that appeared during the development made me work harder spending a lot of hours and effort to solve these problems. Now I think that these effort and time have brought me a lot of knowledge and also have fulfilled me as a software developer.

All the learned lessons during the degree have been useful also to overcome these problems. Over the years I have noticed that the first years of the college I thought that all the lessons that I was learning will be useless to my future but once you get to this future and you put all this knowledge into practice you realize that it has not been a waste of time.

For all these reasons, I have realized that they have got to give to the students that were interested on computer engineering the knowledge to become into a competitive computer engineer in the professional world.

6.3 Future Work

Starting from the work that has been made in this thesis we mention different ways in which it could be extended.

1. To develop more competitive classification algorithms for OpenViBE to bring to the user more choices.
2. To develop classification algorithms based on evolutionary algorithms.
3. An exhaustive analysis of the algorithms developed in this work using different databases and also determining for what type of paradigms and data they are the best options.
4. To collect new databases created with our own device to be able to classify these databases and test our algorithms with more users.
5. A research to improve the developed algorithm in accuracy and speed data analysis.

Appendices

APPENDIX A.

Software installation

A.1 Ubuntu installation

It is necessary to follow these simple steps to install Ubuntu in the computer:

1. Download the installation Ubuntu CD, concretely the Desktop CD from its website:

   http://www.ubuntu.com/download/desktop

2. The downloaded file is a ISO image that must be burned to a CD or burn it virtually with an auxiliary Software.

3. Switch on the computer from the CD, for this reboot the computer with the recorded CD in the reader.

4. Finally, follow the installation steps. There is a website that helps with the installation:

   http://www.ubuntu.com/download/desktop/install-ubuntu-desktop

   In this website, the previous steps that I have explained before and the installation steps can be seen.

A.2 OpenViBE installation

OpenViBE is available for Windows and Linux but this installation tutorial is only for the Linux OpenViBE.

1. Download the installation OpenViBE file from its website, concretely the stable source code:

   http://openvibe.inria.fr/downloads/

2. Decompress the OpenViBE file and copy it to the place where was decided to be the program.

3. Open a new terminal in your Ubuntu and obtain switch on the administrator with *sudosu* command to avoid permissions errors.

4. In order to build the software, install several dependencies. This may be done using the provided script. Run the following inside the *scripts/* directory:

   $linux-install\_dependencies$ installs dependencies under Linux.

   The Linux installer processes a few source packages and builds them from scratch. Most of the packages are installed from Linux distribution.

5. To build OpenViBE, you can use the $linux-build$ script that it is inside the *scripts/* directory.

6. Finally, the $linux-test$ script will launch the most interesting applications for the user. Now the OpenViBE is built in *dist/* directory where the user can find different applications. One of the most interesting application is the $openvibe-designer.sh$.

Bibliography

[1] Altman, D. G. and Bland, J. M. (1994). Diagnostic tests. 1: Sensitivity and specificity. *BMJ: British Medical Journal*, 308(6943):1552.

[2] Anderson, C. W. and Sijercic, Z. (1996). Classification of EEG signals from four subjects during five mental tasks. In *Solving engineering problems with neural networks: proceedings of the conference on engineering applications in neural networks (EANN'96)*, pages 407–414. Turkey.

[3] Ang, K. K., Chin, Z. Y., Wang, C., Guan, C., and Zhang, H. (2012). Filter bank common spatial pattern algorithm on BCI competition IV datasets 2a and 2b. *Frontiers in Neuroscience*, 6.

[4] Arrouët, C., Congedo, M., Marvie, J.-E., Lamarche, F., Lécuyer, A., and Arnaldi, B. (2005). Open-vibe: a three dimensional platform for real-time neuroscience. *Journal of Neurotherapy*, 9(1):3–25.

[Astigarraga et al.] Astigarraga, A., Arruti, A., Muguerza, J., Santana, R., Martin, J. I., and Sierra, B. User adapted motor-imaginary brain-computer interface by means of EEG channel selection based on estimation of distributed algorithms.

[6] Back, T. (1996). *Evolutionary algorithms in theory and practice*. Oxford Univ. Press.

[7] Baskar, S., Suganthan, P., Ngo, N., Alphones, A., and Zheng, R. (2006). Design of triangular FBG filter for sensor applications using covariance matrix adapted evolution algorithm. *Optics Communications*, 260(2):716–722.

[8] Bastuji, H., García-Larrea, L., Franc, C., and Mauguière, F. (1995). Brain processing of stimulus deviance during slow-wave and paradoxical sleep: A study of human auditory evoked responses using the oddball paradigm. *Journal of Clinical Neurophysiology*, 12(2):155–167.

[9] BCI2000 (2015). BCI2000 sofware platform. http://www.schalklab.org/research/bci2000. [Online; accessed 15-april-2015].

[10] BCILAB (2015). BCILAB. http://sccn.ucsd.edu/wiki/BCILAB. [Online; accessed 15-april-2015].

[11] Bennett, K. P. and Campbell, C. (2000). Support vector machines: hype or hallelujah? *ACM SIGKDD Explorations Newsletter*, 2(2):1–13.

[12] Beyer, H.-G. and Sendhoff, B. (2008). Covariance matrix adaptation revisited: The CMSA evolution strategy. In *Parallel Problem Solving from Nature–PPSN X*, pages 123–132. Springer.

[13] Bishop, C. (1992). Exact calculation of the Hessian matrix for the multilayer perceptron. *Neural Computation*, 4(4):494–501.

[14] Bishop, C. M. et al. (1995). Neural networks for pattern recognition.

[15] Blankertz, B., Kawanabe, M., Tomioka, R., Hohlefeld, F., Müller, K.-r., and Nikulin, V. V. (2007). Invariant common spatial patterns: Alleviating nonstationarities in brain-computer interfacing. In *Advances in neural information processing systems*, pages 113–120.

[16] Bostanov, V. (2004). BCI competition 2003-data sets Ib and IIb: feature extraction from event-related brain potentials with the continuous wavelet transform and the t-value scalogram. *Biomedical Engineering, IEEE Transactions on*, 51(6):1057–1061.

[17] Breslow, N. (1970). A generalized Kruskal-Wallis test for comparing k samples subject to unequal patterns of censorship. *Biometrika*, 57(3):579–594.

[18] Brunner, C., Andreoni, G., Bianchi, L., Blankertz, B., Breitwieser, C., Kanoh, S., Kothe, C. A., Lécuyer, A., Makeig, S., Mellinger, J., et al. (2013). BCI software platforms. In *Towards Practical Brain-Computer Interfaces*, pages 303–331. Springer.

[19] Burger, C. (2014). A novel method of improving EEG signals for BCI classification. *Stellenbosch University Library and information services*, (1):6–53.

[20] Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167.

[21] Chiappa, S. and Bengio, S. (2003). HMM and IOHMM modeling of EEG rhythms for asynchronous BCI systems. Technical report, IDIAP.

[22] Congedo, M., Goyat, M., Tarrin, N., Ionescu, G., Varnet, L., Rivet, B., Phlypo, R., Jrad, N., Acquadro, M., and Jutten, C. (2011). " Brain invaders": a prototype of an open-source p300-based video game working with the openvibe platform. In *5th International Brain-Computer Interface Conference 2011 (BCI 2011)*, pages 280–283.

[23] Congedo, M., Lotte, F., and Lécuyer, A. (2006). Classification of movement intention by spatially filtered electromagnetic inverse solutions. *Physics in medicine and biology*, 51(8):1971.

[24] Danielsson, P.-E. (1980). Euclidean distance mapping. *Computer Graphics and image processing*, 14(3):227–248.

[25] Daum, F. and Huang, J. (2003). Curse of dimensionality and particle filters. In *Aerospace Conference, 2003. Proceedings. 2003 IEEE*, volume 4, pages 4_1979–4_1993. IEEE.

[26] Dawid, A. P. (1979). Conditional independence in statistical theory. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–31.

[27] De Maesschalck, R., Jouan-Rimbaud, D., and Massart, D. L. (2000). The Mahalanobis distance. *Chemometrics and intelligent laboratory systems*, 50(1):1–18.

[28] Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons.

[29] del R Millan, J., Mourino, J., Babiloni, F., Cincotti, F., Varsta, M., and Heikkonen, J. (2000). Local neural classifier for EEG-based recognition of mental tasks. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 632–636. IEEE.

[30] Dong, W., Chen, T., Tino, P., and Yao, X. (2013). Scaling up estimation of distribution algorithms for continuous optimization. *Evolutionary Computation, IEEE Transactions on*, 17(6):797–822.

[31] Duda, R. O., Hart, P. E., and Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.

[32] Dudani, S. A. (1976). The distance-weighted k-nearest-neighbor rule. *Systems, Man and Cybernetics, IEEE Transactions on*, (4):325–327.

[33] Eddy, S. R. (1996). Hidden Markov models. *Current opinion in structural biology*, 6(3):361–365.

[34] Est, N.-G. (2010). Neuromimetic intelligence.

[35] Fabiani, G. E., McFarland, D. J., Wolpaw, J. R., and Pfurtscheller, G. (2004). Conversion of EEG activity into cursor movement by a brain-computer interface (BCI). *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 12(3):331–338.

[36] Fazel-Rezai, R. and Ahmad, W. (2011). P300-based brain-computer interface paradigm design. *Recent Advances in Brain-Computer Interface Systems*, pages 83–98.

[37] Fernández, F. and Isasi, P. (2008). Local feature weighting in nearest prototype classification. *Neural Networks, IEEE Transactions on*, 19(1):40–53.

[38] Fukunaga, K. (2013). *Introduction to statistical pattern recognition*. Academic press.

[39] Fukunaga, K. and Hummels, D. M. (1987). Bayes error estimation using Parzen and KNN procedures. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (5):634–643.

[40] Gargava, P., Sindwani, K., and Soman, S. (2014). Controlling an Arduino robot using brain computer interface. In *Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions), 2014 3rd International Conference on*, pages 1–5. IEEE.

[41] Genkin, A., Lewis, D. D., and Madigan, D. (2007). Large-scale Bayesian logistic regression for text categorization. *Technometrics*, 49(3):291–304.

[42] Greenblatt, R., Pflieger, M., and Ossadtchi, A. (2012). Connectivity measures applied to human brain electrophysiological data. *Journal of neuroscience methods*, 207(1):1–16.

[43] Hansen, N., Müller, S., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18.

[44] Herbert (2012). Brain Computer Interfaces: OpenViBE as a Platform for a P300 Speller. pages 37–39.

[45] Heyman, D. P. and Sobel, M. J. (2003). *Stochastic Models in Operations Research: Stochastic Optimization*, volume 2. Courier Dover Publications.

[46] Hiraiwa, A., Shimohara, K., and Tokunaga, Y. (1990). EEG topography recognition by neural networks. *Engineering in Medicine and Biology Magazine, IEEE*, 9(3):39–42.

[47] Hoya, T., Hori, G., Bakardjian, H., Nishimura, T., Suzuki, T., Miyawaki, Y., Funase, A., and Cao, J. (2003). Classification of single trial eeg signals by a combined principal+ independent component analysis and probabilistic neural network approach. In *Proc. ICA2003*, volume 197.

[48] Jain, A. K., Duin, R. P. W., and Mao, J. (2000). Statistical pattern recognition: A review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(1):4–37.

[49] Jerbi, K., Lachaux, J.-P., Karim, N., Pantazis, D., Leahy, R. M., Garnero, L., Baillet, S., et al. (2007). Coherent neural representation of hand speed in humans revealed by MEG imaging. *Proceedings of the National Academy of Sciences*, 104(18):7676–7681.

[50] Joachims, T. (1996). A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. Technical report, DTIC Document.

[51] Koradi, R., Billeter, M., Engeli, M., Güntert, P., and Wüthrich, K. (1998). Automated peak picking and peak integration in macromolecular NMR spectra using AUTOPSY. *Journal of Magnetic Resonance*, 135(2):288–297.

[52] Larrañaga, P., Karshenas, H., Bielza, C., and Santana, R. (2012). A review on probabilistic graphical models in evolutionary computation. *Journal of Heuristics*, 18(5):795–819.

[53] Larrañaga, P. and Lozano, J. A. (2002). *Estimation of distribution algorithms: A new tool for evolutionary computation*, volume 2. Springer Science & Business Media.

[54] Le, J., Menon, V., and Gevins, A. (1994). Local estimate of surface Laplacian derivation on a realistically shaped scalp surface and its performance on noisy data. *Electroencephalography and Clinical Neurophysiology/Evoked Potentials Section*, 92(5):433–441.

[55] Lécuyer, A., Lotte, F., Reilly, R. B., Leeb, R., Hirose, M., and Slater, M. (2008). Brain-computer interfaces, virtual reality, and videogames. *IEEE Computer*, 41(10):66–72.

[56] Lee, C., Rabiner, L., Pieraccini, R., and Wilpon, J. (1990). Acoustic modeling for large vocabulary speech recognition. *Computer Speech & Language*, 4(2):127–165.

[57] Lee, D. T. and Yamamoto, A. (1994). Wavelet analysis: theory and applications.

[58] Leung, K. M. (2007). Naive Bayesian classifier. *Polytechnic University Department of Computer Science/Finance and Risk Engineering*.

[59] Lotte, F., Congedo, M., Lécuyer, A., and Lamarche, F. (2007). A review of classification algorithms for EEG-based brain–computer interfaces. *Journal of neural engineering*, 4.

[60] Lugt, A. V. (1964). Signal detection by complex spatial filtering. *Information Theory, IEEE Transactions on*, 10(2):139–145.

[61] Massey Jr, F. J. (1951). The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78.

[62] McFarland, D. J., Anderson, C. W., Muller, K., Schlogl, A., and Krusienski, D. J. (2006). BCI meeting 2005-workshop on BCI signal processing: Feature extraction and translation. *IEEE transactions on neural systems and rehabilitation engineering*, 14(2):135.

[63] McFarland, D. J., Lefkowicz, A. T., and Wolpaw, J. R. (1997). Design and operation of an EEG-based brain-computer interface with digital signal processing technology. *Behavior Research Methods, Instruments, & Computers*, 29(3):337–345.

[64] McFarland, D. J., Sarnacki, W. A., and Wolpaw, J. R. (2011). Should the parameters of a BCI translation algorithm be continually adapted? *Journal of neuroscience methods*, 199(1):103–107.

[65] McFarland, D. J. and Wolpaw, J. R. (2005). Sensorimotor rhythm-based brain-computer interface (BCI): Feature selection by regression improves performance. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 13(3):372–379.

[66] McFarland, D. J. and Wolpaw, J. R. (2008). Sensorimotor rhythm-based brain-computer interface (BCI): Model order selection for autoregressive spectral analysis. *Journal of neural engineering*, 5(2):155.

[67] Mcfarland, D. J. and Wolpaw, J. R. (2010). Brain–computer interfaces for the operation of robotic and prosthetic devices. *Advances in Computers*, 79:169–187.

[68] Mendialdua, I. (2015). *Contributions on Distance-Based algorithms, Multi-Classifier Construction and Pairwise Classification*. PhD thesis, Informatika fakultatea UPV/EHU.

[69] Mendialdua, I., Martínez-Otzeta, J., Rodriguez, I., Ruiz-Vazquez, T., and Sierra, B. (2015). Dynamic selection of the best base classifier in one versus one. *Knowledge-Based Systems*.

[70] Mitani, Y. and Hamamoto, Y. (2006). A local mean-based nonparametric classifier. *Pattern Recognition Letters*, 27(10):1151–1159.

[71] Moschitti, A. (2003). *A study on optimal parameter tuning for Rocchio text classifier*. Springer.

[72] Murphy, K. P. (2006). Naive Bayes classifiers. *University of British Columbia*.

[73] Nguyen, H. V. and Bai, L. (2011). Cosine similarity metric learning for face verification. In *Computer Vision–ACCV 2010*, pages 709–720. Springer.

[74] Niemytzki, V. (1927). On the" third axiom of metric space". *Transactions of the American Mathematical Society*, 29(3):507–513.

[75] Nijholt, A. (2009). BCI for games: A 'state of the art' survey. In *Entertainment Computing-ICEC 2008*, pages 225–228. Springer.

[76] Nijholt, D. S. T. A. (2010). Brain-computer interfaces.

[77] Nocedal, J. and Wright, S. J. (2006). *Conjugate gradient methods*. Springer.

[78] OpenViBE (2015a). OpenViBE P300 Magic Card Scenario. http://openvibe.inria.fr/openvibe-p300-magic-card/. [Online; accessed 18-May-2015].

[79] OpenViBE (2015b). OpenViBE software. http://openvibe.inria.fr/. [Online; accessed 16-February-2015].

[80] OpenViBE (2015c). OpenViBE supported hardware. http://openvibe.inria.fr/supported-hardware/. [Online; accessed 16-February-2015].

[81] Papadimitriou, C. H. and Steiglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Courier Dover Publications.

[82] Parikh, R., Mathai, A., Parikh, S., Sekhar, G. C., and Thomas, R. (2008). Understanding and using sensitivity, specificity and predictive values. *Indian journal of ophthalmology*, 56(1):45.

[83] Parker, J. S., Mullins, M., Cheang, M. C., Leung, S., Voduc, D., Vickery, T., Davies, S., Fauron, C., He, X., Hu, Z., et al. (2009). Supervised risk predictor of breast cancer based on intrinsic subtypes. *Journal of clinical oncology*, 27(8):1160–1167.

[84] Pearl, J. (2000). *Causality: Models, Reasoning and Inference*. Cambridge University Press.

[85] Pfurtscheller, G. and Aranibar, A. (1979). Evaluation of event-related desynchronization (erd) preceding and following voluntary self-paced movement. *Electroencephalography and clinical neurophysiology*, 46(2):138–146.

[86] Pfurtscheller, G., Flotzinger, D., and Kalcher, J. (1993). Brain-computer interface—a new communication device for handicapped persons. *Journal of Microcomputer Applications*, 16(3):293–299.

[87] Pfurtscheller, G. and Neuper, C. (2001). Motor imagery and direct brain-computer communication. *Proceedings of the IEEE*, 89(7):1123–1134.

[88] Pfurtscheller, G., Neuper, C., Müller, G. R., Obermaier, B., Krausz, G., Schlögl, A., Scherer, R., Graimann, B., Keinrath, C., Skliris, D., et al. (2003). Graz-BCI: state of the art and clinical applications. *IEEE transactions on neural systems and rehabilitation engineering: a publication of the IEEE Engineering in Medicine and Biology Society*, 11(2):177–180.

[89] Platt, J. C., Cristianini, N., and Shawe-Taylor, J. (1999). Large margin DAGs for multiclass classification. In *nips*, volume 12, pages 547–553.

[90] Rak, R. J., Kołodziej, M., and Majkowski, A. (2012). Brain-computer interface as measurement and control system the review paper. *Metrology and Measurement Systems*, 19(3):427–444.

[91] Rakotomamonjy, A. and Guigue, V. (2008). BCI competition III: Dataset II-ensemble of SVMs for BCI P300 speller. *Biomedical Engineering, IEEE Transactions on*, 55(3):1147–1154.

[92] Rance, G., Dowell, R. C., Rickards, F. W., Beer, D. E., and Clark, G. M. (1998). Steady-state evoked potential and behavioral hearing thresholds in a group of children with absent click-evoked auditory brain stem response. *Ear and Hearing*, 19(1):48–61.

[93] Razali, N. M. and Wah, Y. B. (2011). Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests. *Journal of Statistical Modeling and Analytics*, 2(1):21–33.

[94] Refaeilzadeh, P., Tang, L., and Liu, H. (2009). Cross-validation. In *Encyclopedia of database systems*, pages 532–538. Springer.

[95] Renard, Y., Lotte, F., Gibert, G., Congedo, M., Maby, E., Delannoy, V., Bertrand, O., and Lécuyer, A. (2010). Openvibe: an open-source software platform to design, test, and use brain-computer interfaces in real and virtual environments. *Presence: teleoperators and virtual environments*, 19(1):35–53.

[96] Rivet, B., Souloumiac, A., Attina, V., and Gibert, G. (2009). xDAWN algorithm to enhance evoked potentials: application to brain–computer interface. *Biomedical Engineering, IEEE Transactions on*, 56(8):2035–2043.

[97] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[98] Rossini, L., Izzo, D., and Summerer, L. (2009). Brain-machine interfaces for space applications. In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, pages 520–523. IEEE.

[99] Ruck, D. W., Rogers, S. K., Kabrisky, M., Oxley, M. E., and Suter, B. W. (1990). The multilayer perceptron as an approximation to a Bayes optimal discriminant function. *Neural Networks, IEEE Transactions on*, 1(4):296–298.

[100] Sajda, P., Müller, K.-R., and Shenoy, K. V. (2008). Brain-computer interfaces. *IEEE Signal Processing Magazine*, 25(1):16–17.

[101] Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 26(1):43–49.

[102] Santana, R., Bielza, C., and Larrañaga, P. (2011a). An ensemble of classifiers approach with multiple sources of information. In Klami, A., editor, *Proceedings of*

ICANN/PASCAL2 Challenge: MEG Mind Reading, Aalto University Publication series SCIENCE + TECHNOLOGY, pages 25–30. Aalto University.

[103] Santana, R., Bonnet, L., Legény, J., and Lécuyer, A. (2012). Introducing the use of model-based evolutionary algorithms for EEG-based motor imagery classification. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 1159–1166. ACM.

[104] Santana, R., Muelas, S., Latorre, A., and Peña, J. M. (2011b). A direct optimization approach to the P300 speller. In *Proceedings of the 2011 Genetic and Evolutionary Computation Conference GECCO-2011*, pages 1747–1754, Dublin, Ireland.

[105] Scholkopft, B. and Mullert, K.-R. (1999). Fisher discriminant analysis with kernels. In *Proceedings of the 1999 IEEE Signal Processing Society Workshop Neural Networks for Signal Processing IX, Madison, WI, USA*, pages 23–25.

[106] Stein, C. M. (1981). Estimation of the mean of a multivariate normal distribution. *The annals of Statistics*, pages 1135–1151.

[107] Stein, E. M. and Weiss, G. L. (1971). *Introduction to Fourier analysis on Euclidean spaces*, volume 1. Princeton university press.

[108] Sun, J., Zhang, Q., and Tsang, E. P. (2005). De/eda: A new evolutionary algorithm for global optimization. *Information Sciences*, 169(3):249–262.

[109] Suykens, J. A. and Vandewalle, J. (1999). Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300.

[110] Tass, P., Rosenblum, M., Weule, J., Kurths, J., Pikovsky, A., Volkmann, J., Schnitzler, A., and Freund, H.-J. (1998). Detection of n:m phase locking from noisy data: Application to magnetoencephalography. *Physical Review Letters*, 81(15):3291.

[111] Tibshirani, R., Hastie, T., Narasimhan, B., and Chu, G. (2002). Diagnosis of multiple cancer types by shrunken centroids of gene expression. *Proceedings of the National Academy of Sciences*, 99(10):6567–6572.

[112] TOBI (2015). TOBI common implementation platform. http://www.tobi-project.org/. [Online; accessed 15-april-2015].

[113] Tudor, M., Tudor, L., and Tudor, K. I. (2004). Hans Berger (1873-1941)–the history of electroencephalography. *Acta medica Croatica: casopis Hravatske akademije medicinskih znanosti*, 59(4):307–313.

[114] Ubuntu (2015). Ubuntu operating system. http://www.ubuntu.com/. [Online; accessed 01-March-2015].

[115] Ulichney, R. (1987). *Digital halftoning*. MIT press.

[116] Van Erp, J. B., Lotte, F., and Tangermann, M. (2012). Brain-computer interfaces: beyond medical applications. *Computer*, (4):26–34.

[117] Vapnik, V. N. and Vapnik, V. (1998). *Statistical learning theory*, volume 1. Wiley New York.

[118] Vaughan, T. M., Heetderks, W. J., Trejo, L. J., Rymer, W. Z., Weinrich, M., Moore, M. M., Kübler, A., Dobkin, B. H., Birbaumer, N., Donchin, E., et al. (2003). Brain-computer interface technology: a review of the second international meeting. *IEEE transactions on neural systems and rehabilitation engineering: a publication of the IEEE Engineering in Medicine and Biology Society*, 11(2):94–109.

[119] Walker, J. S. (1996). *Fast fourier transforms*, volume 24. CRC press.

[120] Wang, J., Neskovic, P., and Cooper, L. N. (2006a). Neighborhood size selection in the k-nearest-neighbor rule using statistical confidence. *Pattern Recognition*, 39(3):417–423.

[121] Wang, Q., Garrity, G. M., Tiedje, J. M., and Cole, J. R. (2007). Naive Bayesian classifier for rapid assignment of rRNA sequences into the new bacterial taxonomy. *Applied and environmental microbiology*, 73(16):5261–5267.

[122] Wang, Y., Gao, S., and Gao, X. (2006b). Common spatial pattern method for channel selection in motor imagery based brain-computer interface. In *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*, pages 5392–5395. IEEE.

[123] Wang, Y., Zhang, Z., Li, Y., Gao, X., Gao, S., and Yang, F. (2004). BCI competition 2003-data set IV: an algorithm based on CSSD and FDA for classifying single-trial EEG. *Biomedical Engineering, IEEE Transactions on*, 51(6):1081–1086.

[124] White, H. (1980). A heteroskedasticity-consistent covariance matrix estimator and a direct test for heteroskedasticity. *Econometrica: Journal of the Econometric Society*, pages 817–838.

[125] Wilamowski, B. M. (2009). Neural network architectures and learning algorithms. *Industrial Electronics Magazine, IEEE*, 3(4):56–63.

[126] Wolpaw, J. and Wolpaw, E. W. (2011). *Brain-computer interfaces: Principles and practice*, chapter BCI Signal Processing: Feature Extraction. Oxford University Press.

[127] Wolpaw, J. R., Birbaumer, N., Heetderks, W. J., McFarland, D. J., Peckham, P. H., Schalk, G., Donchin, E., Quatrano, L. A., Robinson, C. J., Vaughan, T. M., et al. (2000). Brain-computer interface technology: A review of the first international meeting. *IEEE transactions on rehabilitation engineering*, 8(2):164–173.

[128] Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, 5(2):241–259.

[129] Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Philip, S. Y., et al. (2008). Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37.

[130] Yazdani, A., Ebrahimi, T., and Hoffmann, U. (2009). Classification of EEG signals using Dempster Shafer theory and a k-nearest neighbor classifier. In *Neural Engineering, 2009. NER'09. 4th International IEEE/EMBS Conference on*, pages 327–330. IEEE.

[131] Zeng, Y., Yang, Y., and Zhao, L. (2009). Pseudo nearest neighbor rule for pattern classification. *Expert Systems with Applications*, 36(2):3587–3595.

[132] Zhang, Q., Sun, J., Tsang, E., and Ford, J. (2004). Hybrid estimation of distribution algorithm for global optimization. *Engineering computations*, 21(1):91–107.

[133] Zhang, Q., Zhou, A., and Jin, Y. (2008). RM-MEDA: A regularity model-based multiobjective estimation of distribution algorithm. *Evolutionary Computation, IEEE Transactions on*, 12(1):41–63.

[134] Zhao, J., Meng, Q., Li, W., Li, M., Sun, F., and Chen, G. (2013). An openvibe-based brainwave control system for cerebot. In *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on*, pages 1169–1174. IEEE.

Acronyms

ACO Ant Colony Optimization.

AR Autoregressive Modeling.

BCI Brain Computer Interfaces.

BLRNN Bayesian Logistic Regression Neural Network.

CA Classification Algorithm.

CIP Common Implementation Platform.

CMAES Covariance Matrix Adaptation Evolution Strategy.

CO Combinatorial Optimization.

DE Differential Evolution.

DP Dynamic Programming.

EA Evolutionary Algorithm.

EDA Estimation of Distribution Algorithm.

EEG Electroencephalography.

EP Evolutionary Programming.

ES Evolution Strategy.

FFT Fast Fourier Transform.

GA Genetic Algorithm.

GM Gradient Method.

GP Genetic Programming.

HMM Hidden Markov Model.

ICA Independent Component Analysis.

IOHMM Input-Output HMM.

KNN *k* Nearest Neighbors.

LDA Linear Discriminant Analysis.

LVQ Learning Vector Quantization.

MEG Magnetoencephalography.

MLP MultiLayer Perceptron.

NB Naive Bayes.

NN Neuronal Networks.

PCI Principal Component Analysis.

PSO Particle Swarm Optimization.

PVL Phase Locking Value.

RBF Radial Basis Function.

SO Stochastic Optimization.

SVM Support Vector Machine.

TOBI Towards Practical Brain-Computer Interfaces.