Training a Gaming Agent on Brainwaves Online

Using brain signals as feedback for reinforcement learning

Abelenda Marcos, Vazquez Agustin Ignacio, Manganaro Bello Santiago

Supervisor: Ramele Rodrigo

Final Project

Computer Engineering Degree

Abstract

The field of Brain Computer Interaction has diverse applications in real life problems, most of them little explored. One of them is the training of an agent immersed in a system making use of human brain signals. These signals can be obtained in different ways, among which Electroencephalography (EEG) stands out due to its ease of use, low cost, effectiveness and its non-invasive condition.

Event-related potential (ERP) is the brain response as a consequence of an event in a given time. Error-related potential (ErrP) is a type of ERP that is related to the detection of an error. When a subject perceives a mistake, in this case, made by an agent, it triggers these potentials. ErrP are detectable in brain signals with the appropriate processing and from this detection it is possible to perform some desired action or take a certain decision.

Spatial filters are a tool that allows to improve signals, eliminating a lot of non-redundant information such as noise.

Reinforcement learning consists of training an agent based on a reward that he seeks to maximize while interacting with the environment. The information obtained from the brain signals can be used as a source to calculate the rewards in a given problem.

Carrying out the complete extraction, processing and learning procedure in real time has greater utility and number of practical applications, and its results could be better.

This thesis works as an extension to [1]. It is based on the training of an agent with the reinforcement learning algorithm with ErrP signals, captured with EEG. It proposes a basic scheme for the online procedure. The use of spatial filters is tested by generating and comparing different processing pipelines.

Keywords – BCI, EEG, ERP, ErrP, Reinforcement Learning, Q-Table, OHC, xDawn, PCA, Cross Validation

Contents

List of Figures

List of Tables

1 Introduction

1.1 Brain computer Interfaces

A Brain-computer interface (BCI) is a system that measures central nervous system (CNS) activity and converts it into articial output that replaces, restores, enhances, supplements, or improves natural CNS output and thereby changes the ongoing interactions between the CNS and its external and internal environment [2].

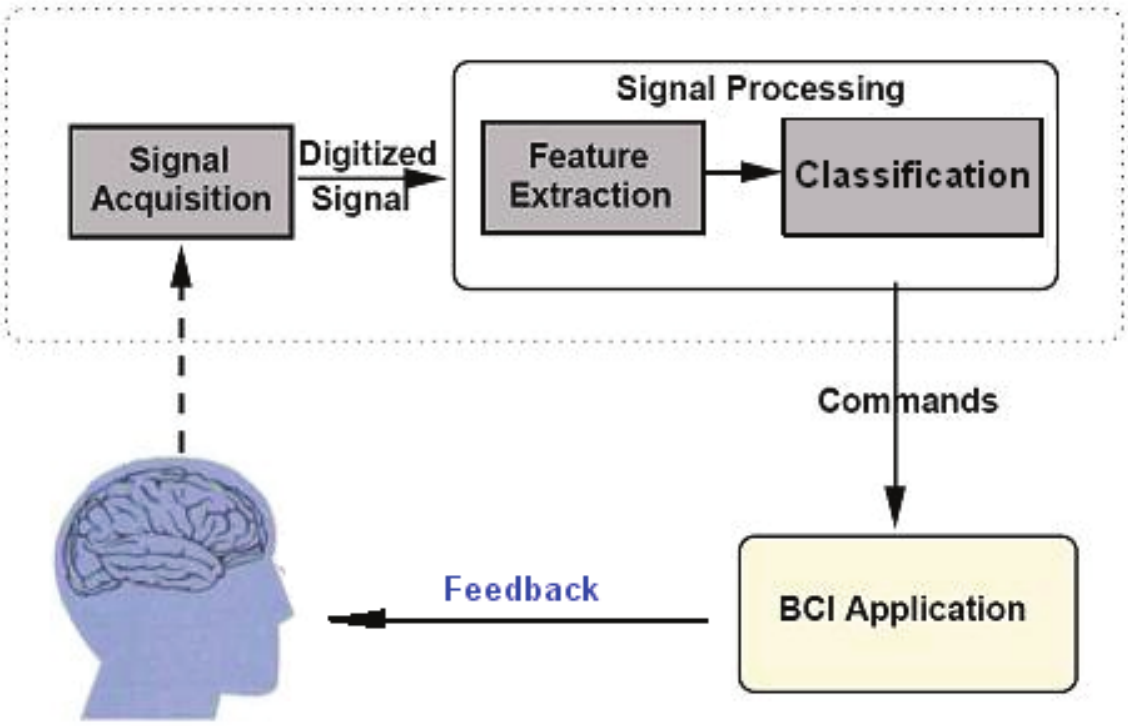


Figure 1.1: The block diagram of a BCI system

The architecture is described in Figure 1.1. It has the following 5 main components:

1. An experiment the subject observes which activates the desired brain activity.
2. A signal acquisition device that captures the brain data in order to be transmitted to a computer device for processing.
3. An signal featuring step where signal is filtered, resampled, among other transformations with the objective of eliminating noise and artifacts and that builds a feature in order to distinguish the signals characteristics and get the most

out of data.

4. A signal classification module which infers the information.
5. An application system that uses the generated information as input to perform a desired action.

Brain-computer interfaces (BCIs) give the possibility to assist people with some kind of motor disability [3], this can be done by converting thoughts into movements of an external device. Helping disabled people to recover autonomy was the initial purpose of BCIs but nowadays the application scope has been growing over time up to the point it has begun to be used in gaming, emotion recognition, mental fatigue evaluation, vigilance estimation, etc. [4] Currently BCIs can determine the intent of the user detecting and processing their Electroencephalographic signals (EEG), for instance, the user may control the modulation of some brain waves (e.g., mu or beta rhythms) or the BCI may exploit natural automatic responses of the brain to external stimuli (e.g., event-related potentials (ErrP)). [5] EEG is the most used measurement method of brain waves, for its non-invasive (does not require implants nor surgeries) and simple nature. However, the captured signals are very weak and of poor quality, as they need to cross several layers of tissues—such as the meninges, the skull and the scalp—before being captured by the electrodes. For this reason, it is often necessary to use several electrodes to have a higher spatial resolution and a more precise system.

1.1.1 Signal Acquisition

Over the past years, number of technologies have been developed to measure the activity of the human brain. Some of the techniques measure the variation of the electrical activities related to the different states of the brain while some other techniques measure other different parameters. Available modalities can be classified under the two categories based on their invasiveness, non-invasive and invasive. Invasive techniques further can be categorized to technologies using intra–cortical electrode arrays and electrocorticography (ECoG, intracranial electroencephalography). Non-invasive techniques also can be further categorized into EEG, near infrared spectroscopy (NIRS), functional magnetic resoncance imaging (fMRI), positron emission tomography (PET) and magnetoencephalography (MEG).

Invasive methods require complex craniotomy surgery. Taking that into account, testing with them is not a viable option. Out of the non-invasive methods EEG is more widely used compared to MEG, mainly because EEG is inexpensive and simple, instead MEG is expensive, cumbersome and is only used in research settings.

1.1.1.1 Electroencephalography

Electroencephalography (EEG) is an electrophysiological monitoring method to record electrical activity of the brain. EEG measures the electrical activity fluctuations of the brain from the scalp of the human head. Each electrode is called EEG channel. In Figure 1.2 theres an example of an EEG signal (time axis is in seconds and five seconds are displayed. The eight channels provided by this device are shown). Abnormalities of EEG are used to diagnose various neurological disorders, sleep disorders, brain death, tumors, stroke, etc... in clinical applications. However, due to the easiness of use, noninvasive nature and higher temporal resolution compared to other noninvasive BSAT, EEG draws more attention to be used as BCI modality frequently. Despite, EEG recording are prone to contaminate with noise. [6] [7]

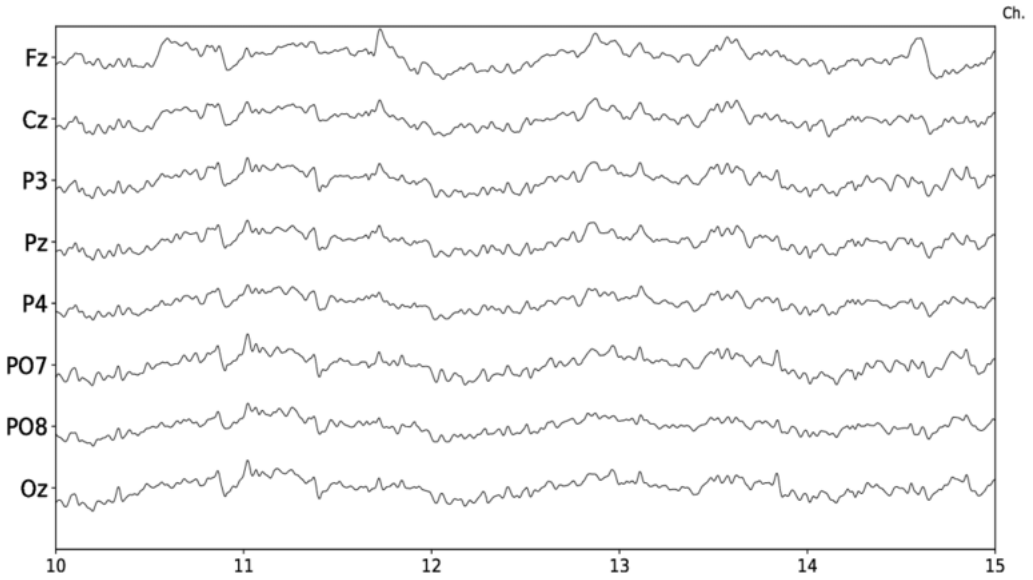


Figure 1.2: Sample EEG signal obtained from (g.Nautilus, g.Tec, Austria) [8]

The work uses EEG signals from different observational human critics (OHCs) who were asked to observe how a computer game was solved while the EEG signals emitted by the subject were captured, and then through the ErrPs evoked from said recordings, train an agent to solve the game in question. Observational human critics are silent subjects

observing a computer gaming agent playing the game.

1.1.2 Signal Processing

Signal processing means to mathematically manipulate an information signal to modify or improve it in some way. In order to achieve the desired objective of the BCI, which is to carry out a particular action based on results, it is almost always necessary to perform some type of processing on the data. In the first instance, there is a pre-processing stage, in which all unwanted or junk information is eliminated, such as interference, noise or artifacts. The difference between noise and artifacts is that noise is caused due to background neurological activity where artifacts are not i.e. eye blinks. This can be done in different ways, among which bandpass filters, frequency filters, spatial filters, among others, stand out. [9] Once this is done, you can proceed to the feature extraction stage. Feature extraction is the process of distinguishing the pertinent signal characteristics from extraneous content and representing them in a compact and/or meaningful form, amenable to interpretation by a human or computer [2]. This means that based on detection and separation of intrinsic characteristics of a signal, with the application of a relevant translation the command for device control can be generated

1.1.3 Spatial Filtering

As mentioned before, EEG signals are weak, easily contaminated by interference and noise, non-stationary for the same subject, and varying across different subjects and sessions. Therefore, it is important to build a model that allows to improve the signals during the different sessions, for different devices and tasks. To achieve an improvement in the signal and obtain a better detection of the ErrPs, filters and algorithms are usually applied that remove noise and interference from the signals, such as the band-pass filter [10], xDawn algorithm [11], among others.

1.1.4 Signal Classification

The goal of classification is to translate the signal features provided by the feature extractor into commands or orders that carry out user's intent. This is done by processing the feature vector that returns the recognized command as output. In order to discern a

command, the model must be trained in such a way that it can differentiate what a feature vector represents. In EEG signals it is possible to detect traces (voltage fluctuations) related to the existence of events in a given time. These events are called event-related potentials (ERP). The events that trigger ERP signals could be perceptual, cognitive or motor events, in response to an activity that reflects the brain activity (Ferrez, 2007) and a model can be made to detect them. In particular, the study in question deals with a particular type of ERP called Error-related potentials (ErrP).

1.1.4.1 Error Related Potential

ErrP [12] occurs in different situations, so it is possible to say that there are different types of ErrP, for example, interaction ErrP, which is evoked by recognizing an error during the interaction between human and machine, feedback ErrP, which is obtained by recognizing an error that is made conscious by feedback presented to the human, response ErrP, which is generated by recognizing the person's own error who is performing a task that requires quick response, or observation ErrP, which is evoked, while observing an erroneous action of a subject. The last one is the one that will be used in this investigation.

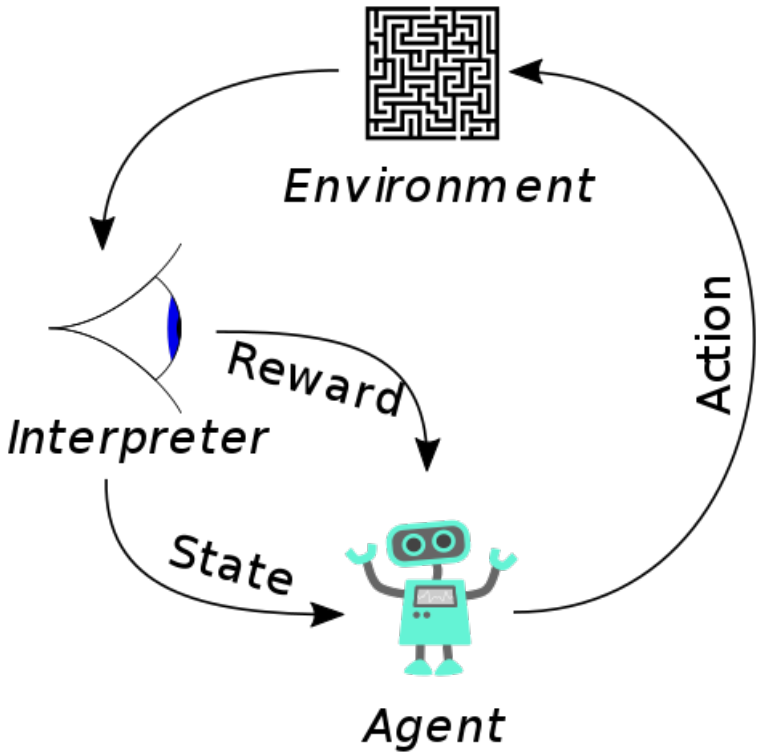


Figure 1.3: Reinforcement learning diagram

1.2 Reinforcement Learning

Single-trial detections of ErrPs are possible by using machine learning techniques and signal processing methods [13], which has been demonstrated in various application areas [14] [15]. Reinforcement Learning (RL) is an Artificial Intelligence approach that can be used in issues such as detecting ErrP signals. RL problems require to know how to map situations to actions with the objective of maximize a numerical reward. In an essential way they are closed-loop problems because the learning system's actions influence its later inputs. [16] These RL systems are widely used in the field of video games, especially in games that respond to stochastic Markov decision processes.

Recently, this technique has been used extensively in the context of advances in artificial intelligence [17], like in 2019, where OpenAI Five became the first AI system to defeat the world champions at an esports game. By defeating the Dota 2 world champion (Team OG), OpenAI Five demonstrates that self-play reinforcement learning can achieve superhuman performance on a difficult task. [18] Papers like [19] and [20] demonstrate that it is possible to train robots with the brain to perform specific tasks. In these papers they use RL in conjunction with ErrPs signals registered by a BCI system based on EEG, which through the stimuli obtained generate implicit feedback in the form of a reward for the RL. [4] The current study focuses on seeking improvements in classification, by implementing different spatial filters applied to the signals and using variants of classification methods, with the aim of obtaining improvements in the learning of RL.

RL is based upon the concept of learning from interaction, automatizing decision making and goal directed learning. Setting a numerical reward for every action given a particular state and maximizing such reward as the objective. However the best actions to take are not explicitly indicated and the agent must discover which path to take in order to maximize the reward. The learner must discover new states in addition to states it has already explored. If not it may be possible to lose important information which might lead to better rewards. When the agent chooses an action it has done in the past that have granted high rewards, it is exploiting previous experiences, nevertheless the agent also needs to explore new paths. The agent must try countless set of actions and prioritize the best that have appear in previous experiences. This method allows us to avoid the problems of creating an explicit model for the behaviour.

1.2.1 Elements

The two main elements of RL are the environment and the agent. In addition to these the other four elements are the policy, the reward system, the value function and the model for the environment.

The agent is the entity capable of perceiving its environment and reacting accordingly in a rational way, trying to maximize the final result or reward.

1.3 Q-Learning

Q-Learning [21] is a variant of model-free reinforcement learning algorithm where an agent receives rewards or penalties to achieve the proper evaluation at a particular state of the consequences of their actions. To represent this rewards a matrix known as Q-Table is used where rows represent all possible states and columns correlate with all possible actions. Q-Table is formally defined as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma * \max_{\tilde{a}} Q(\tilde{s},\tilde{a}) - Q(s,a)] \tag{1.1}$$

The algorithm randomly chooses the action to be taken and updates the Q-Table iteratively based on the reward r obtained by the result of the equation. s is the current state, a the action, α the learning rate, and γ as the discount factor settling the importance of long term against immediate rewards. Formerly when the environment has been explored and the Q-Table acquired, the action for a given state must be the one maximizing the expected reward. At first the Q-Table is initialized with zeros, unless a pre-existing table is given as an argument. To learn from the feedback generated by the subject, the Q-Table is ignored by the agent and it is not used to choose the action at a particular state. Instead the action to achieve in a given state is determined by the agent's actions taken from the brainwave session results. Using this feedback implies the reward is subject directly and only to the brain activity. The Q-Table is updated in each iteration and its agent's performance tested once it finishes the iteration through all training cases. The following pseudo-code shows how the Q-table is trained

Algorithm 1 Q-Learning

Require: step size $\alpha \in (0, 1], small > 0$

Initialize $Q(s, a)$ for all $s \in S, a \in A(s)$ arbitrary except that $Q(terminal, .) = 0$
$Reward \leftarrow 0$
for each *episode* **do** Initialize S
    while S not *terminal* **do**
        Choose A from S using policy derived from Q
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \in max_{\mathrm{a}} Q(S', a) - Q(S, A)]$
        $S \leftarrow S'$
    end while
end for

Figure 1.4: Q-Table Training Algorithm

1.4 Online

Many of the studies that are carried out on BCI are carried out offline. [22] This means that the processing and learning part can be temporarily disconnected with the data collection part. In the day to day, and as a solution to a problem in the real world, although this type of studies work as a base or complement, they are not useful enough. Suppose you want to implement a system in which an agent moves a robotic arm based on the brain signals of a human. In the first instance, following the structure mentioned above, you should take training data initially, repeatedly, until you have enough. The second step would be to process the signals and classify them. Only then can training be carried out so that, the next time the device is used, it works as expected. Then, some problems arise, some more explicit and visible than others and that an online version of the same procedure comes to solve. The first is that the amount of data that is collected depends purely and exclusively on the training sessions. The worst case is that the data that can be obtained while the system is being used, is discarded. It is well known that with algorithms such as reinforcement learning, it is a matter of time for better results to be generated with training. This implies that much information that could be used is being wasted. If, on the contrary, it is decided to store these data, then the problem

is transformed. In the first place, one should begin to take into account the issue of storage memory, which although it seems smaller, it is not, taking into account the size of the EEG signal records. And this is closely related to the second point, how often the processing and training will be carried out. Doing them too often can be tedious; and the more time is left between them, not only implies more memory, but also the existence of a loss of efficiency in those periods of time. If the data taken at time t = 0 were used as feedback at that instant for the data taken at t = 1, learning would be on average faster. Another point to highlight is the ease of adaptation that arises in an online system. A bit based on the same idea, if for some reason the rules of the game change, with online learning, that adaptation is done more quickly. That being said, other approach of the following study is to analyze how an agent behaves when training it online, it seeks to know what is the degree of learning that can be obtained compared to training it offline. In summary, this type of online learning is more practical in daily use than offline learning, and it allows more quickly to obtain a greater number of tests since the agent can be trained while performing the EEG recording sessions.

1.5 Contributions

1. Analysis of processing pipelines to identify single-trial ErrP. This includes: classification algorithms and signal filters.

2. Scheme to perform the online procedure of the method.

3. A method to simulate the online situation (which could not have been conducted due to the pandemic).

2 Materials And Methods

The experimental procedure is represented in Figure 2.1. The system consists of extracting brain signals from an OHC that is watching an agent play a game. The agent knows how the game works, but does not know how to play it, and therefore, can make mistakes. The subject emits an error signal when perceiving erroneous movements of the agent. Based on these responses and under a learning mechanism, the agent improves his performance. The experimental procedure can be divided in two sub-procedures: training a classifier (Offline) and training an IA (Online). At first, Brainwaves were obtained by OpenVibe Server from an OHC. The Game Manager was responsible for generating the game screen, the game mechanics, and the game movements performed by the gaming agent. It was also connected to the Acquisition Server to send stimulus information. The captured information was stored by the OpenVibe Designer and by the Game Manager. Offline: Saved EEG signals are classified. Online: Saved EEG are streamed to the OpenVibe Acquisition Server. The Game Replicator is responsible for replicating saved game movements performed by the gaming agent. The captured information is received and sent by the OpenVibe Designer. After training a classifier offline, then the experiment is repeated and, by linking signals and stimulus to game movements, rewards are calculated. This information is used by a Reinforcement Learning algorithm that iteratively trains a Q-Table in order to improve the performance of the agent that plays the game.

2.1 Brainwave Session

Due to the context of a global pandemic, taking data from subjects in real time was practically impossible. However, this study was already in process and it was decided to continue with it, making some adaptations. The data used in this project was borrowed under the approval of the Departamento de Investigacion y Doctorado from the Instituto Tecnologico de Buenos Aires (ITBA), and the authors of a similar previous study [1] who had the possibility to record it. The retrieval of the OHC's brain activity, called brainwave session, was done voluntarily and anonymously. A brainwave session consists of several matches, each one being a game play. The brainwaves sessions involved 8 subjects, 5 males and 3 females, with an average age of 25.12 years, a standard deviation of 1.54

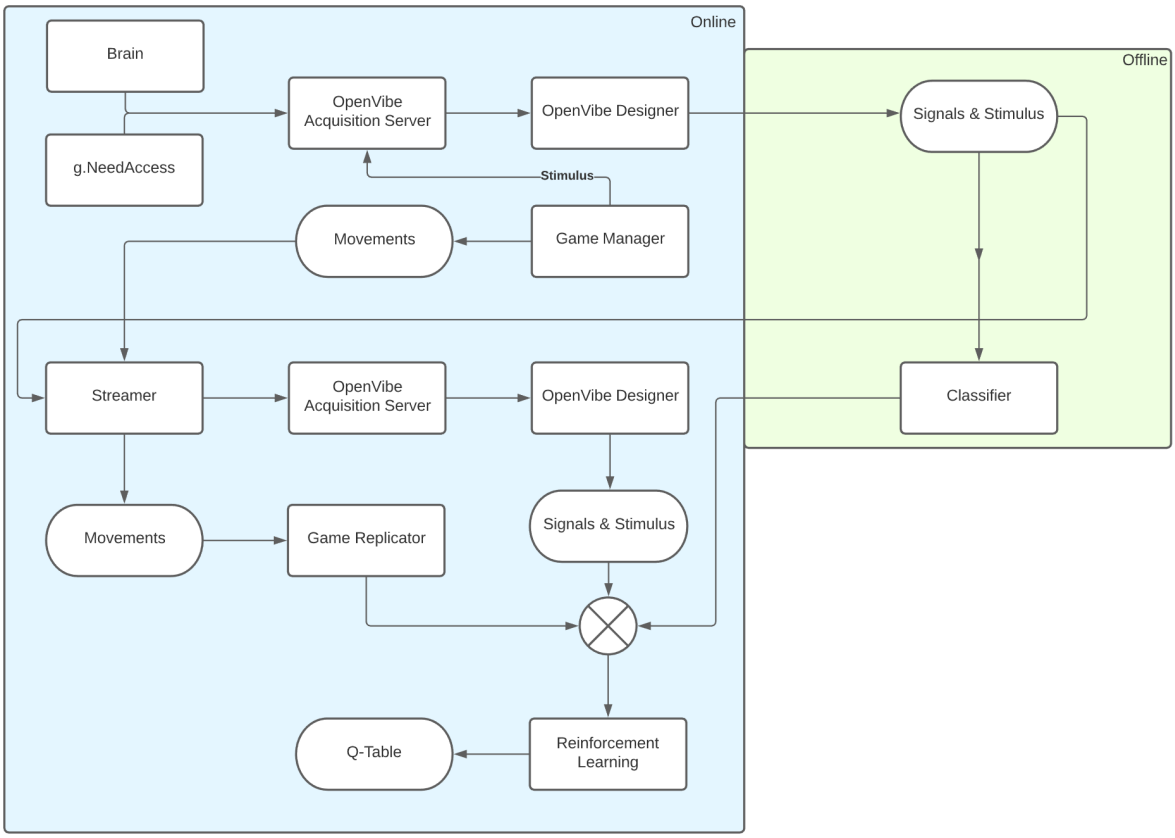


Figure 2.1: Overview of the experimental procedure

years, and a range of 22-28 years. All subjects had normal vision, were right-handed and no history of neurological disorders. They were only told that the objective of the agent was to reach the goal and the four movements that the agent can make. To capture brain signals, a wireless digital EEG device (g.Nautilus, g.Tec, Austria) that is worn by the subject. It had eight electrodes (g.LADYbird, g.Tec, Austria) on the positions Fz, Cz, Pz, Oz, P3, P4, PO7, and PO8, identified according to the 10-20 International System, with a reference set to the right ear lobe and ground set as the AFz position. The electrode contact points were adjusted applying conductive gel until the impedance values displayed by the program g.NeedAccess (g.Tec, Austria) were within the desired range. OpenVibe Acquisition Server program, from the OpenVibe platform [23], was launched and configured with a sampling rate of 250 Hz. A 50 Hz notch filter was applied to filter out power line noise. An additional band-pass filter between 0.5 Hz and 60 Hz was applied. Data was handled and processed with the OpenVibe Designer, from the same platform, using 8 channels for the brain data (one channel per electrode) and an additional channel to record the stimulus, which corresponds to a game movement performed by the agent. To replicate this situation, the stored data is collected through a program

and streamed via lab streaming layer (LSL) [24], which is a system that handles the networking, time-synchronization and real-time access for time series measured in research experiments. This stream is received by the OpenVibe Server and sent to the OpenVibe Designer as if it was in real time. From there, the data is transmitted by the same mechanism and received by a client that will be responsible for carrying out the necessary processing, so that it can be used in the learning of the agent.

2.2 Cognitive Game Procedure

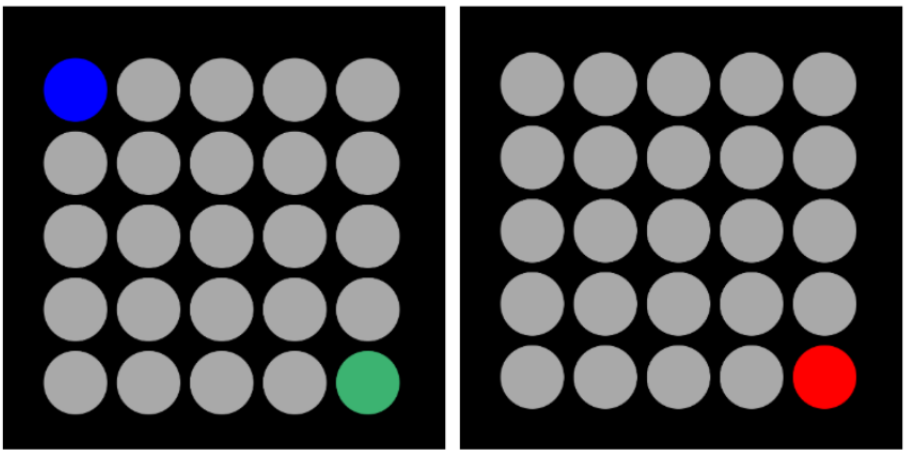


Figure 2.2: Representation of the grid system used in the cognitive game and replicated later

In correlation with the fact that the data obtained from the subjects (EEG and stimuli) is associated with a specific game, in order for the learning to work, it is also necessary to replicate both the game and the particular movements made by the agent in each of the games. The game used consisted of a grid of 5x5 2.2 circles where one indicates the current position of the agent and another the target to be reached. The blue spot represents the initial location while the green spot represents the target location. Once the agent reaches the target spot, its color turns red to indicate the end of the play, and after a few seconds, the game restarts. Similar to the one used in [25]. Both the initial position of the agent (1; 1) [row; column], and the position of the target (5; 5) are static in all games. The different states of the game are determined by the movements of the agent. These can be in four directions: right, left, up and down (with the restriction that it cannot left the grid). The agent performs a movement every two seconds. In the first instances of the experiment, the direction is chosen randomly; then, when testing the learning, it makes

use of a q-table. Every time the game state changed, the Game manager would send a signal to OpenVibe indicating that a stimuli to the OHC had occurred. The state changes were visually abrupt, in order to trigger the ErrP. To replicate this situation, the stored data is read by the Game Replicator which works exactly as the Game Manager regarding the execution of the game, with the difference that the movements that are made are those previously made by the agent (there is no randomness here). Using transmission control protocol (TCP) [26], which is a connection-oriented communications protocol that allows the exchange of messages between devices in a network, the data is transmitted and received by the same client mentioned before.

2.3 Signal Processing

The main idea consists of the detection of the ErrP within the signals. For this, a pipeline is established in which the signals received from OpenVibe are processed, followed by a classifier that will allow distinguishing whether or not there was an ErrP. [27] It is developed in Python using the MNE software platform, for exploring, visualizing, and analyzing human neurophysiological data such as EEG [28] [29] [30], which is build upon the machine learning library Scikit-Learn [31].

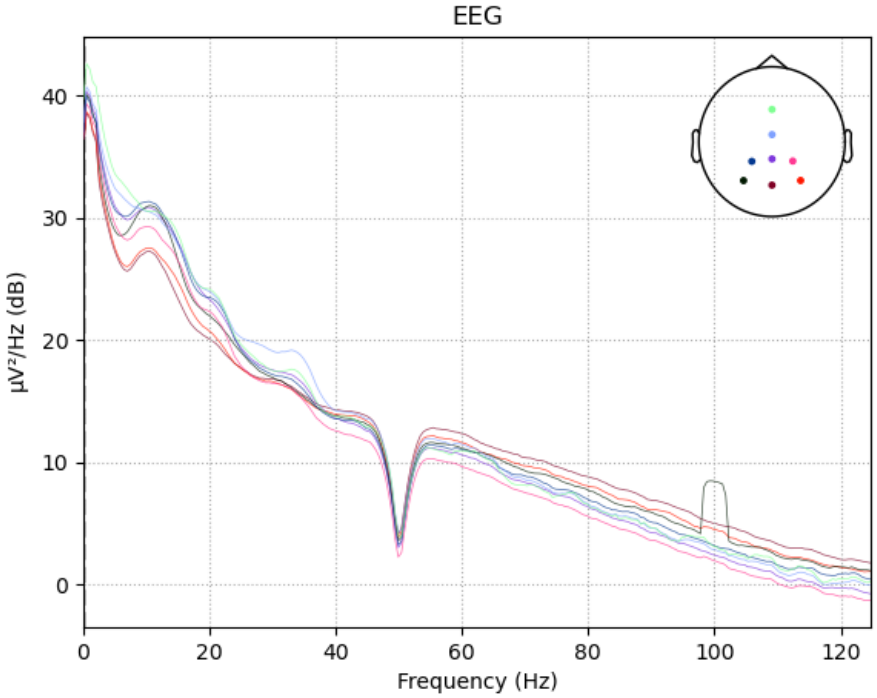


Figure 2.3: Power spectral density (PSD) of a raw dataset smoothed.

The pipeline used is the same in the online part as in the offline part and has the following structure. First, the result of the brainwave session is received as raw data. An additional 0.5-10.0 Hz band-pass filter is immediately applied to the signal, followed by a Notch filter of 50 Hz to eliminate the fixed frequency of the power line.

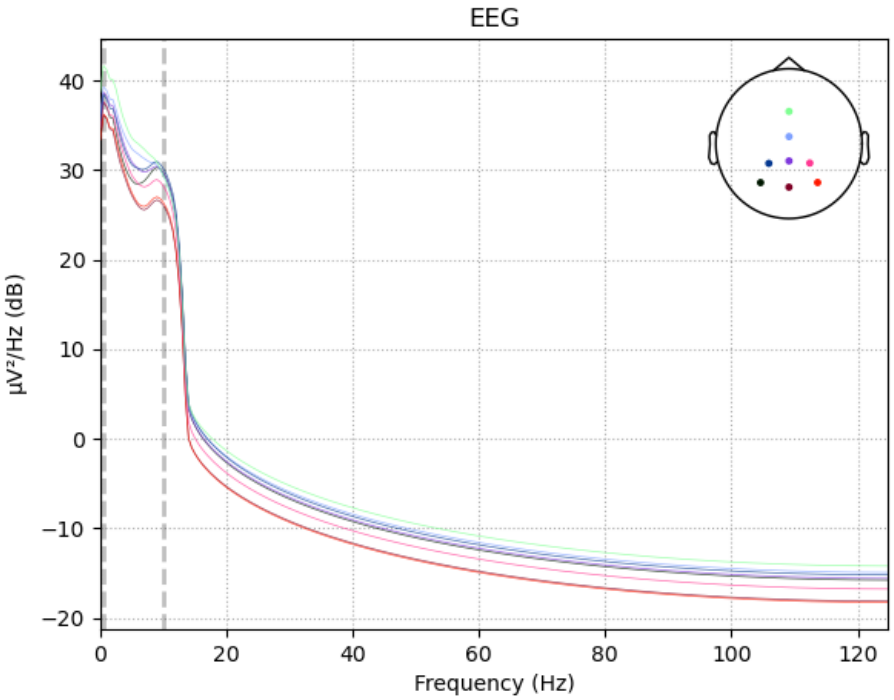


Figure 2.4: Power spectral density (PSD) of a raw dataset after applying a band-pass 0.5-10.0 Hz filter smoothed.

2.4 Segmentation

The next step is to create the epochs. An epoch refers to the succession of samples in a period of time of 2 seconds, from the moment an event occurs. To split the data in epochs, the samples corresponding to the start of an event are tagged using the information received in the extra channel that sends the stimulus. Considering that the sampling frequency is 250 Hz, the epochs turn out to be structures of 500 samples each. Taking into account that each sample contains 8 channels, the result is a 500x8 matrix.

Each game consists of an array of epochs, the length of which is variable depending on the duration of it. The number of games depends on the samples that have been taken from each subject.

All data that is not associated with a particular game, that is, the one received between

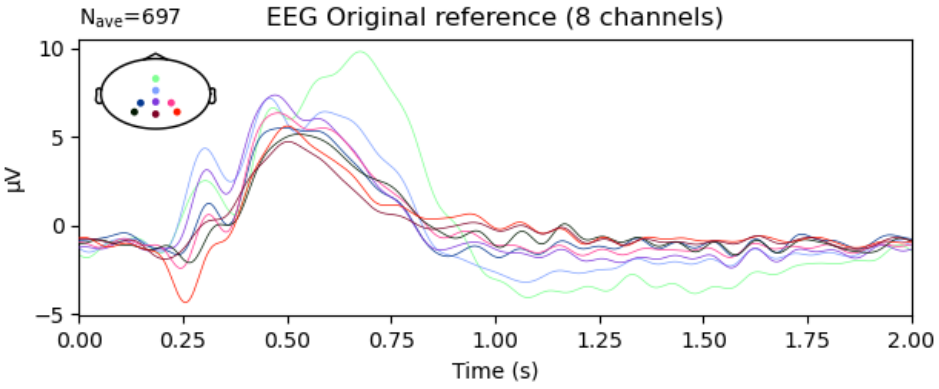


Figure 2.5: Grand avarage signal values of all epochs types of a measured dataset.

several successive games or at the beginning or end of a brainwave session is discarded.

2.5 Spatial Filtering

There exists a typical response synchronized with the target stimuli, and then this synchronous response can be enhanced by a spatial filtering, i.e, in order to eliminate noise from signals [32], spatial filters [33] [34] are applied. Tests are performed with combinations of two spatial filters, PCA and XDawn.

2.5.1 PCA

PCA is defined as an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some scalar projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. [35]

Let $x_{\mathrm{j}}(t)$ denote the EEG signal recorded by the j^{th} sensor at time index t and let $X \in \mathbb{R}^{\mathrm{N_t} \times \mathrm{N_s}}$ be the matrix of recorded EEG signals whose $(i,j)^{\mathrm{th}}$ entry is $x_{\mathrm{j}}(i)$. N_{s} is the number of sensors and N_{t} the number of temporal samples. The sample mean of each column has been shifted to zero.

The transformation is defined by a set of l p-dimensional vectors of weights $\mathbf{w}_{(k)} = (w_1, \ldots, w_p)_{(k)}$ that map each row vector $\mathbf{x}_{(i)}$ of X to a new vector of principal component scores $\mathbf{t}_{(i)} = (t_1, \ldots, t_l)_{(i)}$, given by $t_{k(i)} = \mathbf{x}_{(i)} \cdot \mathbf{w}_{(k)} \qquad \text{for} \qquad i = 1, \ldots, n \qquad k = 1, \ldots, l$ in such a way that the individual

variables $t_1, \ldots, t_\ell$ of t considered over the data set successively inherit the maximum possible variance from X, with each coefficient vector w constrained to be a unit vector (where ℓ is usually selected to be less than p to reduce dimensionality).

The first weight vector $w_{(1)}$ has to satisfy

$$\mathbf{w}_{(1)} = \underset{\|w\|=1}{\operatorname{argmax}} \left\{ \sum_i (t_1)^2_{(i)} \right\} = \underset{\|w\|=1}{\operatorname{argmax}} \left\{ \sum_i (\mathbf{x}_{(i)} \cdot \mathbf{w})^2 \right\} \tag{2.1}$$

Equivalently, writing this in matrix form gives,

$$\mathbf{w}_{(1)} = \underset{\|w\|=1}{\operatorname{argmax}} \{ \|\mathbf{X}\mathbf{w}\|^2 \} = \underset{\|w\|=1}{\operatorname{argmax}} \{ \mathbf{w}^T \mathbf{X}^\mathbf{T} \mathbf{X} \mathbf{w} \} \tag{2.2}$$

in order to maximize variance, which equivalently satisfiesWx

$$\mathbf{w}_{(1)} = \arg\max \left\{ \frac{\mathbf{w}^T \mathbf{X}^\mathbf{T} \mathbf{X} \mathbf{w}}{\mathbf{w}^T \mathbf{w}} \right\} \tag{2.3}$$

since $w_{(1)}$ has been defined to be a unit vector.

The first principal component of a data vector $x_{(i)}$ can then be given as a score $t_{1(i)} = x_{(i)} \cdot w_{(i)}$ in the transformed co-ordinates, or as the corresponding vector in the original variables, $\{x_{(i)} \cdot w_{(i)}\} w_{(i)}$.

The k^{th} component can be found by subtracting the first $k-1$ principal components from X:

$$\hat{\mathbf{X}}_k = \mathbf{X} - \sum_{s=1}^{k-1} \mathbf{X} \mathbf{w}_{(s)} \mathbf{w}_{(s)}^{\mathrm{T}} \tag{2.4}$$

and then finding the weight vector which extracts the maximum variance from this new data matrix

$$\mathbf{w}_{(k)} = \underset{\|w\|=1}{\operatorname{argmax}} \left\{ \|\hat{\mathbf{X}}_k \mathbf{w}\|^2 \right\} = \operatorname{argmax} \left\{ \frac{\mathbf{w}^T \hat{\mathbf{X}}_k^T \hat{\mathbf{X}}_k \mathbf{w}}{\mathbf{w}^T \mathbf{w}} \right\} \tag{2.5}$$

The k^{th} principal component of a data vector $x_{(i)}$ can therefore be given as a score $t_{k(i)} = x_{(i)} \cdot w_{(k)})$ in the transformed co-ordinates, or as the corresponding vector in the space of the original variables, $x_{(i)} \cdot w_{(k)}) w_{(k)}$, where $w_{(k)}$ is the k^{th} eigenvector of $X^{\mathrm{t}}X$.

The full principal components decomposition of X can therefore be given as $\mathbf{T} = \mathbf{X}\mathbf{W}$ where W is a p-by-p matrix of weights whose columns are the eigenvectors of $X^{\mathrm{t}}X$.

2.5.2 xDawn

The xDAWN [11] algorithm designs spatial filters to enhance the discrimination between signal and noise and reduce the dimension of the EEG signals Let $x_{\mathrm{j}}(t)$ denote the EEG signal recorded by the j^{th} sensor at time index t and let $X \in \mathbb{R}^{\mathrm{N_t} \times \mathrm{N_s}}$ be the matrix of recorded EEG signals whose $(i,j)^{\mathrm{th}}$ entry is $x_{\mathrm{j}}(i)$. N_{s} is the number of sensors and N_{t} the number of temporal samples. The sample mean of each column has been shifted to zero. Let $A \in \mathbb{R}^{\mathrm{N^e}} N_{\mathrm{s}}$ be the matrix of ErrP signals whose $(i,j)^{\mathrm{th}}$ entry is $a_{\mathrm{j}}(i)$. N_{e} is the number of temporal samples of the ErrP. The observation X can then be modeled by:

$$X = DA + N \tag{2.6}$$

$X \in \mathbb{R}^{\mathrm{N_t} \times \mathrm{N_e}}$ is a Toeplitz matrix that represents the time position of the ErrP in the observations whose first column is defined such that $D_{\mathrm{Tk,1}} = 1$, where Tk is the stimulus onset of k^{th} target stimulus ($1 \leq k \leq K$, with K being the total number of target stimuli) and such that all the other elements are null. $N \in \mathbb{R}^{\mathrm{N_t} \times \mathrm{N_c}}$ is a noise matrix. A is defined by:

$$\tilde{A} = \underset{A}{\operatorname{argmin}} ||X - DA||^2 = (D^T D)^{-1} D^T X \tag{2.7}$$

where $.^{\mathrm{T}}$ is the transpose operator. If there is no overlap between trials, $\tilde{A}$ is equal to the averaged signal ($D^{\mathrm{T}} A$). The next step of xDawn method consists of estimating N_{f} spatial filters u_{i} ($1 \leq i \leq N_{\mathrm{f}} \leq N_{\mathrm{s}}$) such that the synchronous response is enhanced by the spatial filtering:

$$XU = DAU + NU \tag{2.8}$$

where $U \in \mathbb{R}^{\mathrm{N_s} \times \mathrm{N_f}}$ are the spatial filters matrix whose i^{th} column is u_{i}. The filters are designed in such a way that U maximizes the signal-to-signal plus noise ratio (SSNR) with the generalized Rayleigh quotient:

$$\tilde{U} = \underset{U}{\operatorname{argmax}} \frac{Tr(U^T \tilde{A}^T D^T D \tilde{A} U)}{Tr(U^T X^T X U)} \tag{2.9}$$

where $Tr(\cdot)$ is the trace operator. This optimization problem can be solved by a QR factorization with a singular value decomposition.

2.6 Classification

The epochs can be classified according to their type in two: hit or no-hit. The former correspond to those in which the agent executes an action that takes him away from the goal. In the latter, on the contrary, the agent's action brings him closer to the goal. ErrP should be detected in hits. Then, objective of the classifier is to be able to differentiate the hits from the no-hits. To get the data ready for classification, the stimulus channel is removed to classify the signals using only the EEG data. The eight channels are concatenated using the MNE Vectorizer function, which transforms the data matrix into 2D array. Each epoch is regularized using a MinMaxScaler, which is scaling each feature to a given range by subtracting the minimum value in the epoch and dividing by the signal peak-to-peak amplitude [36].

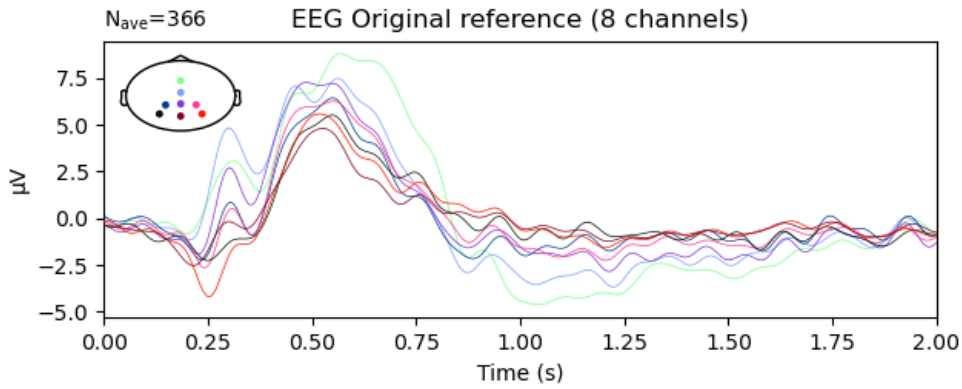


Figure 2.6: Grand avarage signal values of epochs that represent a hit.

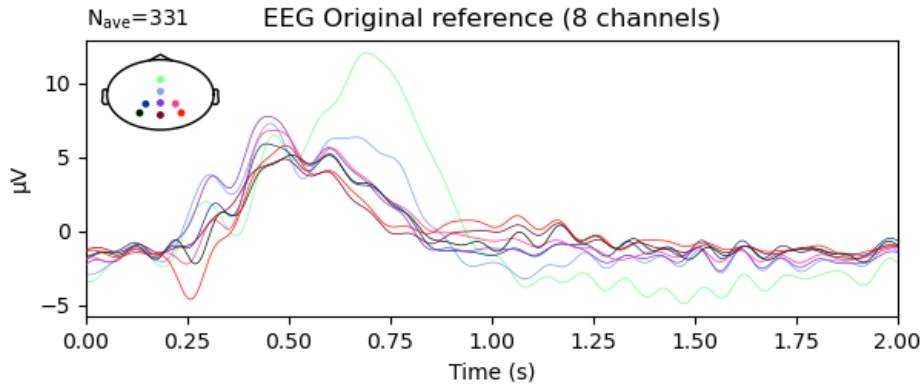


Figure 2.7: Grand avarage signal values of epochs that represent a no-hit.

The classification algorithms used are Logistic Regression [**Logistic-Regression**] and a linear kernel Support Vector Classifier (i.e. SVM) [**Linear-SVC**] . The idea is to compare

different configurations of these two algorithms and to be able to detect which of them best suits the problem to be treated.

2.7 Cross Validation

Games are divided into two sets. The first one will be used to train the classifier to identify the ErrP within the signals. The second has two different uses: testing the classifier separately and the online part of the project, that is, the games that are streamed and with which reinforcement learning will be carried out. The size of each of the sets depends on the number of games recorded. Ideally, if this is an even number, it is decided to divide it in two, having the same size for both sets ($50\% - 50$). Otherwise, the testing set is bigger by 1.

By partitioning the available data into sets, the number of samples which can be used for learning the model is drastically reduced, and the results can depend on a particular random choice for the pair of (train, testing) sets. This phenomenon is even stronger in cases like the one presented in this study, where the amount of data is scarce. In order to eliminate any bias that may arise in relation to the selection of data, two cross validation methods were used. These are: K-Fold and Shuffle Split whose implementations taken from Scikit-Learn [31]. The classifier was trained and tested with both methods, but only the latter was used to train the agent, since K-fold does not allow to split the data in half. 2.8 represents visually how these two cross validation methods work. In the K-Fold example, the number of splits k=4. In the ShuffleSplit k=4, and testing set size n=0.15 * total size. Note that neither of them is affected by classes or groups.

K-Fold procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into and is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups.
3. For each unique group:
   (a) Take the group as a hold out or test data set
   (b) Take the remaining groups as a training data set

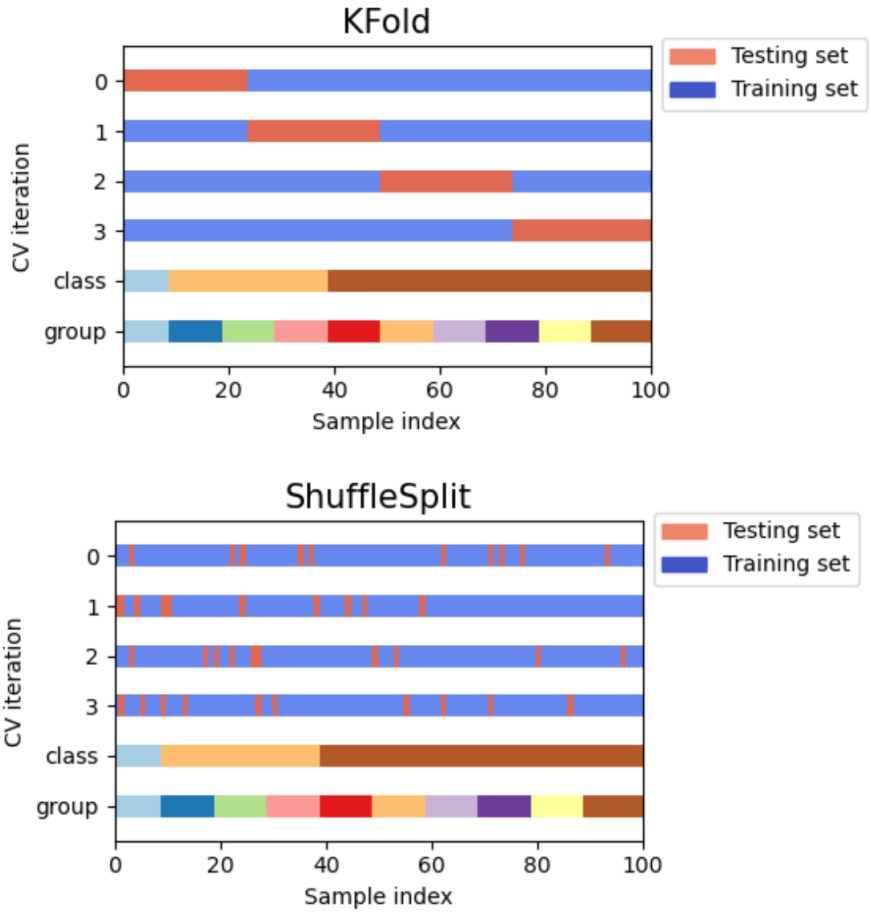


Figure 2.8: Visualization of both of the cross-validation methods.

(c) Fit a model on the training set and evaluate it on the test set

(d) Retain the evaluation score and discard the model

4. Summarize the skill of the model using the sample of model evaluation scores

ShuffleSplit is a good alternative to KFold cross validation since it allows a finer control on the number of iterations and the proportion of samples on each side of the train/test split. The main difference then, is that the size of the testing set n can be fixed. The procedure is as follows:

1. Shuffle the dataset randomly.

2. Split the dataset into k groups.

3. For each unique group:

(a) Take the group + $(n-1)$ groups randomly from the remaining groups as a

hold out or test data set

(b) Take the remaining $(total - n)$ groups as a training data set

(c) Fit a model on the training set and evaluate it on the test set

(d) Retain the evaluation score and discard the model

4. Summarize the skill of the model using the sample of model evaluation scores

2.7.1 Calibration and Selection

The algorithm calibration consists of finding the set of parameters that gives the best results when classifying. In machine learning, this technique is called hyperparameter optimization. The same kind of machine learning model can require different constraints, weights or learning rates to generalize different data patterns. These measures are called hyperparameters, and have to be tuned so that the model can optimally solve the machine learning problem. The most traditional way of doing it is called grid search, which basically consists of doing an exhaustive search on a manually specified subset of the hyperparameter space. To calculate the scores for each configuration, a k-fold cross-validation process is carried out, which guarantees the training capacity of the training data set. Each setting is scored 10 times. The final score for each configuration is taken as the average of these 10 scores. [37] The configuration with the best score is selected for each subject and each classification algorithm. From this, the best configuration results for each algorithm for each subject are compared and the one with the highest score is selected for further calculations

2.7.2 Comparison

Having the best configuration for each classification algorithm, the classification can be performed. As mentioned above, for this the test set is used, that is, the data that has not yet been used. This is where the online part comes in.

2.7.3 Performance

In order to measure the performance of the classification algorithms, for each subject there is a confusion matrix and a receiver operating characteristic (ROC) curve. A confusion

matrix is a tool that allows the visualization of the performance of an algorithm that is used in supervised learning. Each column in the matrix represents the number of predictions for each class, while each row represents the instances in the actual class. The classes are hits and no-hits. From this graph you can see two types of errors:

- Type 1 error (false positives): Samples are classified as positive when they are actually negative. In this case, no-hits predicted as hits.
- Type 2 error (false negatives): Samples are classified as negative when they are actually positive. In this case, hits predicted as no-hits

A ROC curve is a graphical representation of sensitivity versus specificity for a binary classifier system as the discrimination threshold is varied. In other words, it is the representation of the ratio of true positives versus the ratio of false positives. If the curve is above the identity function, it means that the classifier has better performance than a random selection. On the contrary, if the curve is below, then the classifier is under-performing. In any case, it is always good to obtain results that are far from the identity (in the latter case it would be enough to invert the classifier's predictions). The goal then is to maximize that distance.

2.8 Reinforcement Learning

With the classifier trained, the received epochs are classified into hits and no hits. Each of the agent's movements is given a reward based on the predictions obtained from the classifier. The reward can either be -1 when the event is classified as a hit or 0 when it is classified as a no-hit. The accuracy of these rewards depends on the performance of the classifier. The sequence of movements with their associated reward is used by the reinforcement learning algorithm to train the agent. The algorithm used is called Q-Learning.

2.9 Q-Learning

The Q-Learning implementation used is the one provided by the OpenAI Gym library. Gym is a toolkit for developing and comparing reinforcement learning algorithms. [38]

2.9.1 Algorithm Implementation

The algorithm is based on the existence of a Q-Table. Initially, it starts with its values at zero. In each iteration, its values are updated. In order to achieve learning, these updates are made taking into account the feedback received from the subject, after experiencing how the agent played the game. The feedback is not explicit, but is obtained after having made the above-mentioned interpretations and classifications of the brain signal data. In other words, the reward is determined based on the classification of an action as error or not, which in turn, is determined based on the brain activities of the subject. It should be noted that the actions carried out by the agent are completely unrelated to the Q-Table. Each action is chosen randomly within all possible ones given a state. This is mainly due to the fact that, given the context, it was not possible to obtain new data from the experiments and that everything is based on recordings of past games. The equation used to update the Q-Table is as follows:

$$Q(state, action) \leftarrow Q(state, action) + \alpha[reward + \gamma * Max(state, action) - Q(state, action)], \tag{2.10}$$

After the algorithm finishes iterating through all the training episodes, the Q-Table is saved. Each experience is one $trainingEpisode$.

2.9.2 Environment

In reinforcement learning, the environment is what defines the different game's states and the possible actions. From the combination of the current state and the action carried out by the agent, the corresponding reward is determined. The representation of the environment is the one mentioned in section 2.2.

An environment contains all the necessary functionality to run an agent and allow it to learn. Creating an environment with Gym is quite simple, it requires the implementation to inherit from Gym's class $Gym.Env$. It is necessary to define the type and shape of the $action_space$, which will contain all of the actions possible for an agent to take in the environment. Similarly, the $observation_space$ is defined, which contains all of the environment's data to be observed by the agent. The main methods to be implemented are render, reset and step. *Render* is used to to print a rendition of the environment (a matrix

representing the state). *Reset* method will is called to periodically reset the environment to an initial state (initial position and amount of steps to zero). *Step* function iterates from one state to the next one. It returns four values . These are:

- Observation (object): an environment-specific object representing your observation of the environment.
- Reward (float): amount of reward achieved by the previous action. The scale varies between environments, but the goal is always to increase your total reward.
- Done (boolean): whether it's time to reset the environment again. Done being True indicates the episode has terminated.
- Amount of steps (integer): Amount of steps since the beginning of the experience.

The custom environment developed is called MentalChaseEnv. The environment is in charge of reading the next state from the file and passing the reward as a parameter to the reinforcement learning algorithm.

2.9.3 Testing Implementation

To get the most out of the online implementation, the best solution would have been for the agent to constantly change its actions based on the Q-table. Having done this, an analysis of the learning speed of the agent could have been made, trying to validate the hypothesis that it learns faster. However, putting this into practice was impossible due to context, and therefore, the only way to test the operation of the the agent's learning algorithm is to do it offline. To do so, a parallel algorithm is developed. In this implementation, the Q-Table is used to determine what action should be taken given a state. The equation that shows the action is the following:

$$action(sort(Q[state, :] + random(actionSpace) * (1/(iterationStep + 1)))) \quad (2.11)$$

An environment called *ChaseEnv* is developed for testing.

The reward function takes into account the distance that exists between the position of the agent and the target. If the distance increased in relation to the previous state, then

the reward is negative. If not, it returns zero. Something important to mention is that the reward function doesn't always return the correct reward, in order to test how the Q-Table is trained when the accuracy is less than 100%.

The following pseudo-code describes the reward function

Algorithm Reward Function

Require: $accuracy \in (0, 1]$

$CurrentDistanceToGoal \leftarrow DistanceToGoal$
$PreviousDistanceToGoal \leftarrow 0$
$Reward \leftarrow 0$
if $CurrentDistanceToGoal > PreviousDistanceToGoal$ **then**
    $Reward \leftarrow -1$
else if $random \in [0, 1] < accuracy$ **then**
    return $Reward$
else
    return $Reward$
end if

Figure 2.9: Reward Calculation for ChaseEnv

2.10 Online

The online flow goes as shown in 2.10. Lsl Receiver is the main program, that after a few steps, receives all the data ready to be processed and after doing it, delegates predictions to the Q-Learning Algorithm.

For the implementation of the online part, the first step was to identify the functionalities provided by OpenVibe. In the ideal case, an Acquisition Server would have been built to take the data from the Nautilus device, but considering that this was not feasible, an alternative had to be found. The proposed alternative continues to use the Acquisition Server as a source of information, but instead of coming directly from the brain, it comes from an LSL stream, and this configuration can be seen in the figure 2.11

The LSL signal is sent by a program developed in Python called LSL Sender, which is responsible for reading the corresponding data files that contain both the EEG signals and the markers. LSL works similar to a publish / subscribe pattern. Having the name of the Stream and the type (that is, if it is an EEG signal or a markers signal), the Acquisition Server "subscribes" to the corresponding stream. Periodically, in real time, the data is sent and received. The next step is identical, both in the online and offline

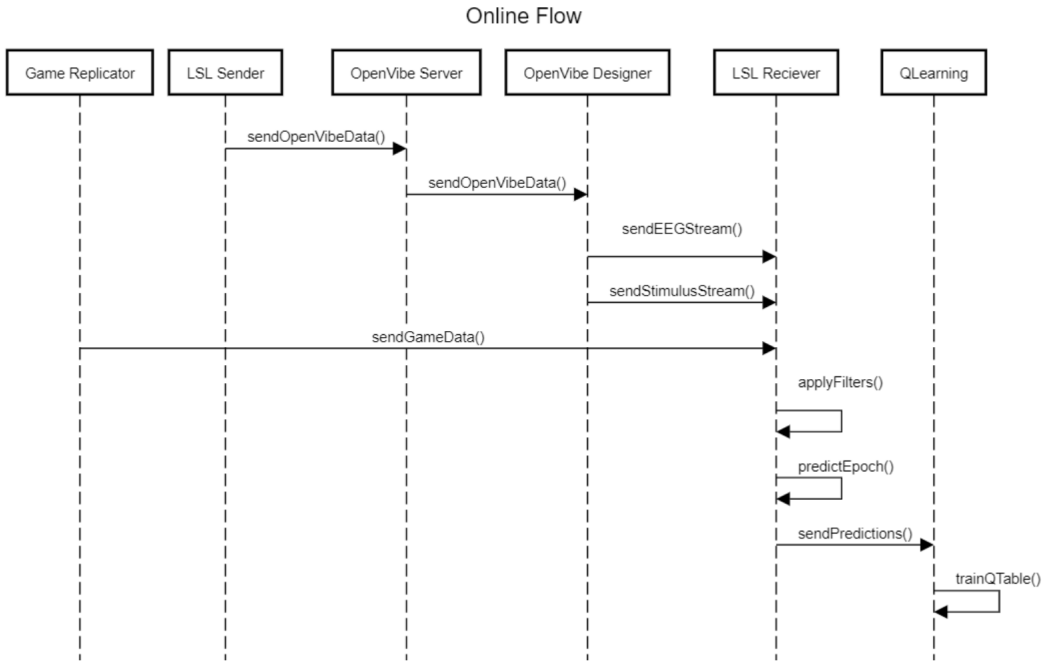


Figure 2.10: Online flow sequence summary diagram

Figure 2.11: OpenVibe Acquisition Server Configuration

implementation, and is the internal connection that the OpenVibe Server makes to the OpenVibe Designer to send the data. This connection is outside the implementation of this work and its management is entirely the responsibility of OpenVibe.

OpenVibe Designer allows to operate with the data received from the server, by creating and executing a scenario. The scheme created for this case is quite simple and can be seen in figure 2.12, since the OpenVibe Designer is expected to only work as a proxy to forward the received signals to a program that will take care of the processing. This program is the LSL Receiver. The LSL connection between the two is equivalent to the previously mentioned connection. The data obtained is divided into two different streams. LSL Export (Gipsa) is used to send EEG signals. LSL Export is used to send the markers

channel.

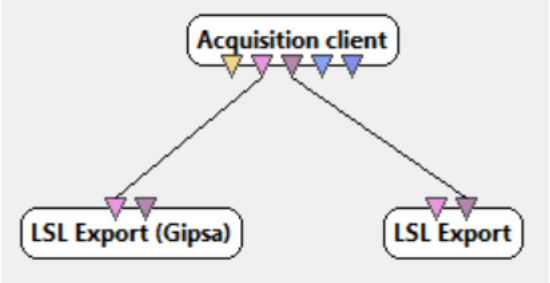


Figure 2.12: OpenVibe Designer Scenario

To associate the signals to the agent's movement, on the other hand, the LSL Receiver must receive the current state of the game. This is sent by Game Replicator. In the first instance, this program, developed in JAVA, reads a file representing a game (it contains initial configuration and consecutive agent positions). With this information it replicates the game in real time and, as it happens, sends the information via TCP. To do this, between the Game Replicator and the LSL Receiver, a previous connection based on this protocol was established on a specific port.

The following pseudo-code describes LSL Receiver main actions, which, after receiving all the raw data, proceedes to filter, segmentate and classificate the signal and train the Q-table.

One point to take into account when carrying out this whole scheme is the synchronization of the signal. For this, the first idea that arose was that it was going to be necessary to make use of the timestamps associated with the signals and markers. However, when analyzing the situation in practice, considering that it works with very low latency, the information was not out of sync. In the case of working with higher latencies, then this addition would have to be done.

Even having the data synchronized, considering that the data comes in separate streams, it is necessary to link it to each other, that is, to associate the EEG signal to the marker. To do this, the moment a marker is received (representing a hit or no hit), an epoch is created with the EEG data. Considering that the sampling frequency is 250Hz, and that the markers are sent every 2 seconds, the epochs are made up of 500 values. In the original data, the amount of samples from the epochs was around that number, but it was not exact, and it depended on how OpenVibe did the cut. For practical purposes it is

Algorithm LSL Receiver

```
1: Classifier ← LinearRegresionClassifier
2: EEGStream ← LSL EEG Stream
3: MarkersStream ← LSL Markers Stream
4: GameState[] ← Game State TCP Client
5: PrevState ← pop(Game State)
6: CurrentState
7: QTables[][]
8: QTable[] ← zeros(games, 4)
9: i ← 0
10: for each Game do
11:     for each Epoch composed by EEGStream delimited by
    MarkerStream do
12:         MNEEpoch ← Apply filters, resample restructure epoch
13:         Predictions[] = Classifier.predict(MNEEpoch)
14:         CurrentState = pop(GameState)
15:         QTables[i] ← QTable.Train(TrainPredictions[], QTable, CurrentSta
16:         PrevState = CurrentState
17:         i = i + 1
18:     end for
19: end for
```

Figure 2.13: Lsl Receiver Algorithm

the same, due to the processing that is carried out later, which includes a resample. Also, the markers's channel is used to divide between games. All the EEG channel information that is not associated with an epoch is discarded.

3 Results

3.1 Training and Test Sets

The number of games and brainwaves sessions available is not homogeneous between each subject. The following table describes the amount of samples that are used throughout the analysis for training and testing sets for each subject.

Subject	Training Set Size	Testing Set Size
1	3	3
2	3	3
3	3	3
4	3	3
5	2	2
6	2	3
7	2	3
8	3	3

Table 3.1: Training and Test Sizes

3.2 Pipelines

In order to determine if the spatial filters are adequate for the detection of error potentials, classification tests were carried out. For each subject the best algorithm was used, and three different pipelines were compared:

- Pipeline 1 (Standard): Vectorizer => MinMaxScaler => Classifier
- Pipeline 2 (PCA): PCA => Vectorizer => MinMaxScaler => Classifier
- Pipeline 2 (xDawn): xDawn => Vectorizer => MinMaxScaler => Classifier

3.3 Signal Classification

3.3.1 Algorithm Calibration

In order to obtain a good detection of ErrP signals, it is necessary to find the best algorithm, this implies the type and its configuration. To do this, different parameters were tested for the two algorithms mentioned above, making use of the GridSearch implementation of

the sklearn library. The metric used to compare is the score, which is based on AUC. The best results were obtained for both xDawn and Standard pipeline.

3.3.1.1 Logistic Regression

For the logistic regression algorithm the parameters that are tested are C and penalty. The values that C can take are 0.001, 0.01, 0.1, 1, 5, 10 and 25. The values that penalty can take are 'l1' and 'l2'. The values that solver can take are 'liblinear', 'lbfgs' and 'saga'.

Logistic Regression		
Subject	Score	Configuration
1	0.65	'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'
2	0.69	'C': 0.001, 'penalty': 'l2', 'solver': 'liblinear'
3	0.69	'C': 5, 'penalty': 'l1', 'solver': 'liblinear'
4	0.68	'C': 5, 'penalty': 'l1', 'solver': 'liblinear'
5	0.58	'C': 0.01, 'penalty': 'l2', 'solver': 'lbfgs'
6	0.68	'C': 10, 'penalty': 'l2', 'solver': 'lbfgs'
7	0.74	'C': 5, 'penalty': 'l2', 'solver': 'liblinear'
8	0.63	'C': 0.001, 'penalty': 'l2', 'solver': 'liblinear'

Table 3.2: Logistic Regression best score per subject - Standard Pipeline.

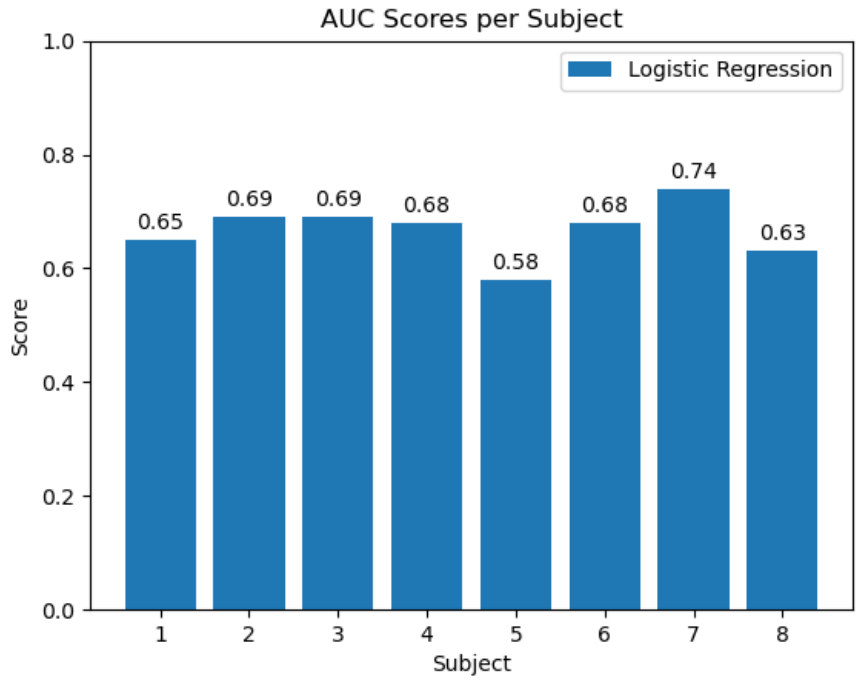


Figure 3.1: Logistic Regresion AUC best score per subject - Standard Pipeline.

Logistic Regression		
Subject	Score	Configuration
1	0.66	'C': 1, 'penalty': 'l1', 'solver': 'liblinear'
2	0.67	'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'
3	0.71	'C': 5, 'penalty': 'l1', 'solver': 'liblinear'
4	0.71	'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'
5	0.63	'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'
6	0.75	'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'
7	0.75	'C': 0.1, 'penalty': 'l2', 'solver': 'saga'
8	0.70	'C': 0.1, 'penalty': 'l2', 'solver': 'saga'

Table 3.3: Support Vector Classifier best score per subject - xDawn.

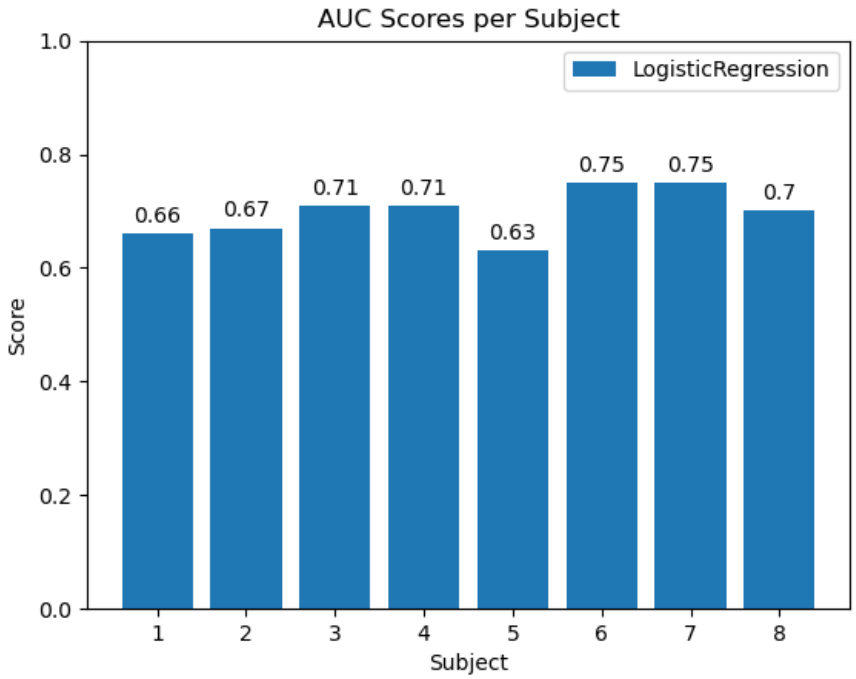


Figure 3.2: Logistic Regresion AUC best score per subject - xDawn Pipeline.

3.3.1.2 Support Vector Classification

For the support vector classifier algorithm the parameters that are tested are C, gamma and kernel. The values that C can take are 1, 10, 100, 1000. The values that gamma can take are 0.0001, 0.001, 0.01, 0.1, 1. The values that kernel takes are 'linear', 'rbf', 'poly' and 'sigmoid'.

Support Vector Classifier		
Subject	Score	Configuration
1	0.65	'C': 1, 'gamma': 0.0001, 'kernel': 'sigmoid'
2	0.68	'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'
3	0.69	'C': 10, 'gamma': 1, 'kernel': 'linear'
4	0.68	'C': 1, 'gamma': 0.1, 'kernel': 'poly'
5	0.61	'C': 1, 'gamma': 1, 'kernel': 'rbf'
6	0.68	'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'
7	0.74	'C': 1, 'gamma': 0.1, 'kernel': 'poly'
8	0.64	'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'

Table 3.4: Support Vector Classifier best score per subject - Standard Pipeline.

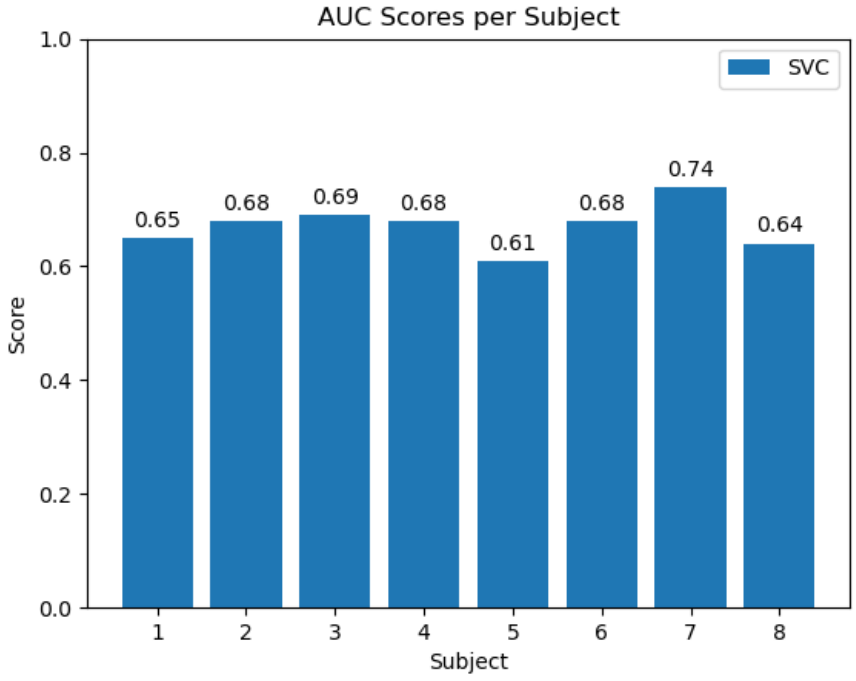


Figure 3.3: SVC AUC best score per subject - Standard Pipeline.

Support Vector Classifier		
Subject	Score	Configuration
1	0.67	'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'
2	0.68	'C': 1, 'gamma': 1, 'kernel': 'poly'
3	0.71	'C': 1, 'gamma': 0.1, 'kernel': 'rbf'
4	0.70	'C': 100, 'gamma': 0.001, 'kernel': 'sigmoid'
5	0.63	'C': 10, 'gamma': 1, 'kernel': 'rbf'
6	0.70	'C': 100, 'gamma': 0.001, 'kernel': 'rbf'
7	0.75	'C': 1, 'gamma': 0.1, 'kernel': 'rbf'
8	0.69	'C': 100, 'gamma': 0.001, 'kernel': 'sigmoid'

Table 3.5: Support Vector Classifier best score per subject - xDawn Pipeline.

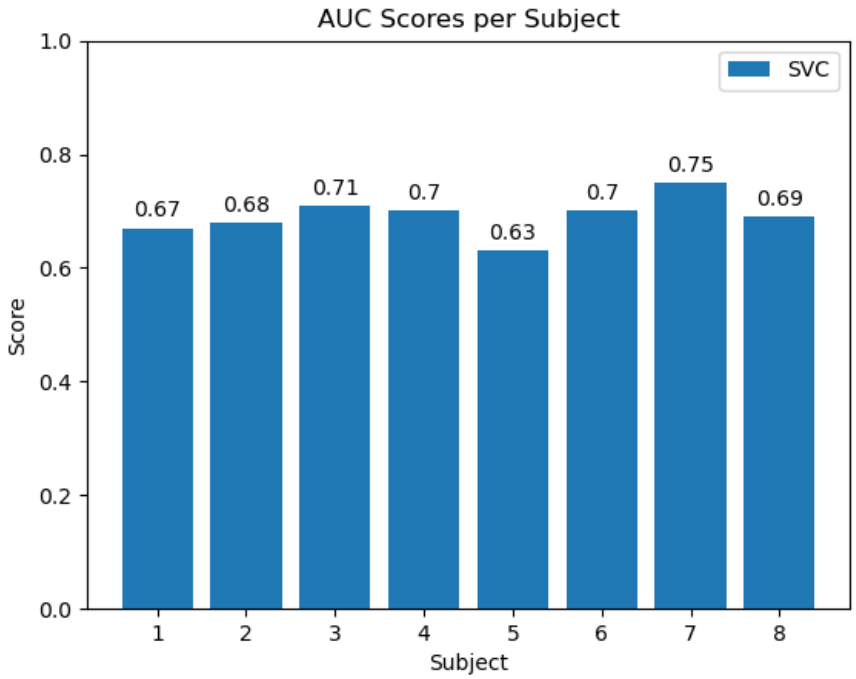


Figure 3.4: SVC AUC best score per subject - xDawn Pipeline.

3.3.2 Optimal Algorithm Selection

The choice of the best algorithm is made by comparing the scores of the best configuration of each algorithm for each subject.

The results are the following:

Subject	Algorithm
1	Support Vector Classifier/Logistic Regression
2	Logistic Regression
3	Support Vector Classifier/Logistic Regression
4	Support Vector Classifier/Logistic Regression
5	Support Vector Classifier
6	Support Vector Classifier/Logistic Regression
7	Support Vector Classifier/Logistic Regression
8	Support Vector Classifier

Table 3.6: Best algorithm per subject - Standard Pipeline.

For Standard pipeline, for subjects 1, 3, 4, 6 and 7 both algorithms provided the same score, so choosing between both was an arbitrary decision.

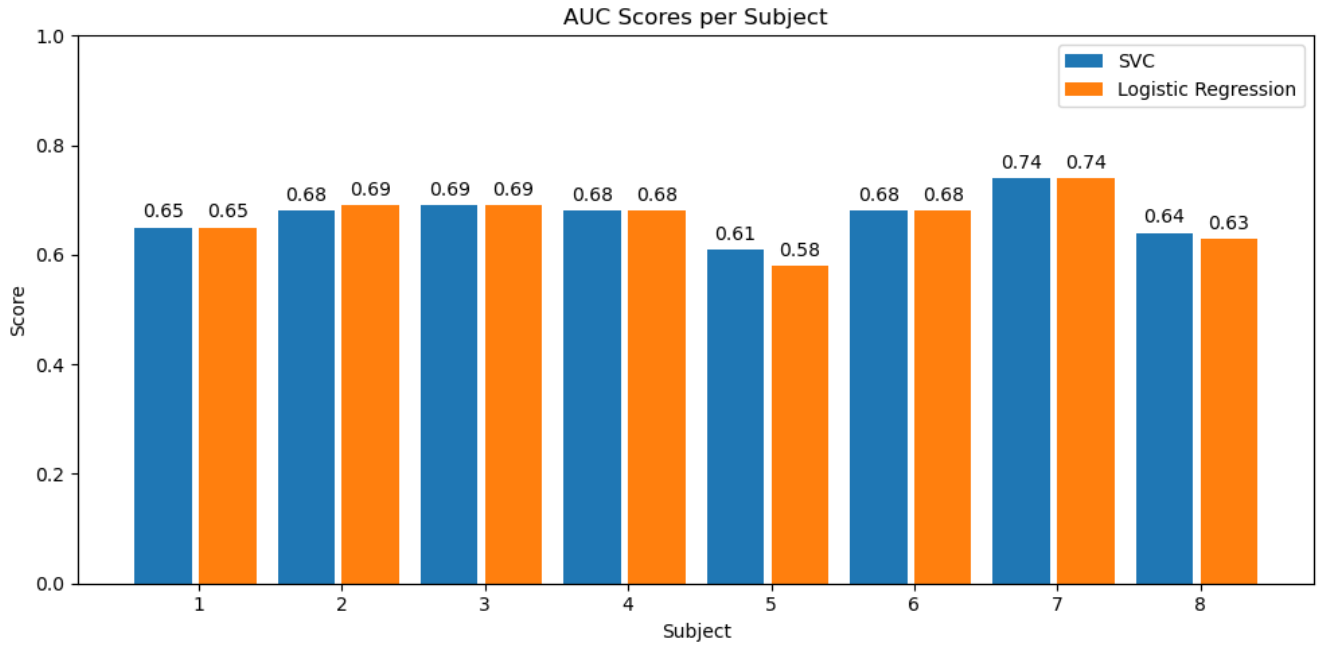


Figure 3.5: SVC vs Logistic Regression best score per subject - Standard Pipeline.

The results are the following:

Subject	Algorithm
1	Support Vector Classifier
2	Support Vector Classifier
3	Support Vector Classifier/Logistic Regression
4	Logistic Regression
5	Support Vector Classifier/Logistic Regression
6	Logistic Regression
7	Support Vector Classifier/Logistic Regression
8	Logistic Regression

Table 3.7: Best algorithm per subject - xDawn Pipeline.

For xDawn pipeline, for subjects 1, 3, 4, 7 and 8 both algorithms provided the same score, so choosing between both was an arbitrary decision. Logistic Regression was chosen.

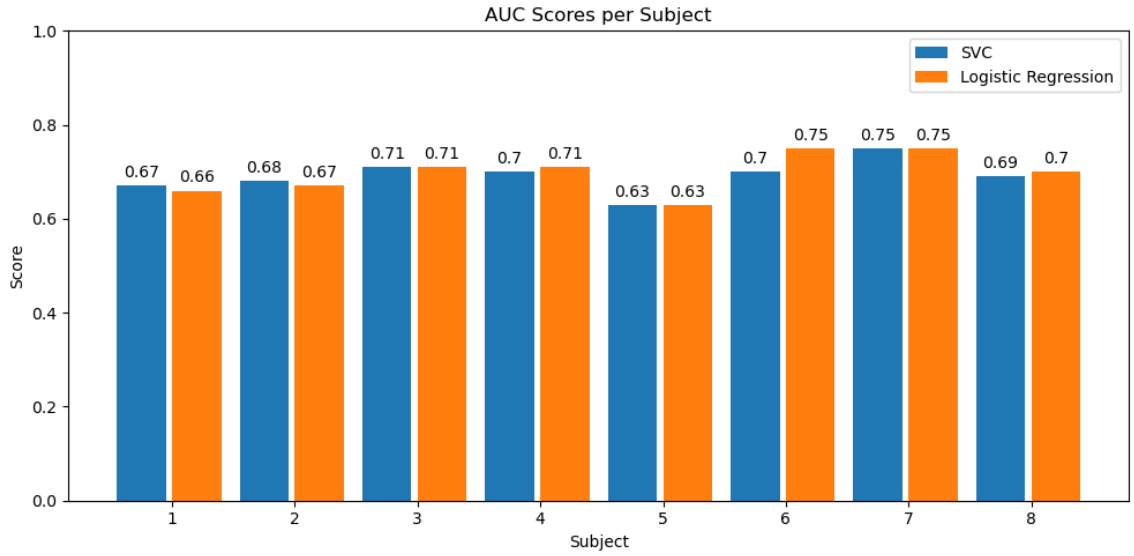


Figure 3.6: SVC vs Logistic Regression best score per subject - xDawn Pipeline.

3.3.3 Spatial Filtering

Average AUC values, with their standard deviation, were calculated using the two cross-validation algorithms mentioned in section 2.7. Configurations for each algorithm are the ones mentioned in tables 3.2, 3.3, 3.4, 3.5. For PCA, both best configurations (for Standard and xDawn) were tested and the one chosen was the best one of those.

	Standard	PCA	xDawn
1	0.64 ± 0.04	0.50 ± 0.08	0.65 ± 0.05
2	0.72 ± 0.12	0.49 ± 0.08	0.67 ± 0.07
3	0.72 ± 0.07	0.53 ± 0.07	0.71 ± 0.10
4	0.67 ± 0.16	0.45 ± 0.18	0.71 ± 0.15
5	0.58 ± 0.05	0.49 ± 0.05	0.63 ± 0.08
6	0.73 ± 0.09	0.46 ± 0.07	0.76 ± 0.08
7	0.75 ± 0.03	0.56 ± 0.05	0.75 ± 0.03
8	0.66 ± 0.17	0.51 ± 0.11	0.70 ± 0.13

Table 3.8: Classification average scores comparison between PCA, xDawn and Standard. Kfold with k=number_of_games (4, 5 or 6 representing training_set_size test_set_size)

	Standard	PCA	xDawn
1	0.64 ± 0.02	0.50 ± 0.01	0.62 ± 0.02
2	0.64 ± 0.06	0.48 ± 0.03	0.65 ± 0.02
3	0.68 ± 0.04	0.56 ± 0.03	0.65 ± 0.05
4	0.63 ± 0.03	0.53 ± 0.03	0.66 ± 0.02
5	0.53 ± 0.05	0.51 ± 0.02	0.62 ± 0.03
6	0.61 ± 0.07	0.49 ± 0.06	0.67 ± 0.05
7	0.70 ± 0.02	0.55 ± 0.03	0.71 ± 0.01
8	0.60 ± 0.04	0.49 ± 0.03	0.66 ± 0.05

Table 3.9: Classification average scores comparison between PCA, xDawn and Standard. ShuffleSplit with k =number_of_games(4, 5 or 6 representing training_set_size test_set_size) and test_size=0.5% or 0.6% (as mentioned in table 3.1)

3.3.3.1 xDawn Analysis

In general, the use of xDawn spatial filters, yielded higher AUCs than the standard method based on 8 predefined channels, for all subjects. In cases where xDawn does not generate an improvement, the results are the same or slightly worse, e.g subject 1 in table 3.9. However, in cases where it does, the improvement is significant, e.g subject 8 in table 3.9.

The changes observed in the standard deviation between Standard and xDawn do not allow to draw solid conclusions

The values with K-fold (table 3.8) are better compared to those of ShuffleSplit (table 3.9) as expected, since the training is carried out with a greater number of games.

Carrying out the comparison between Standard and xDawn, but taking into account the K-fold and ShuffleSplit distributions, it is seen that the improvement in results provided by xDawn is more significant when the number of training sets is less, that is, in ShuffleSplit

3.3.3.2 PCA Analysis

By analyzing the results of the tables we were able to detect that the PCA results are not as expected. Having investigated and taking into account previous analysis, spatial filters for error detection should improve the classification. In fact, as previously discussed, with xdawn the values show an improvement. However, when applying the PCA filter, the same does not happen. To verify that it was not an implementation error and that the results are really reliable, three different tests were carried out. It should be noted that these tests were applied in the rest of the pipeline configurations also as an extra verification, but with PCA are necessary.

The first test consists of shuffling the labels, that is, the hits and no hits. This allows labels to be meaningless in relation to signals. As can be seen from the results shown in figures 3.7 and 3.8, the AUC values are close to 0.5 for all subjects, and therefore the test is successfully completed.

The second test consists of hardcoding very high values for a certain point in the signal, for all times that are hits in such a way that the classification is very close to perfection. As can be seen from the results shown in figures 3.9 and 3.10, the AUC values are close to 1, and therefore the test passes.

The third test consists of training with all the data, and testing with one of them. The AUC values in the ROC curves are expected to be high. As can be seen from the results shown in figures 3.11 and 3.12, the AUC values are high, and therefore the test passes.

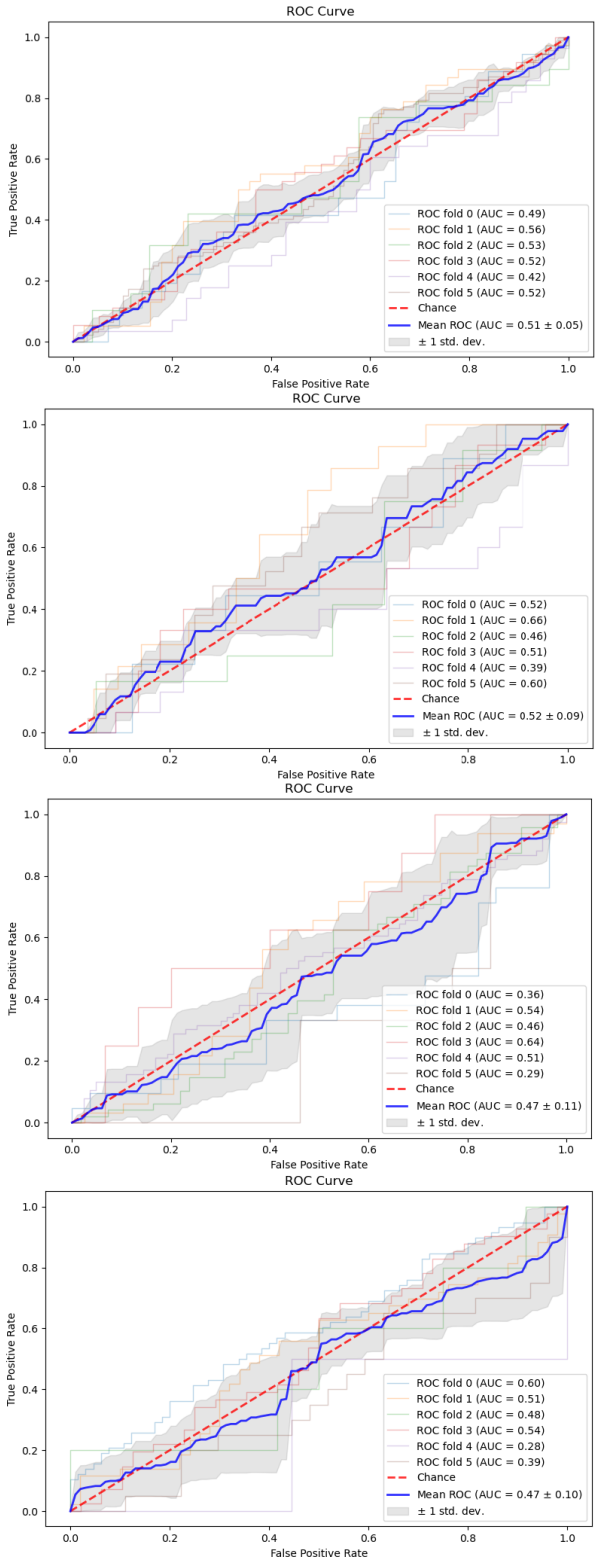


Figure 3.7: PCA ROC Curves, subjects 1 to 4. Training with randoms labels.

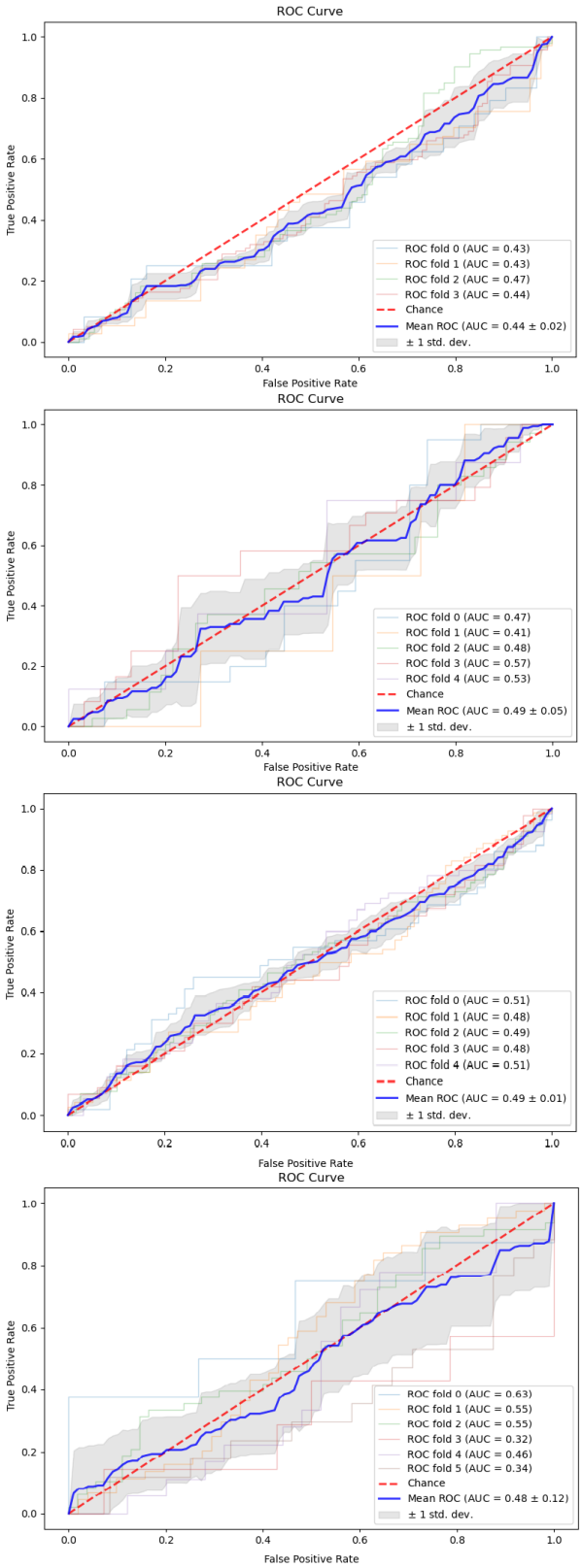


Figure 3.8: PCA ROC Curves, subjects 4 to 8. Training with randoms labels.

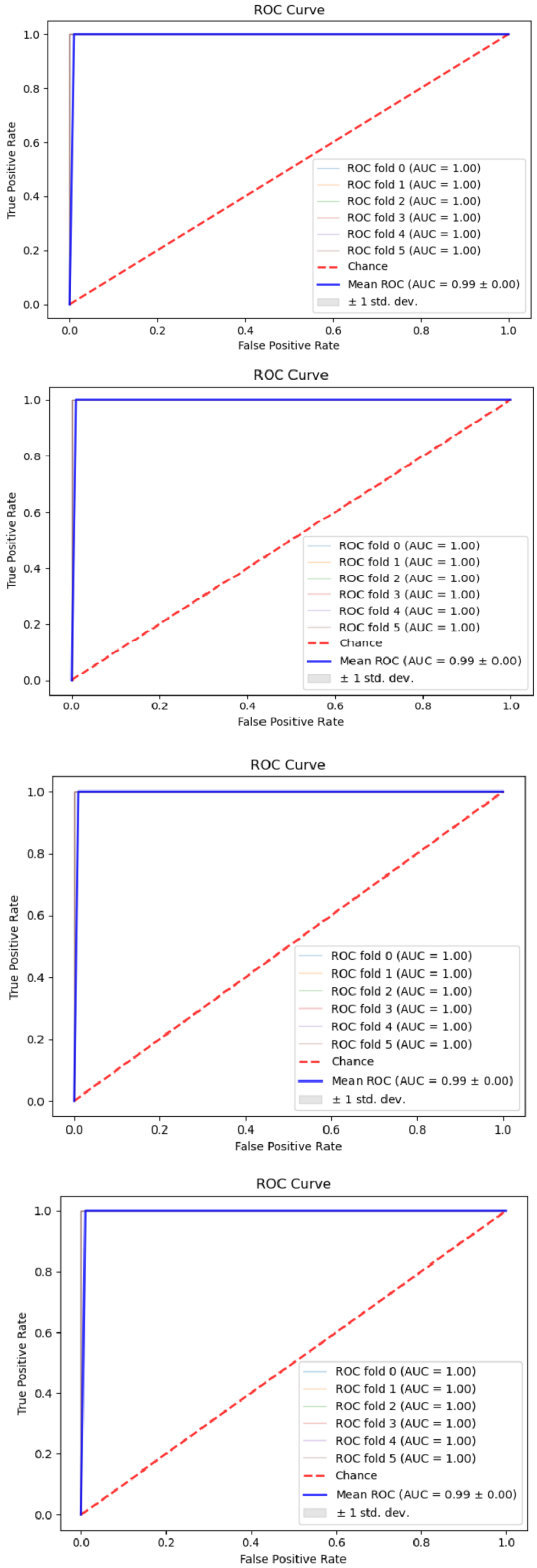


Figure 3.9: PCA ROC Curves, subjects 1 to 4. Training with fixed high value.

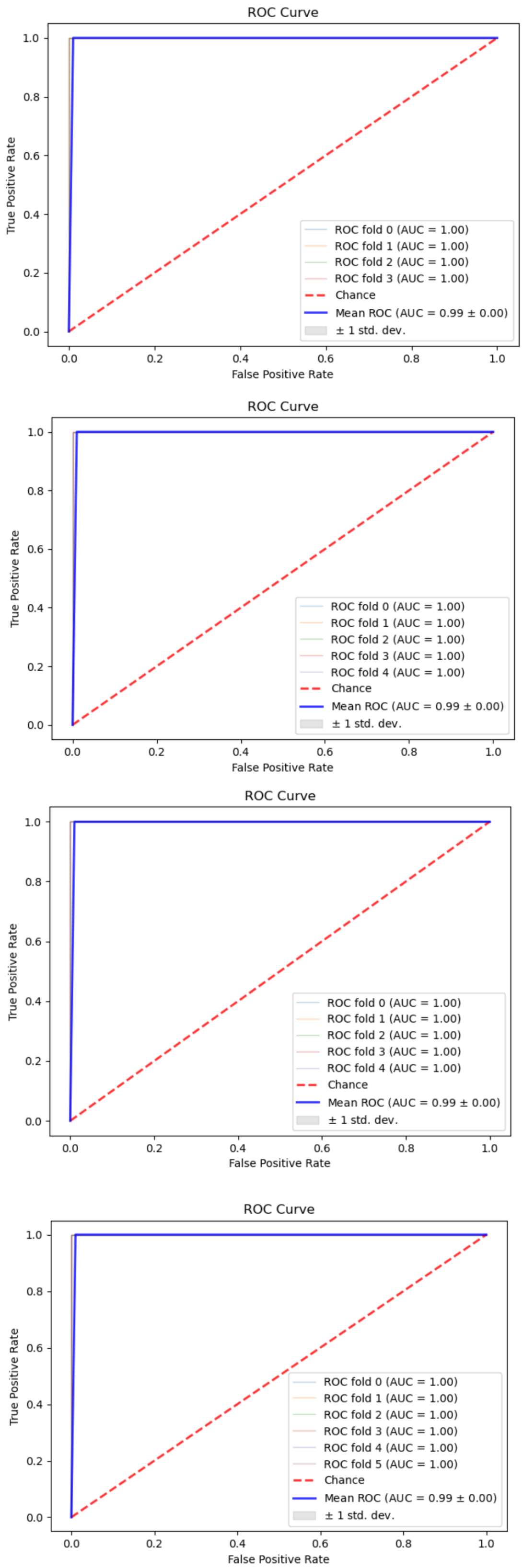


Figure 3.10: PCA ROC Curves, subjects 4 to 8. Training with fixed high value.

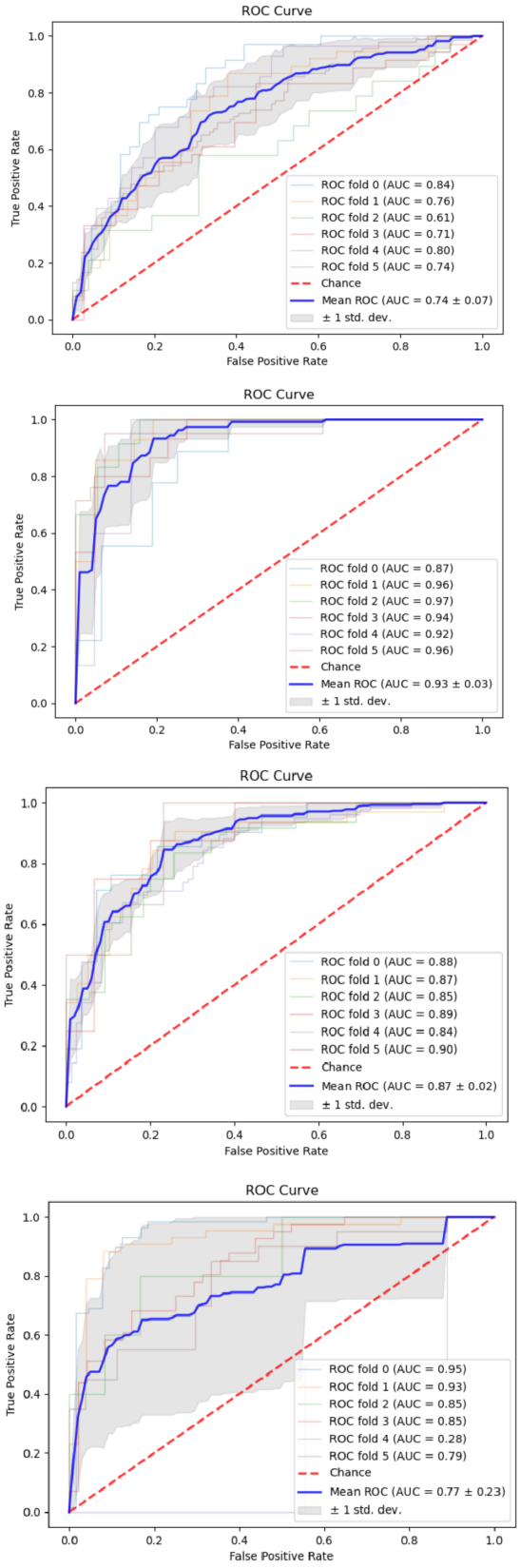


Figure 3.11: PCA ROC Curves, subjects 1 to 4. Training with all experiences, testing with one of those.

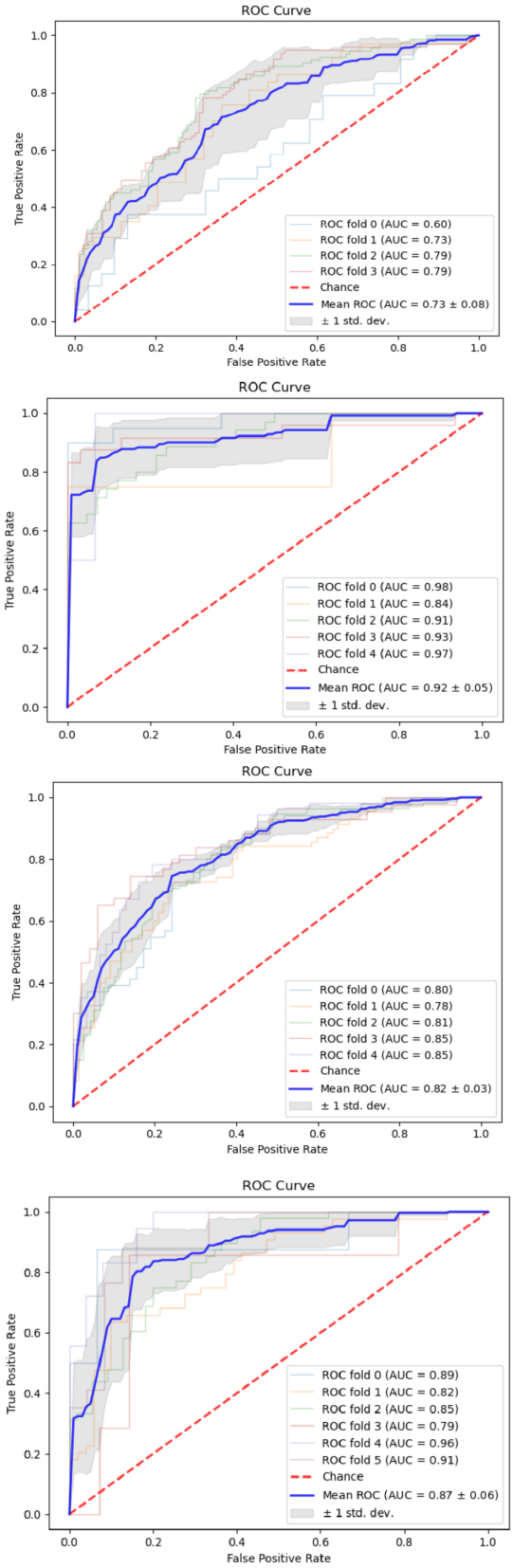


Figure 3.12: PCA ROC Curves, subjects 4 to 8. Training with all experiences, testing with one of those.

There are different hypotesis to explain why PCA produces bad results. PCA is performs poorly when the data has high bias, but that's not the case, since all the test were done using cross validation. Neither can be the fact that the data is imbalanced since data is splitted evenly and contains hits and no-hits with a good ratio. When independent and dependent variables are not linearly related, PCA can also fail, but for this cases there's an extension of the algorithm called Kernel PCA, that, is good to mention, was tried too and failed. Another failure situation could be if within class variance is higher than between class variance, but in that case, then xDawn should produced poor results too. So, after analyzing the failure possibilities and ruling out most of them because they are not applicable to this problem, one remains. Taking into account the main difference between xDawn and PCA, the explanation that pops out is that the noise N is not being taken into account to design filters like PCA, and that leads to a poor estimation of enhancing filters. Moreover, this latter fact also leads to not ensuring that the first principal components have the best output signal-to-noise ratios (SNRs). Taking all this into account, the PCA spatial filtering pipeline is discarded for upcoming analysis.

3.3.4 Classification Results

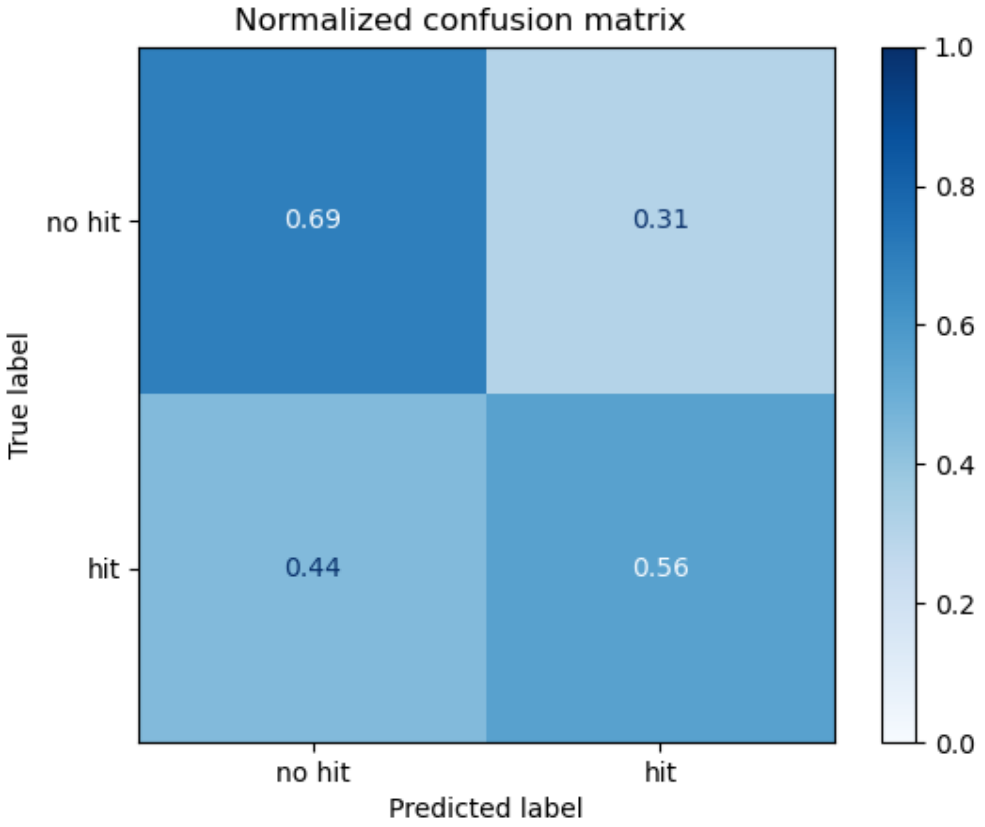
Figure 3.13 to Figure 3.20 show the Confusion Matrix and ROC curves for each subject using the best algorithm and xDawn.

In the confusion matrices it can be observed that:

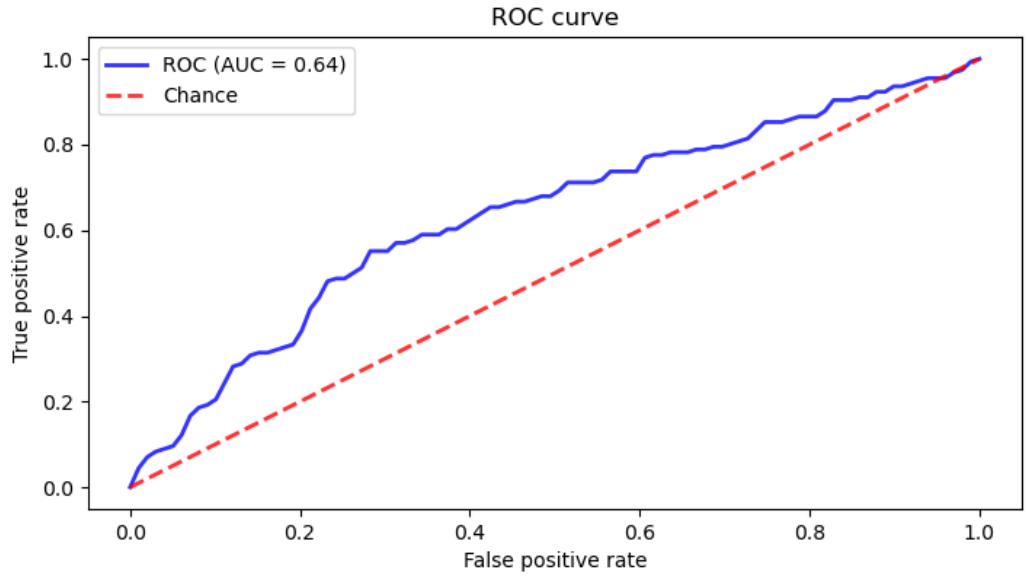
- The correct classifications, that is, the upper left and the lower right, have percentages above 0.5
- The percentage of Type I error (false positive) samples is consistently low for all subjects.
- The percentage of Type 2 error (false negative) is in all cases lower than 0.5

Algorithms classify no-hits better than hits.

Looking at the ROC curves, it can be seen how the subjects learn, some to a greater extent than others, but in all there is a considerable distance between the curve and the identity. Making the comparison with the previous study with the same data, in most cases, the results are better.

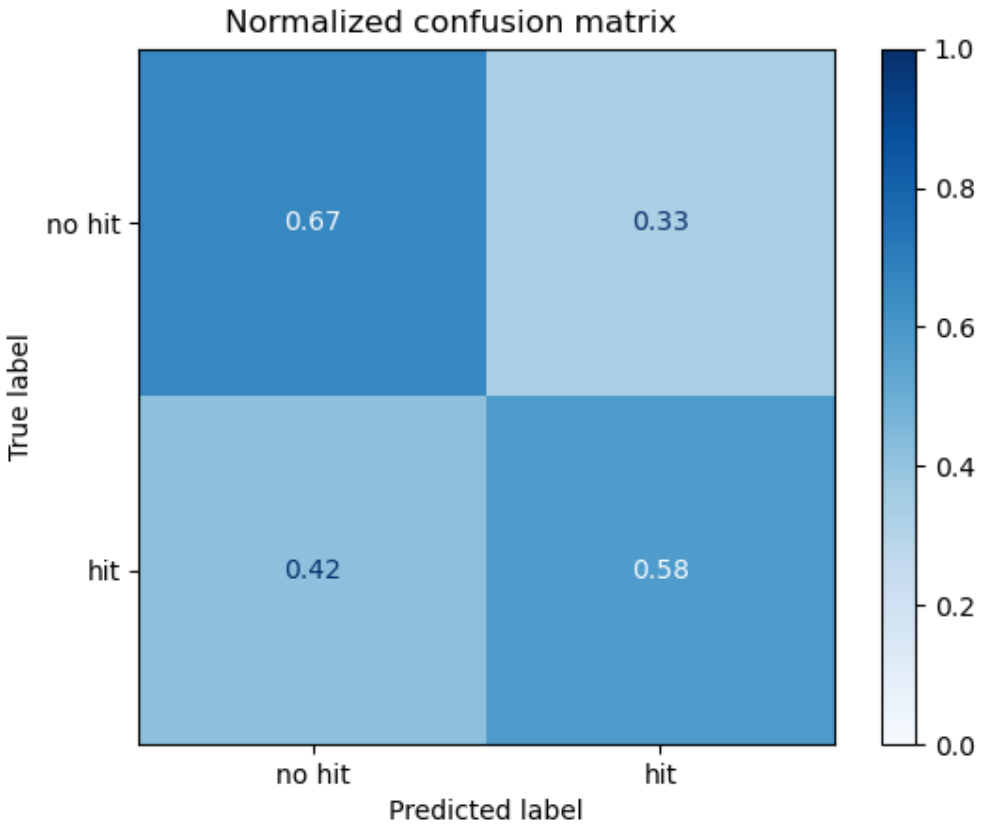


(a) Confusion Matrix: Subject 1

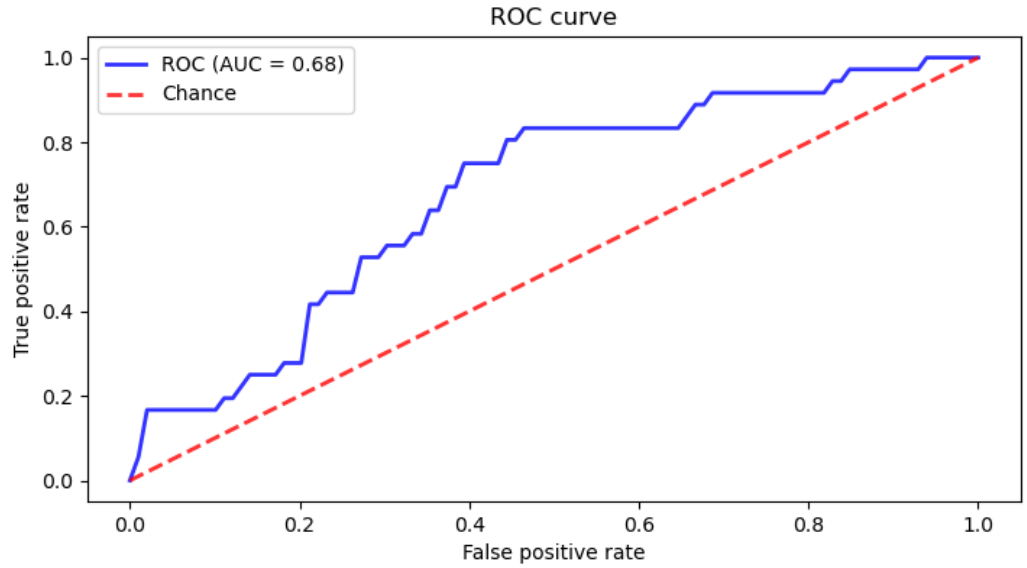


(b) Roc Curve: Subject 1

Figure 3.13: Classification Results: Subject 1

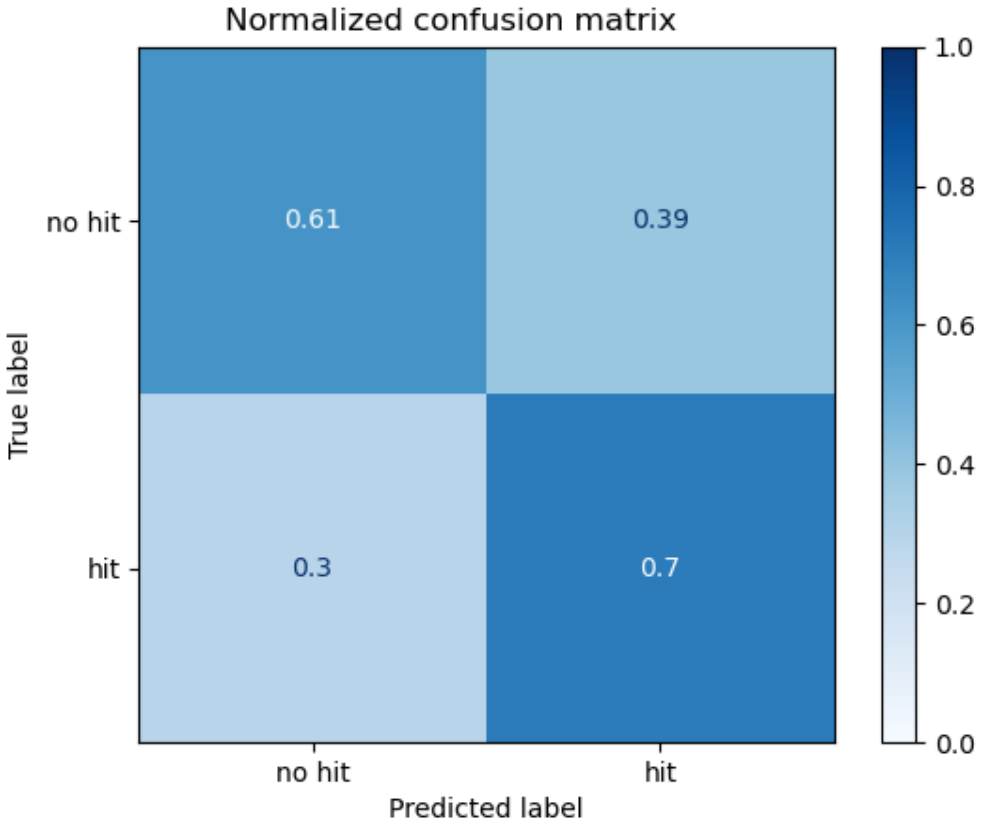


(a) Confusion Matrix: Subject 2

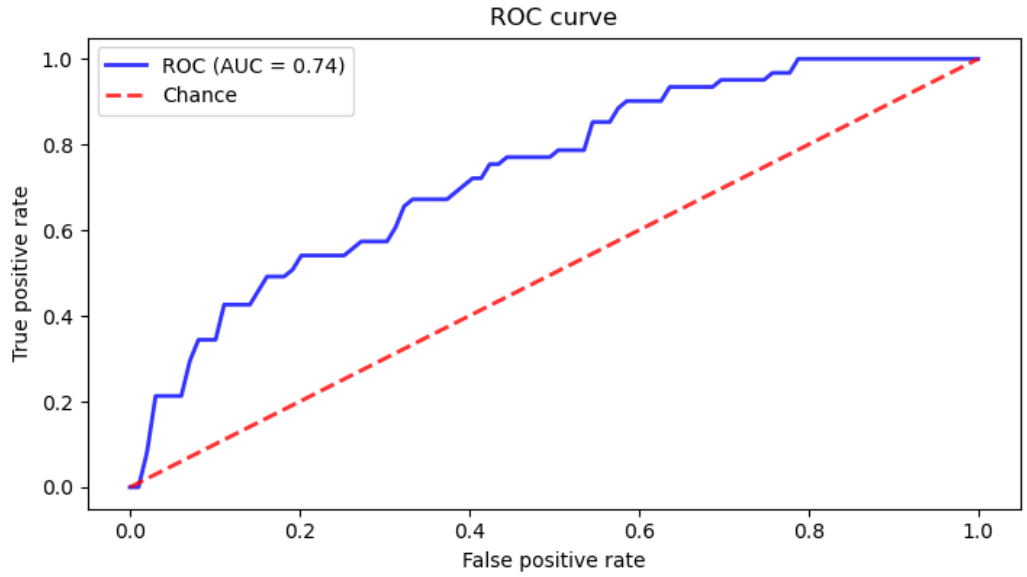


(b) Roc Curve: Subject 2

Figure 3.14: Classification Results: Subject 2

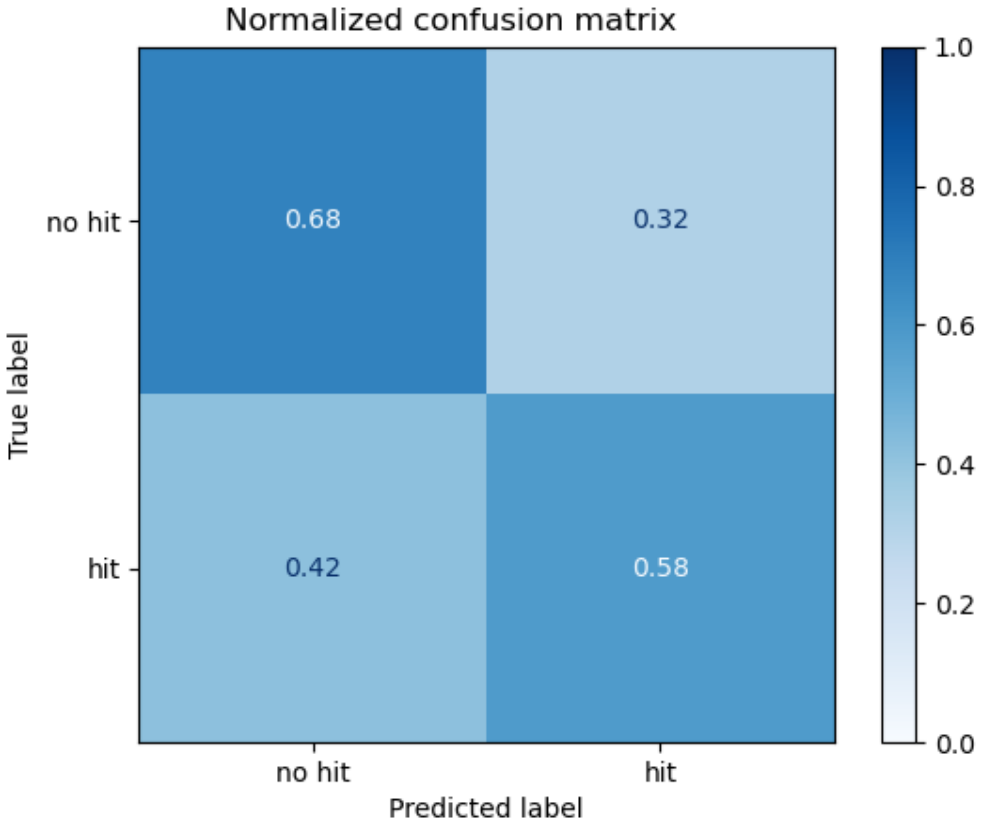


(a) Confusion Matrix: Subject 3

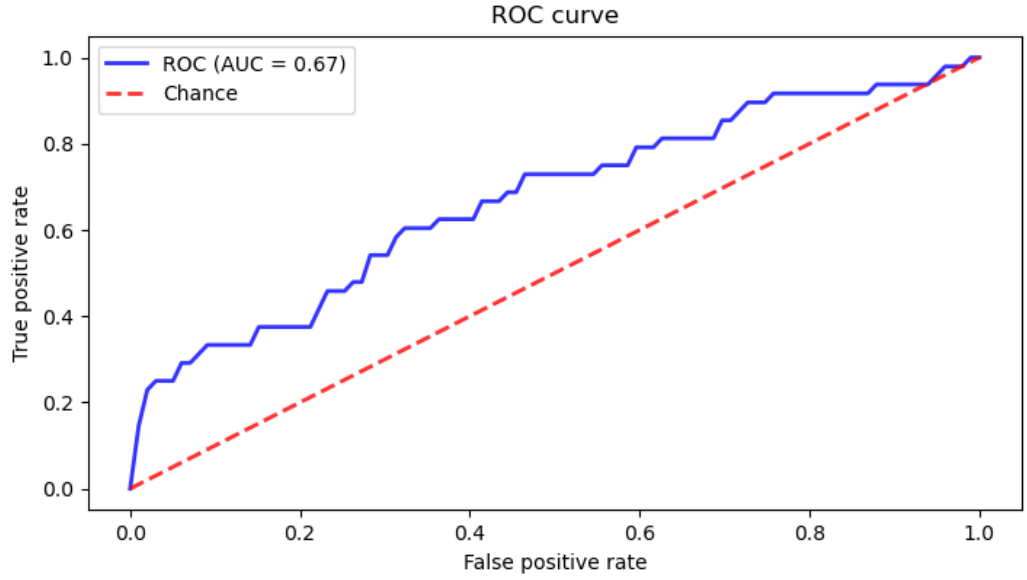


(b) Roc Curve: Subject 3

Figure 3.15: Classification Results: Subject 3

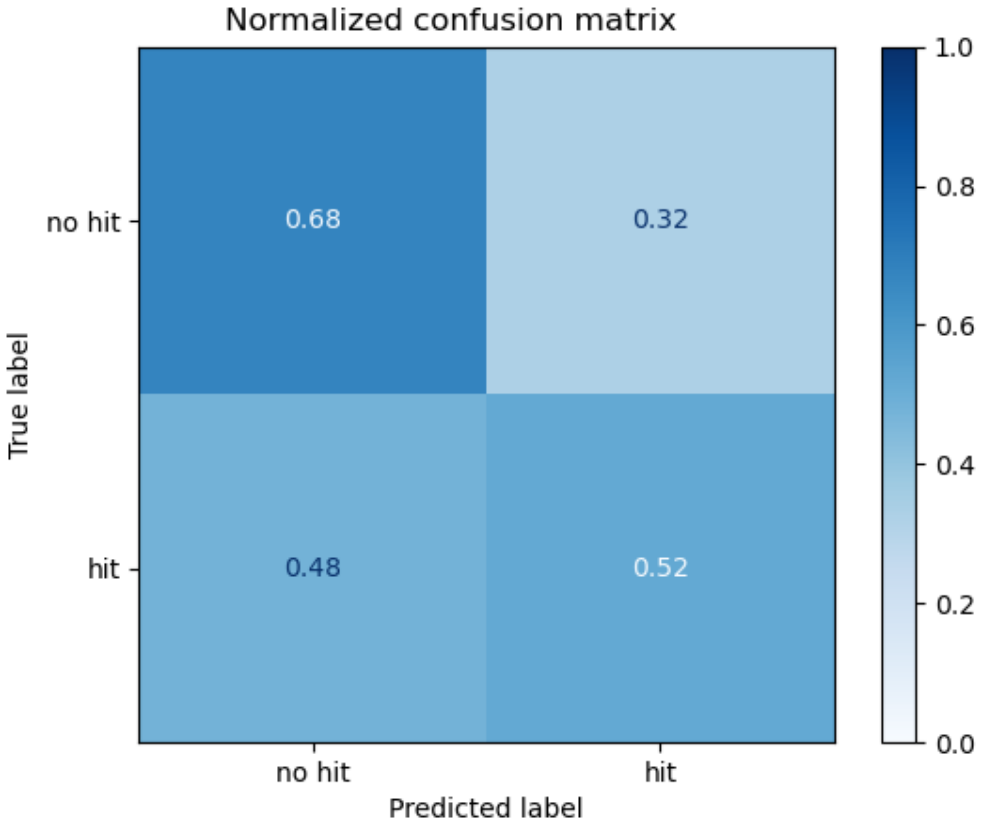


(a) Confusion Matrix: Subject 4

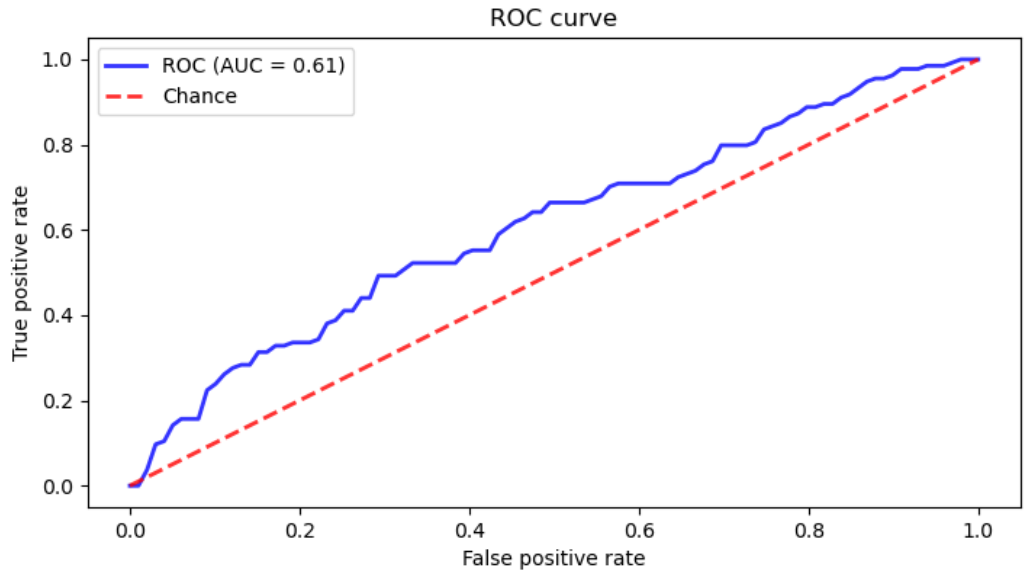


(b) Roc Curve: Subject 4

Figure 3.16: Classification Results: Subject 4

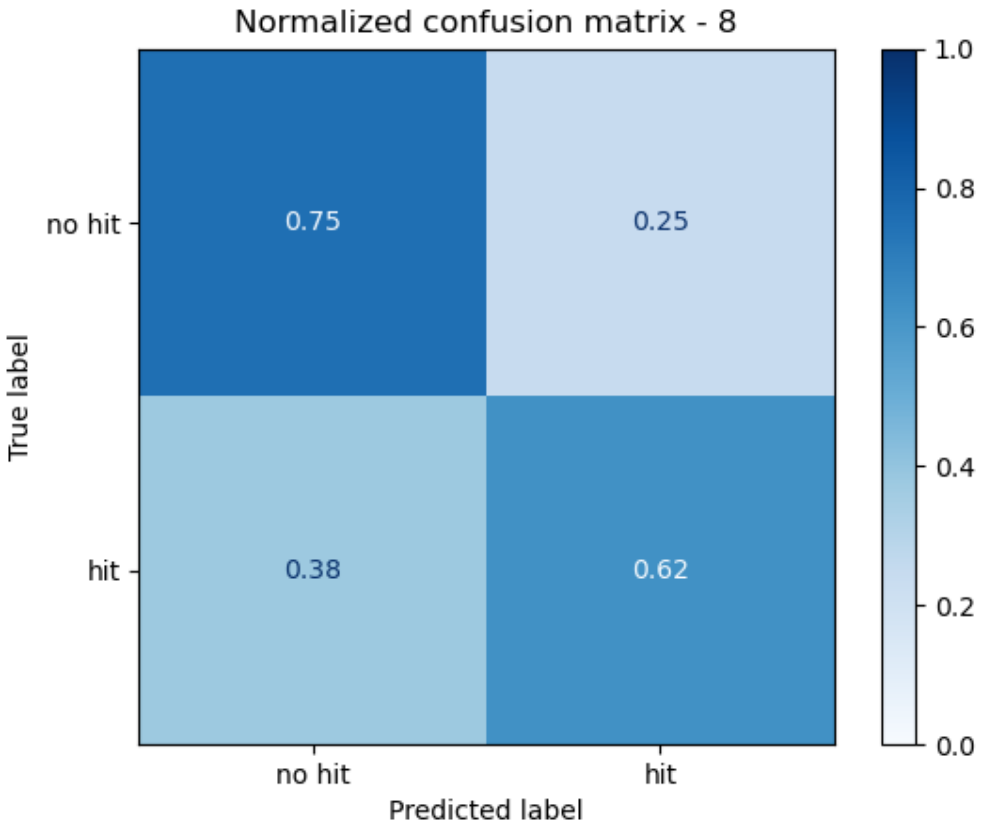


(a) Confusion Matrix: Subject 5

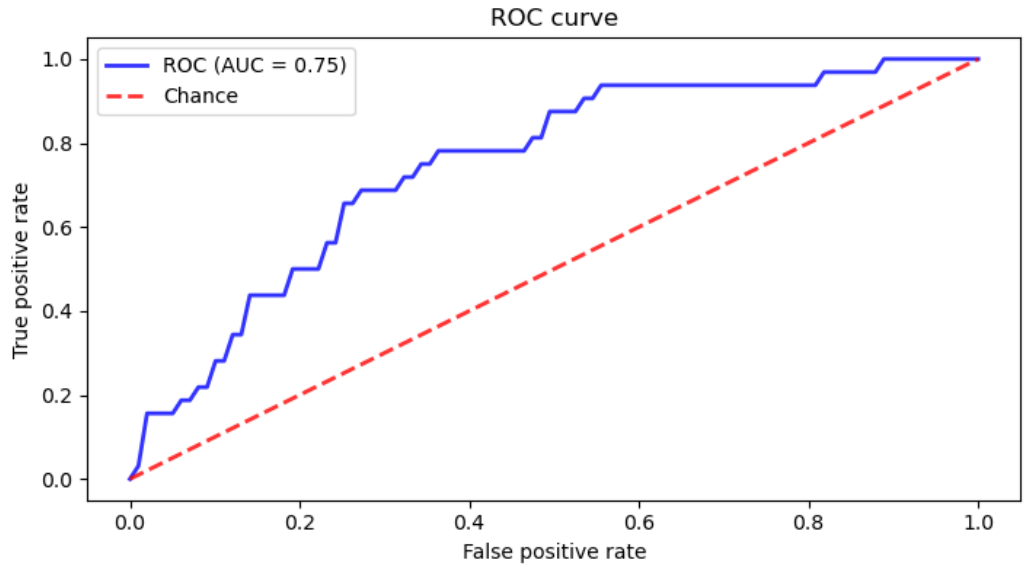


(b) Roc Curve: Subject 5

Figure 3.17: Classification Results: Subject 5

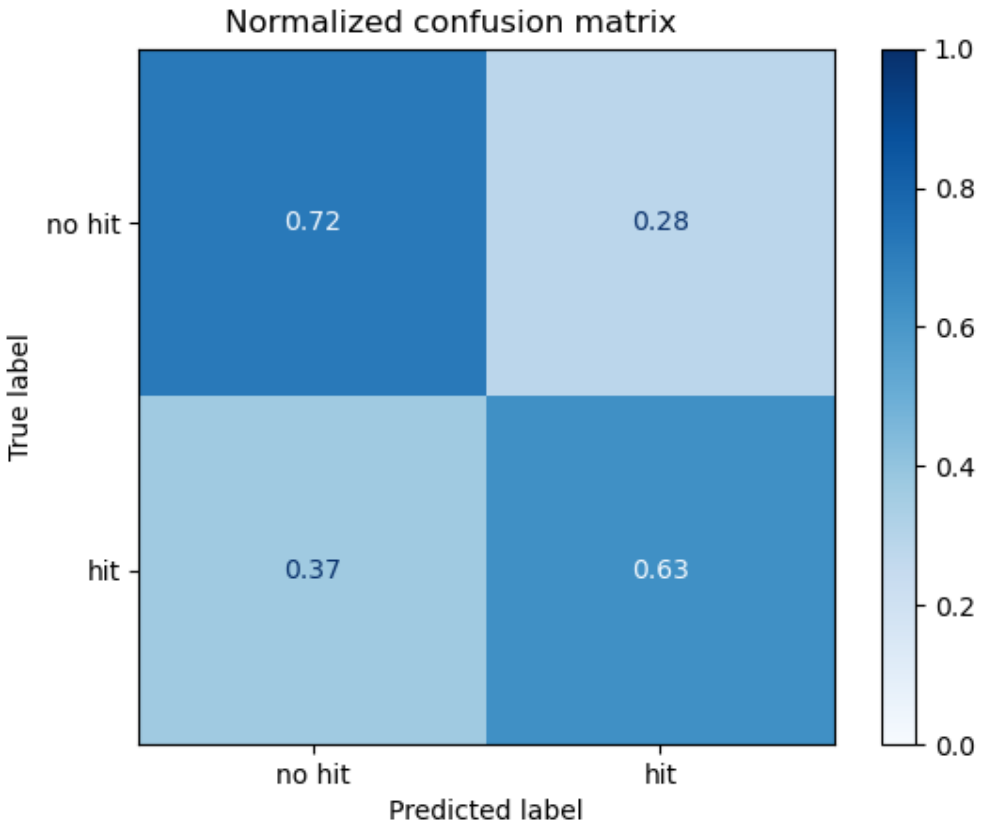


(a) Confusion Matrix: Subject 6

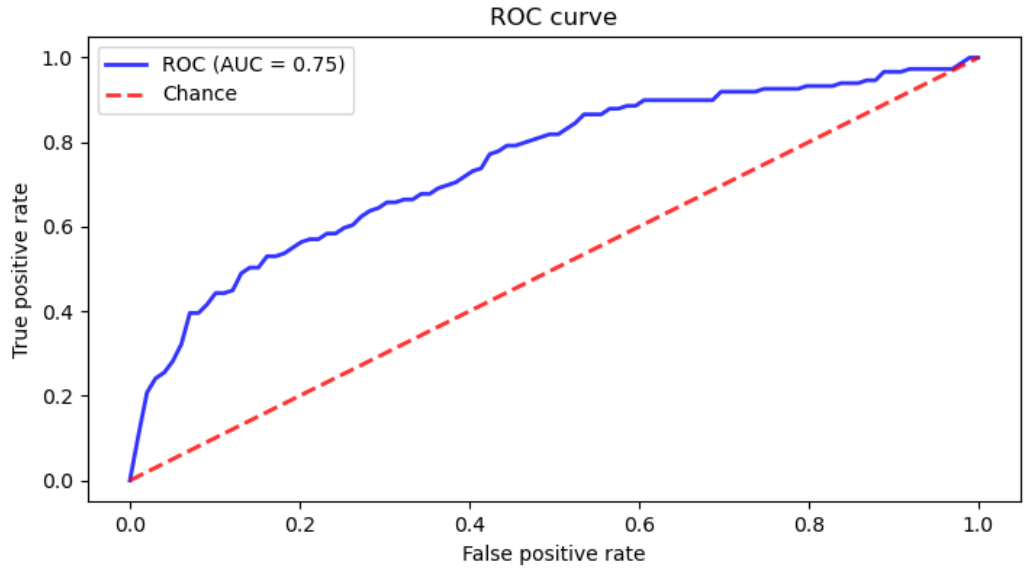


(b) Roc Curve: Subject 6

Figure 3.18: Classification Results: Subject 6

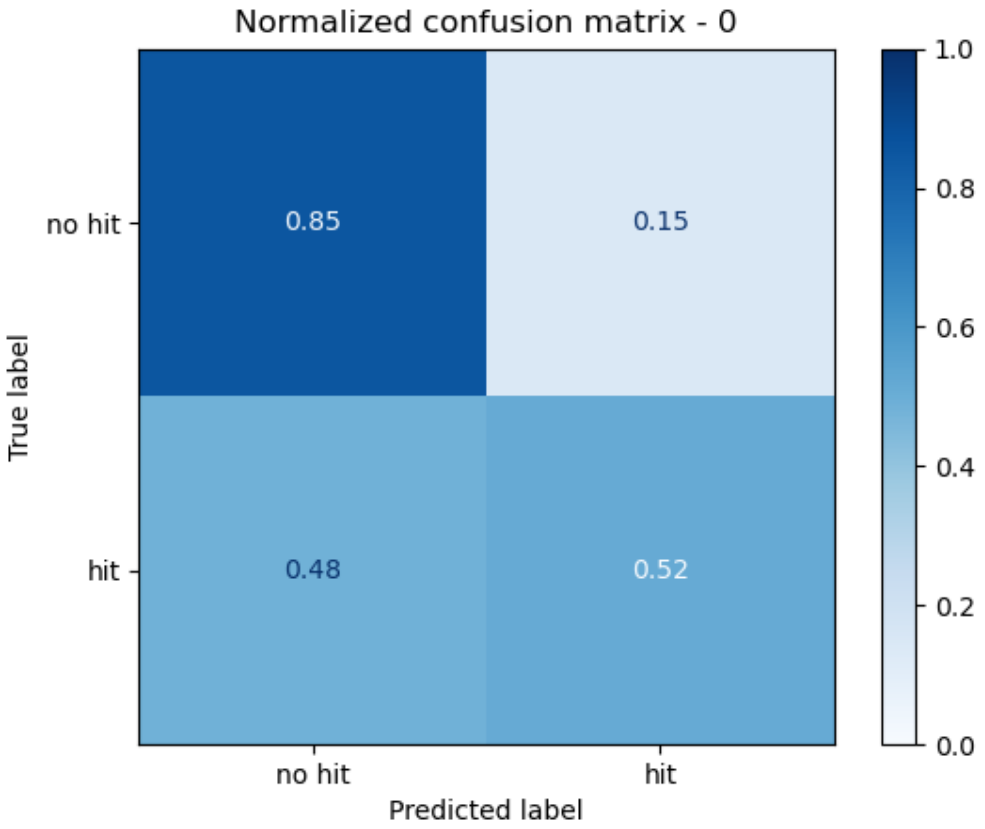


(a) Confusion Matrix: Subject 7

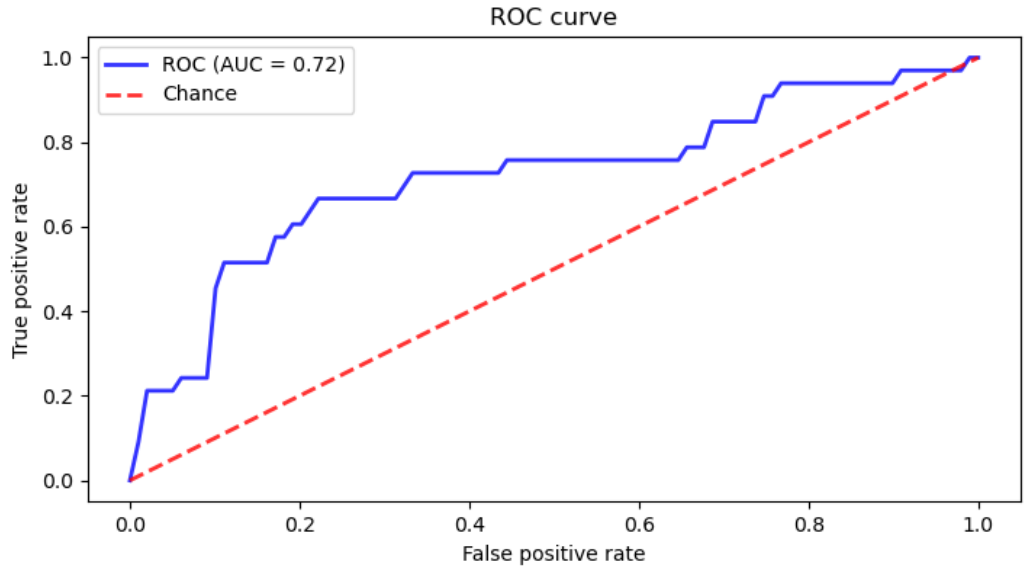


(b) Roc Curve: Subject 7

Figure 3.19: Classification Results: Subject 7



(a) Confusion Matrix: Subject 8



(b) Roc Curve: Subject 8

Figure 3.20: Classification Results: Subject 8

3.4 Reinforcement Learning

3.4.1 Average steps to goal

To assess the agent's level of learning when using the Q-table, the number of steps it takes for the agent to reach the goal is used as a metric. This value is calculated progressively while training the Q-Table using the calculated reward for each subject. In figures 3.21 to 3.28 the progress on the amount of steps can be detected. Each of the points on the x-axis represents a game within the test set. Each point represents on the y-axis the average number of steps the agent takes for a Q-Table in 200 iterations. The number of points represents the number of games that are part of the test set. In the first point, the Q-Table starts from the initial state, that is, empty and therefore the decisions remain randoms. Each advance on the x-axis represents an experience change, and therefore a new Q-table is used, updated with the values obtained with previous runs.

As can be seen, as the Q-Table is trained, the number of steps it takes to reach the goal decreases, that is, the agent learns. However, not all subjects learn in the same way; For example, subject 3's learning is faster than subject 4. In a previous study using the same data, subjects 5 and 6 had not shown learning. The idea that the data was not enough had been predicted, however, in this case, with the application of the spatial filter xDawn, the results were satisfactory.

Figure 3.29 shows how the number of steps evolves when training a Q-Table with a experience of each subject progressively. Analyzing the results, the conclusion reached is that even using information from different subjects, the agent learns and improves his performance. In this way it is proved that the reinforcement learning algorithm is independent of the subjects. This is because the only information that matters is the calculated rewards, so if the classification is done correctly, data from different subjects can be used.

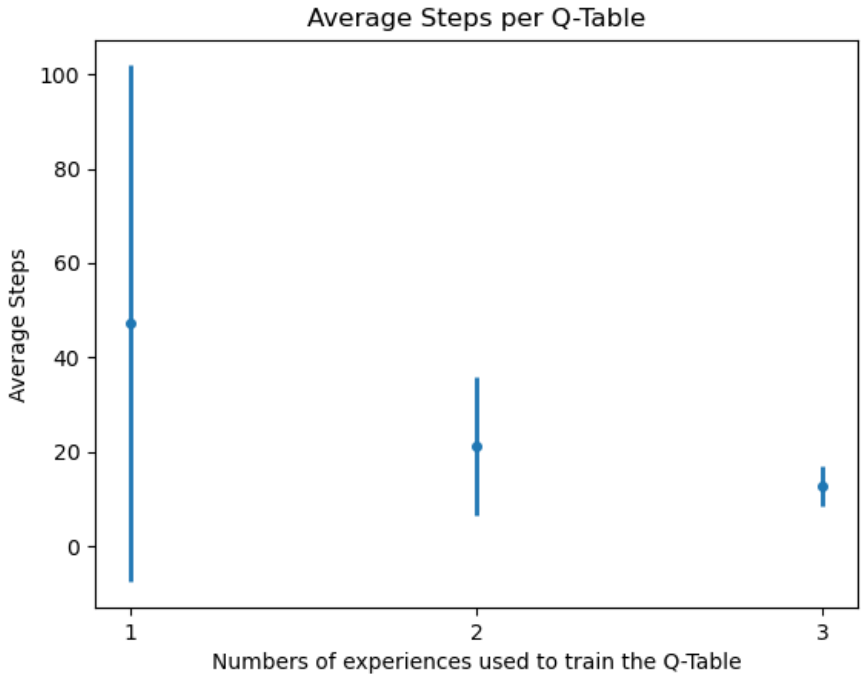


Figure 3.21: Average steps using Q-table: Subject 1

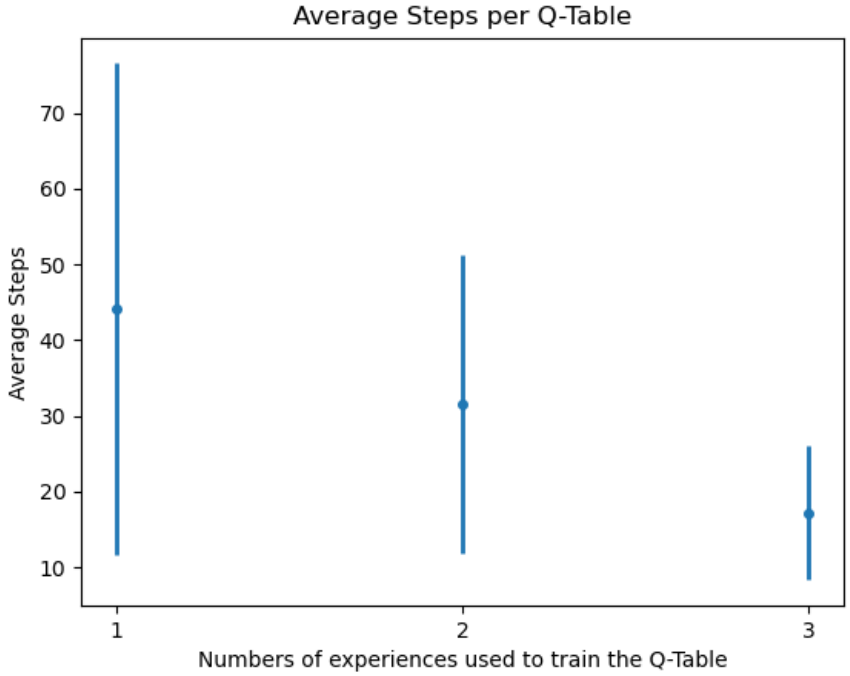


Figure 3.22: Average steps using Q-table: Subject 2

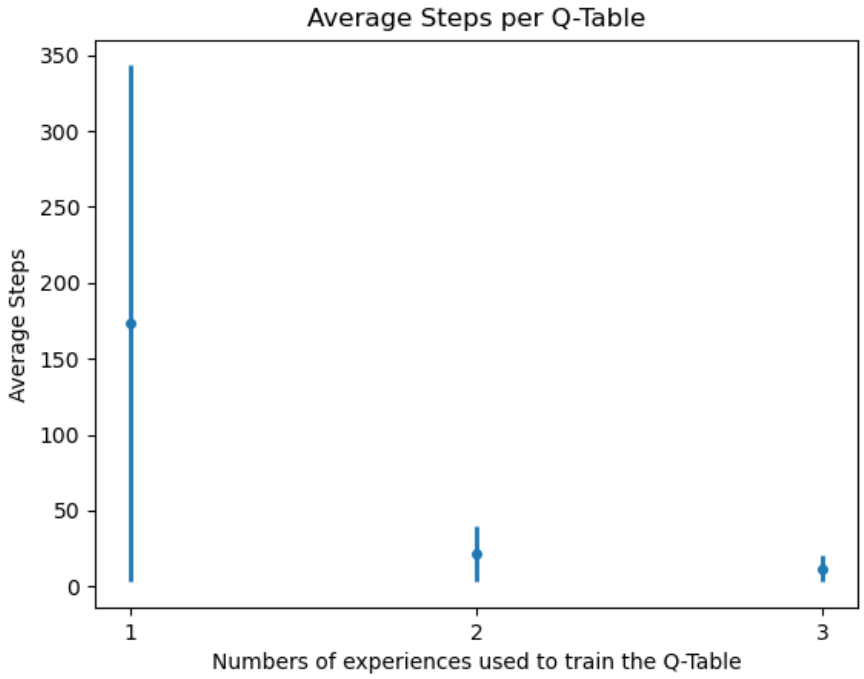


Figure 3.23: Average steps using Q-table: Subject 3

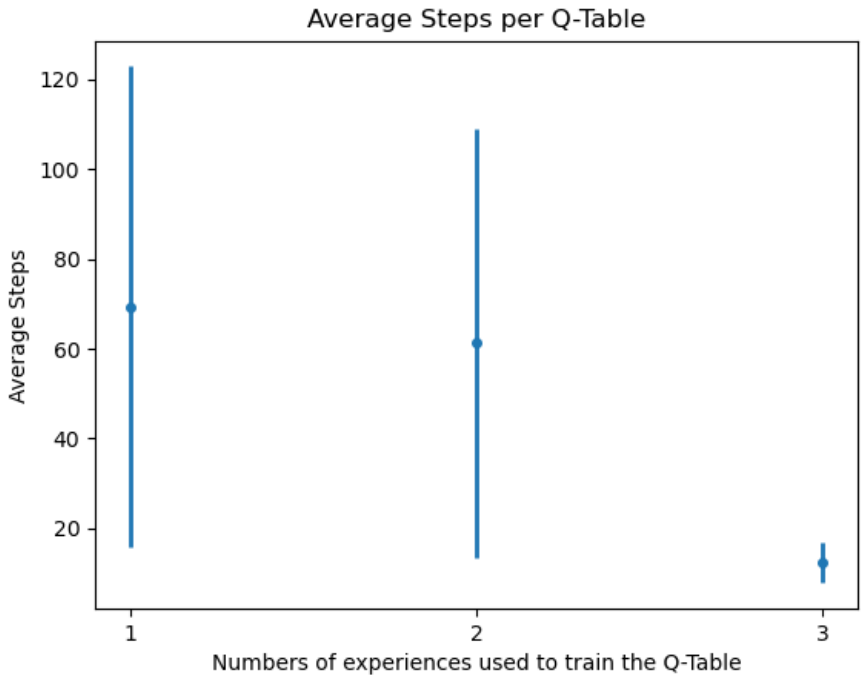


Figure 3.24: Average steps using Q-table: Subject 4

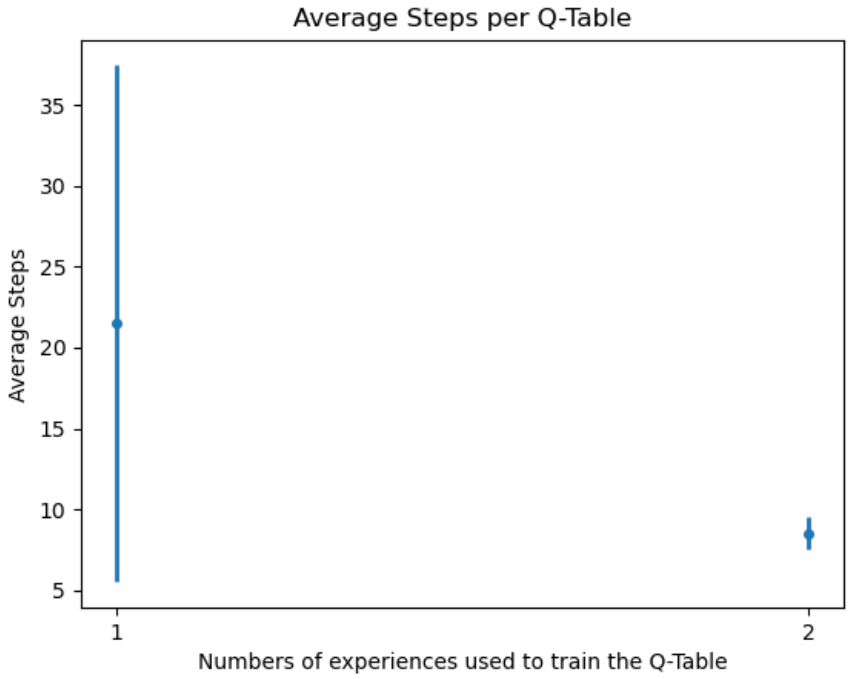


Figure 3.25: Average steps using Q-table: Subject 5

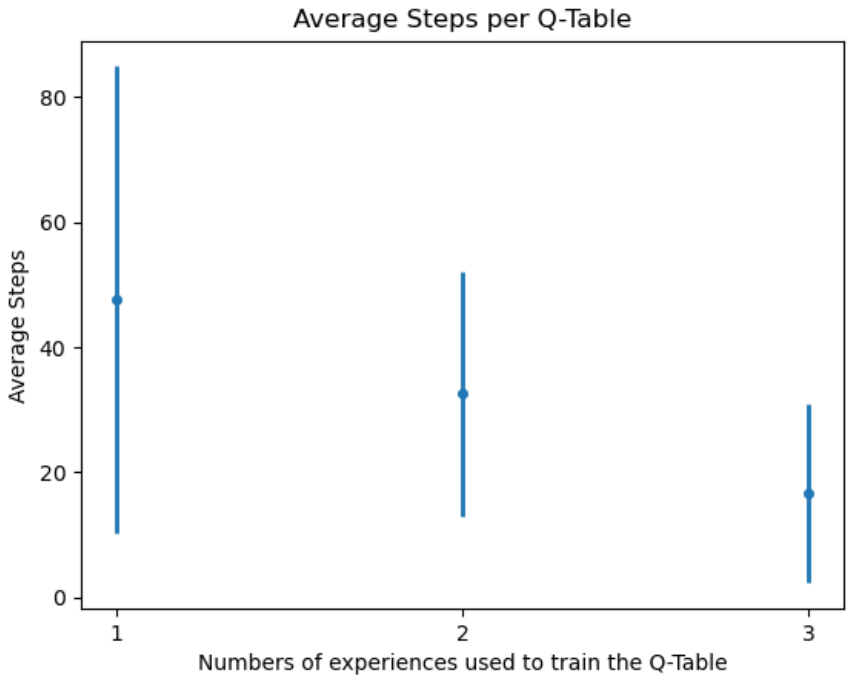


Figure 3.26: Average steps using Q-table: Subject 6

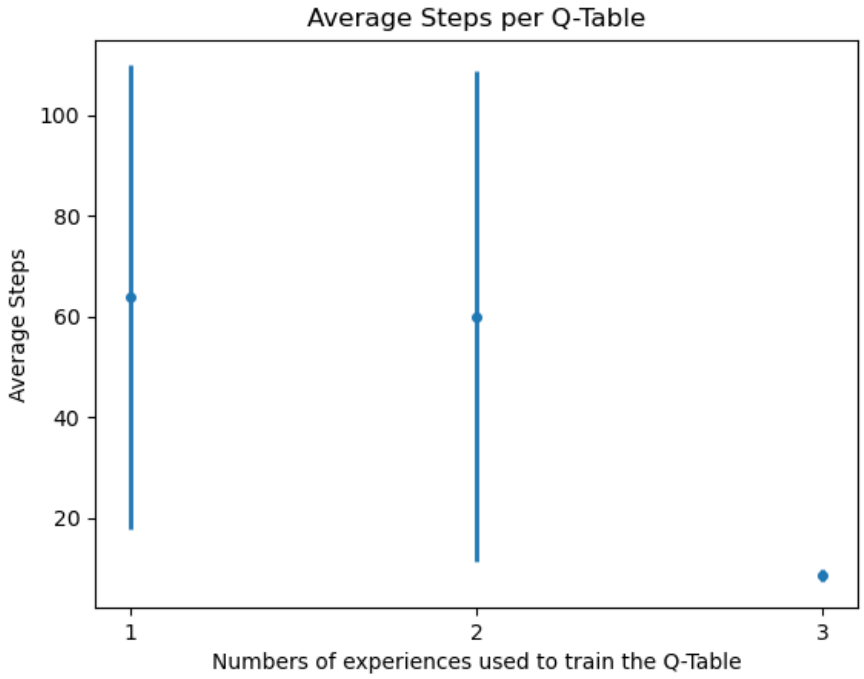


Figure 3.27: Average steps using Q-table: Subject 7

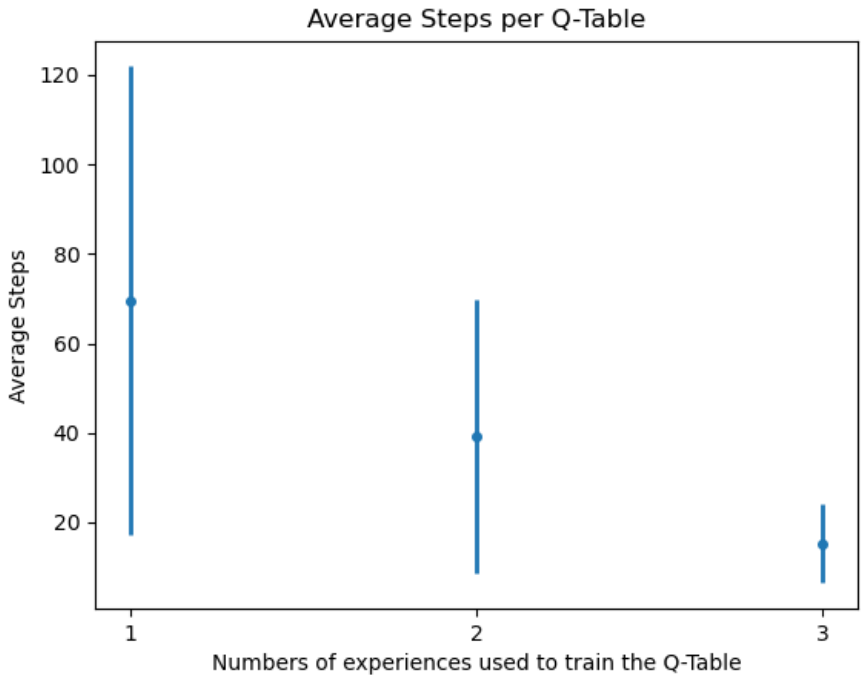


Figure 3.28: Average steps using Q-table: Subject 8

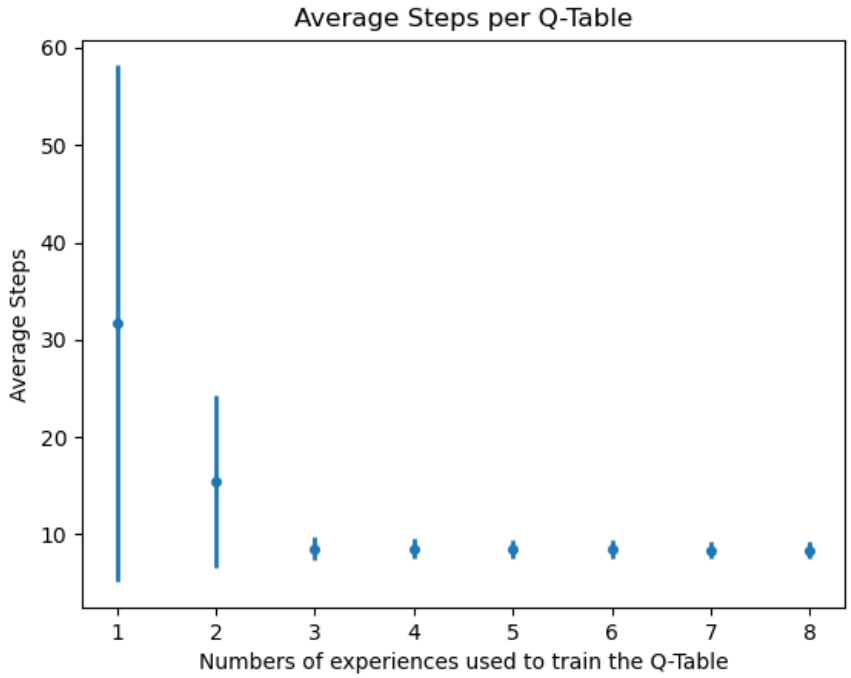


Figure 3.29: Average steps using Q-table: All Subjects, one game per subject.

3.4.2 Heat map of Learned Policies

An interesting complement to the analysis, which shows how the learning algorithm works, is the study of the positions that the agent takes in the game. To do this, the heat maps presented below were made. The color gamut extends from yellow to red, from lowest to highest frequency respectively. For each iteration, the numbers represent the average of 50 samples. Subject 7 was taken as the test subject, because the results obtained and shown in the previous section were the best compared to the rest of the subjects. Four instances are distinguished, where each one uses a different Q-Table that is obtained progressively.

The figure 3.30 represents the initial heat map, with the Q-Table empty, that is, the movements are completely randoms. An oscillation around the position (1; 1) can be observed. The frequency decreases in positions furthest from the epicenter.

The figure 3.31 represents the frequency after training the Q-Table for the first time. This map shows a more homogeneous frequency distribution, and lower values than the previous map. However, there is a peak on position (2; 2), which could be caused probably by a tendency to explore certain states that were not explored during the previous step. It can be detected how some of the positions are already discarded with 0 frequency values

The figure 3.32 shows a new iteration of the Q-table, where there is a greater amount of discarded positions, in the corners that do not correspond to the starting point or the arrival point, and their surroundings. In this way you can glimpse an almost armed road. The first 3 movements taken by the agent are always the same, which indicates that he has already found the optimal way to start. This is explained by the fact that errors in classification lead to penalties for some roads more than others. A perfect classification would show greater equality in the probabilities of choosing two equivalent actions that lead the agent to his goal. The movements towards the end are a little more varied.

The figure 3.33 shows the heat map that represents the occupied positions based on the final Q-table. It can be seen that in almost all cases with few exceptions, the path is the same, which, despite being skewed, is direct and optimal.

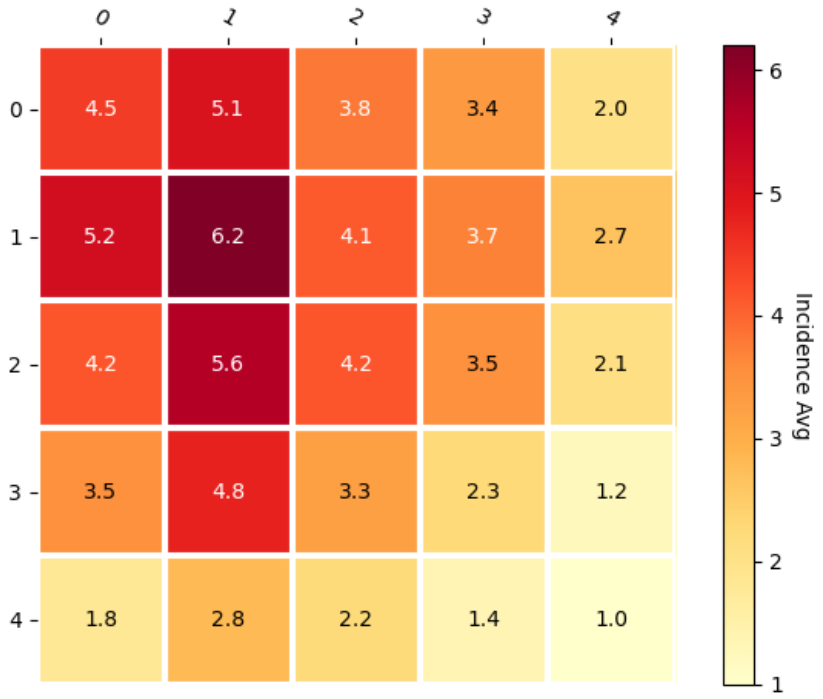


Figure 3.30: Heat map - Empty Q-Table. Subject 7

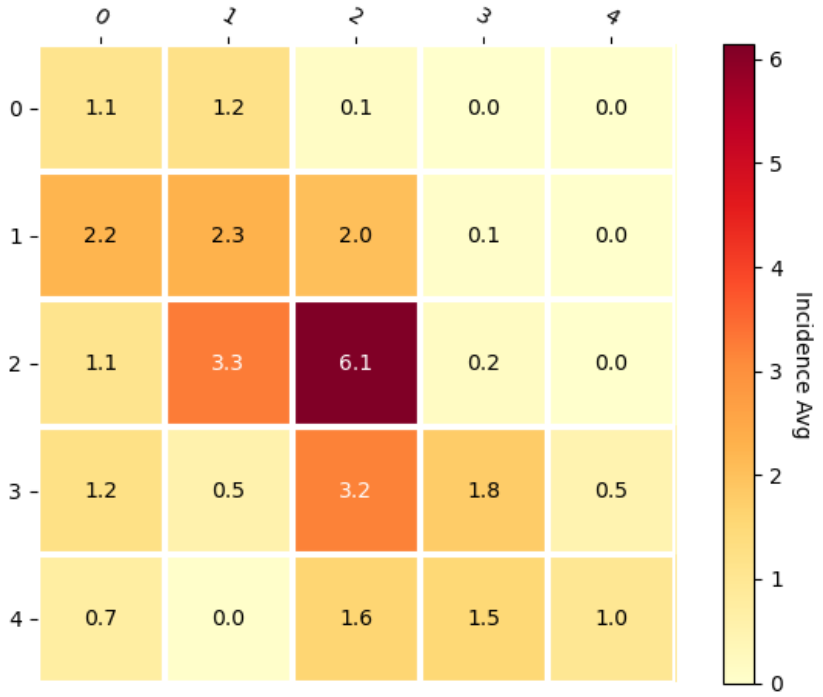


Figure 3.31: Heat map - Q-Table after 1 iteration. Subject 7

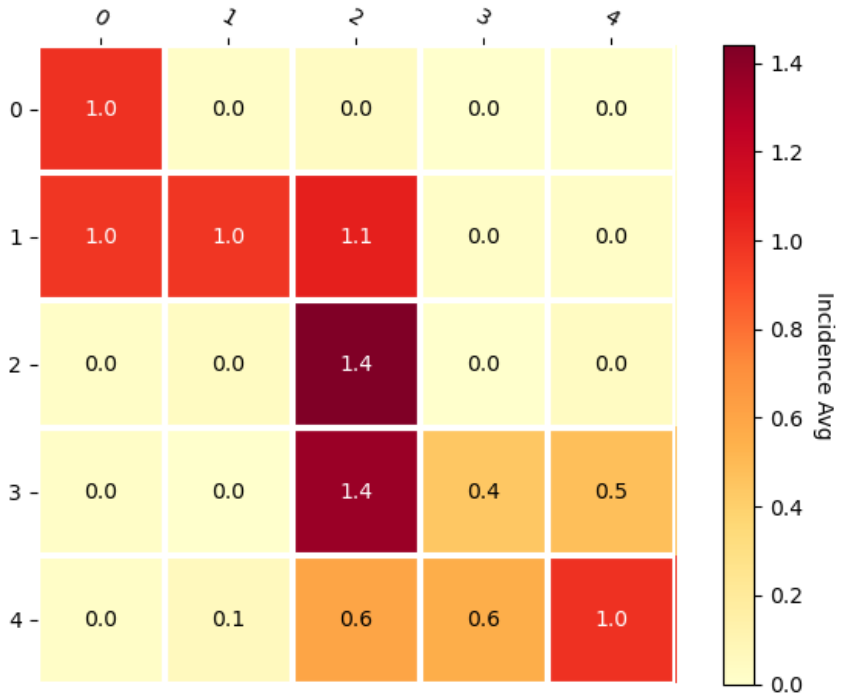


Figure 3.32: Heat map - Q-Table after 2 iteration. Subject 7

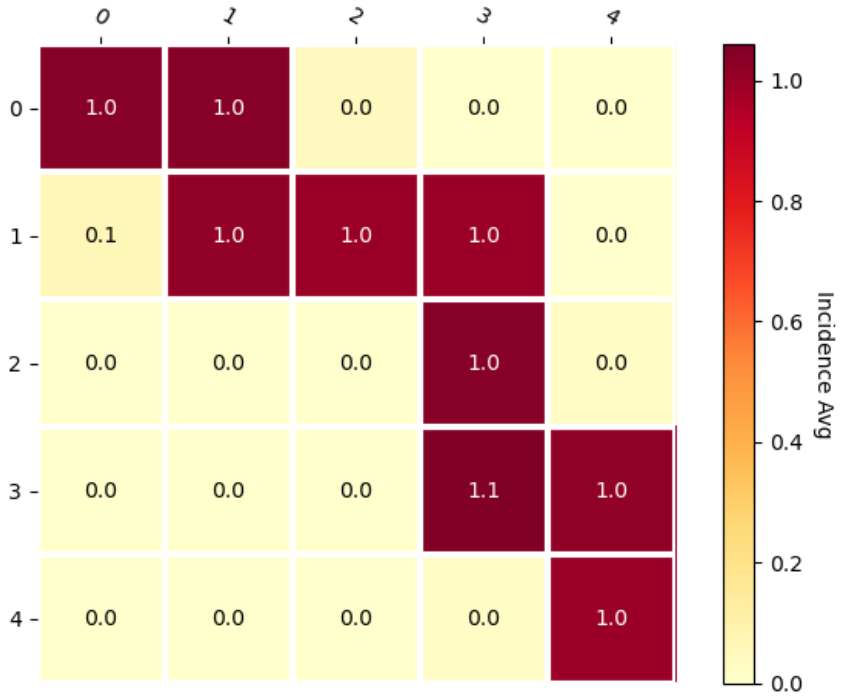


Figure 3.33: Heat map - Final Q-Table. Subject 7

4 Conclusion

After analyzing the results of this work, it was possible to validate that it is feasible to train an agent in real time using reinforcement learning based on ErrP signals. The classifications obtained from the signals can be effective enough for the agent to learn. The fact that it is online requires taking some considerations into account, such as the fact of synchronization, but beyond being a little more complex to implement, it is governed by the same concept and results will at least be equivalent.

When it comes to classifying, both the Support Vector Classifier and Logistic Regression produce good results. Taking into account the combinations tested to obtain the best configuration, it was not possible to obtain a significant difference between the two to choose one over the other. The best configuration of each algorithm depends on the subject in question, that is, the calibration and selection processes are necessary whenever the subject is changed (transfer learning is not possible). The classifications that give the best results are those in which the number of type I errors, that is, those that predict no hits as hits, is low. The existence of type II errors has as a consequence an impact on the speed of learning.

Regarding the use of spatial filters, not all of them are applicable for the detection of ErrPs in EEG signals. These signals are known to have a large amount of noise, therefore, to obtain good results, the spatial filters used must be specifically designed to eliminate it. In particular, PCA is not useful in this type of problem, in fact, it leads to worse results. xDawn, on the other hand, is perfectly suited to the problem being treated, and, in most cases, produces better results, especially when the training data is limited. According to the analysis, the more data that can be used to train the agent, the better the results will be, both in time and in precision. The amount of samples that can be obtained from a subject is limited, due to the fatigue that brainwave sessions can generate. The xDawn application serves to partly make up for this lack of data. If this in turn is complemented by the real-time use of Q-Tables, learning should be not only better but faster.

Once the signals are identified they can be used to train a reinforcement learning algorithm. Although the classification is strictly linked to a subject, rewards earned from different subjects can be used to train the same agent for better results.

Finally, this research verifies that brain signals can be used as an interface between humans and a computer that allows control of the system without explicit user input. Getting better results is only a matter of trying different combinations and additions to processing.

5 Future Work

Like all research work, improvements and extensions can continue. Some exploration possibilities are presented below.

5.1 Full Online Learning

The current implementation aims to be online, but due to limitations, its full potential could not be exploited. The idea would be to make the necessary small adaptations and based on that a comparative analysis between online and offline learning.

5.2 Explore More Spatial Filters

Analyzing other spatial filters can lead to improved results. While other studies do not find a significant difference between xDawn and other denoising filters, it would be a good study area to explore.

5.3 Take More Data

The greater the amount of data, the more foundation the conclusions have.

5.4 Explore More Complex Game Models

The activation of error potentials is related to the ease of distinguishing whenever a mistake is being made. There are other more sophisticated game models to which the same analysis could be applied.

5.5 More EEG Channels

The version of the used EEG cap has 8 channels. There are up to 64 channels in the same type of EEG cap. Increasing the number of channels might produce better results.

5.6 Try Other Reinforcement Learning Algorithms

Other forms of learning might accelerate the process of learning and lower the number of steps the agent need to reach the goal faster.

5.7 Improve Classification

Although the results obtained are sufficient for the agent to learn, alternatives of signal processing might produce better ROC Curves.

6 Acknowledgements

The first person to whom we want to extend our appreciation and gratitude is to our tutor, Dr. Rodrigo Ramele, both for his professional status, which includes guidance and experience, and for his human condition, which includes the predisposition for assistance and patience.

On the other hand, we also want to thank Francisco Bartolome, Juan Moreno, Natalia Navas and José Vitali, whose work was our point of reference and above all, the data that they obtained was the same that we used. Thanks are extended to all those who collaborated with them.

Finally, we want to thank all those people who were by our side all these years, friends, colleagues and family, supporting us when we needed it most and without whom none of this would have been possible.

References

[1] Navas Natalia Vitali José Ramele Rodrigo Moreno Juan Bartolomé Francisco. "Training an agent on brainwaves: using brain signals as feedback for reinforcement learning". In: (2019). URL: https://ri.itba.edu.ar/handle/123456789/1785.

[2] Jonathan Wolpaw and Elizabeth Winter Wolpaw. *Brain-computer interfaces: principles and practice*. OUP USA, 2012.

[3] Gernot R. Müller-Putz Catarina Lopes-Dias Andreea I. Sburlea. "A Generic Error-related Potential Classifier Offers a Comparable Performance to a Personalized Classifier". In: (2020). URL: https://ieeexplore.ieee.org/abstract/document/9176640.

[4] Duo Xu et al. "Playing Games with Implicit Human Feedback". In: *AAAI-20 Workshop on Reinforcement Learning in Games*. New York, New York, USA, 2020. URL: https://www.semanticscholar.org/paper/Playing-Games-with-Implicit-Human-Feedback-Xu-Agarwal/faba141483180e53f461c6035ce95041cfed9a8f.

[5] Urbano Nunes Gabriel Pires. "Error Related Potencials and P300 Event Related Potentials". In: (2017). URL: https://ieee-dataport.org/open-access/error-related-potentials-primary-and-secondary-errp-and-p300-event-related-potentials-\%E2\%80\%93.

[6] Kazuo Kiguchi D.S.V Bandara1. "Brain signal acquisition methods in BCIs to estimate human motion intention". In: (2018). URL: https://www.researchgate.net/publication/327832015_Brain_signal_acquisition_methods_in_BCIs_to_estimate_human_motion_intention_-_a_survey.

[7] Xiaowei Song Shaun Fickling Sujoy Ghosh Hajra Careesa C. Liu. *Developing Brain Vital Signs: Initial Framework for Monitoring Brain Function Changes Over Time*. 2016. URL: https://www.frontiersin.org/articles/10.3389/fnins.2016.00211/full.

[8] Rodrigo Ramele. "Histogram of Gradient Orientations of EEG Signal Plots for Brain Computer Interfaces". In: (2018).

[9] Jose Antonio Urigüen and Begoña Garcia-Zapirain. "EEG artifact removal—state-of-the-art and guidelines". In: *Journal of neural engineering* 12.3 (2015), p. 031001.

[10] Kamelia Niemczyk Jerzy Baranowski Paweł Piątek. "Design and implementation of non-integer band pass filters for EEG processing". In: (2016). URL: https://ieeexplore.ieee.org/document/7760957.

[11] Virginie Attina Guillaume Gibert Bertrand Rivet 1 Antoine Souloumiac. "xDAWN algorithm to enhance evoked potentials: application to brain-computer interface". In: (2009). URL: https://pubmed.ncbi.nlm.nih.gov/19174332/.

[12] Jason Omedes Llorente. "The potential of error-related potentials. Analysis and decoding for control, neuro-rehabilitation and motor substitution". In: (2019). URL: https://dialnet.unirioja.es/servlet/tesis?codigo=257829.

[13] Rodrigo Ramele, Ana Julia Villar, and Juan Miguel Santos. "Histogram of gradient orientations of signal plots applied to P300 detection". In: *Frontiers in computational neuroscience* 13 (2019).

[14] Luis Camarinha-Matos Luis Sebra Lopes. "A machine learning approach to error detection and recovery in assembly". In: (1995). URL: https://www.researchgate.net/publication/3656210_A_machine_learning_approach_to_error_detection_and_recovery_in_assembly.

[15] Christian Niethammer Martin Spüler. "Error-related potentials during continuous feedback: using EEG to detect errors of different type and severity". In: (2015). URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4374466/.

[16] Richard S. Sutton and Andrew G. Barto. "Reinforcement Learning: An Introduction". In: (2014). URL: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

[17] Peter Norvig and Stuart Russell. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press Upper Saddle River, 2009.

[18] Brooke Chan Vicki Cheung Przemysław Dębiak Christy Dennison-David Farhi Quirin Fischer Shariq Hashme Chris Hesse Rafal Józefowicz Scott Gray Catherine Olsson Jakub Pachocki Michael Petrov Henrique Pondé de Oliveira Pinto Jonathan Raiman Tim Salimans Jeremy Schlatter Jonas Schneider Szymon Sidor Ilya Sutskever Jie Tang Filip Wolski Susan Zhang Christopher Berner Greg Brockman. "Dota 2 with Large Scale Deep Reinforcement Learning". In: (). URL: https://arxiv.org/abs/1912.06680.

[19] Su Kyoung Kim et al. "Intrinsic interactive reinforcement learning-Using error-related potentials for real world human-robot interaction". In: *Scientific Reports* 7.1 (2017), p. 17562. ISSN: 20452322. DOI: 10.1038/s41598-017-17682-7. URL: http://www.nature.com/articles/s41598-017-17682-7.

[20] Gen Endo et al. "Mobile follower robot as an assistive device for home oxygen therapy – evaluation of tether control algorithms". In: *ROBOMECH Journal* 2.1 (2015), p. 6. ISSN: 21974225. DOI: 10.1186/s40648-014-0026-3. URL: http://www.robomechjournal.com/content/2/1/6.

[21] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8.3-4 (1992), pp. 279–292. ISSN: 0885-6125. DOI: 10.1007/bf00992698. URL: http://link.springer.com/10.1007/BF00992698.

[22] Sergey Levine et al. "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems". In: (2020). arXiv: 2005.01643. URL: http://arxiv.org/abs/2005.01643.

[23] Yann Renard et al. "OpenViBE: An Open-Source Software Platform to Design, Test, and Use Brain–Computer Interfaces in Real and Virtual Environments". In: *Presence: Teleoperators and Virtual Environments* 19.1 (2010), pp. 35–53. ISSN: 1054-7460. DOI: 10.1162/pres.19.1.35. arXiv: pres.19.1.35. [10.1162]. URL: http://www.mitpressjournals.org/doi/10.1162/pres.19.1.35.

[24] *Lab Streaming Layer (LSL) - Introduction*. URL: https://labstreaminglayer.readthedocs.io/info/intro.html.

[25] Inaki Iturrate, Luis Montesano, and Javier Minguez. "Shared-control brain-computer interface for a two dimensional reaching task using EEG error-related potentials". In: *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*. 2013, pp. 5258–5262. ISBN: 9781457702167. DOI: 10.1109/EMBC.2013.6610735.

[26] *TRANSMISSION CONTROL PROTOCOL - RFC: 793*. URL: https://tools.ietf.org/html/rfc793.

[27] F. Lotte et al. "A review of classification algorithms for EEG-based brain-computer interfaces: A 10 year update". In: *Journal of Neural Engineering* 15.3 (2018), p. 031005. ISSN: 17412552. DOI: 10.1088/1741-2552/aab2f2. URL: http://stacks.iop.org/1741-2552/15/i=3/a=031005?key=crossref.9cd2b15ab65c8ad34b475584b43dc509.

[28] E. Larson D. Engemann D. Strohmeier C. Brodbeck-L. Parkkonen M. Hämäläinen A. Gramfort M. Luessi. *MNE software for processing MEG*

and EEG data, NeuroImage, Volume 86. 2014. URL: https://mne-tools.github.io/0.13/index.html.

[29] E. Larson D. Engemann D. Strohmeier C. Brodbeck-R. Goj M. Jas T. Brooks L. Parkkonen M. Hämäläinen A. Gramfort M. Luessi. *MEG and EEG data analysis with MNE-Python, Frontiers in Neuroscience, Volume 7*. 2013. URL: https://mne-tools.github.io/0.13/index.html.

[30] Alexandre Gramfort et al. "MEG and EEG data analysis with MNE-Python". In: *Frontiers in Neuroscience* 7.7 DEC (2013), p. 267. ISSN: 1662453X. DOI: 10.3389/fnins.2013.00267. arXiv: [1]A.Gramfort,âĂœMEGandEEGdataanalysiswithMNE-Python,âĂİFront.Neurosci.,vol.7,no.December,pp.1âĂŞ13,2013.. URL: http://journal.frontiersin.org/article/10.3389/fnins.2013.00267/abstract.

[31] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. URL: https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf.

[32] Maryam Ahmadi. "Denoising improves visualization of evoked potentials with limited number of trials". In: (2020). URL: https://www.researchgate.net/publication/340241752_Denoising_improves_visualization_of_evoked_potentials_with_limited_number_of_trials.

[33] Iñaki Iturrate José del R. Millán Fumiaki Iwane Ricardo Chavarriaga. "Spatial filters yield stable features for error-related potentials across conditions". In: (). URL: https://ieeexplore.ieee.org/abstract/document/7844316.

[34] Chun-Hsiang Chuang Chin-Teng Lin Tzyy-Ping Jung Dongrui Wu Jung-Tai King. "Spatial Filtering for EEG-Based Regression Problems in Brain-Computer Interface (BCI)". In: (2017). URL: https://arxiv.org/abs/1702.02914.

[35] I. T. Jolliffe. "Principal Component Analysis (PCA). Springer Series in Statistics." In: (2002).

[36] Tian Zhou and Juan P. Wachs. "Spiking Neural Networks for early prediction in human–robot collaboration". In: *International Journal of Robotics Research* 38.14 (2019), pp. 1619–1643. ISSN: 17413176. DOI: 10.1177/0278364919872252. arXiv: 1807.11096. URL: http://journals.sagepub.com/doi/10.1177/0278364919872252.

[37] Wim van Drongelen. "4 - Signal Averaging". In: *Signal Processing for Neuroscientists*. Ed. by Wim van Drongelen. Burlington: Academic Press, 2007, pp. 55 –70. ISBN: 978-0-12-370867-0. DOI: https://doi.org/10.1016/B978-012370867-0/50004-8. URL: http://www.sciencedirect.com/science/article/pii/B9780123708670500048.

[38] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[39] D. Wu, Y. Xu, and B. Lu. "Transfer Learning for EEG-Based Brain-Computer Interfaces: A Review of Progress Made Since 2016". In: *IEEE Transactions on Cognitive and Developmental Systems* (2020), pp. 1–1. URL: https://www.researchgate.net/publication/340643865_Transfer_Learning_for_EEG-Based_Brain-Computer_Interfaces_A_Review_of_Progresses_Since_2016.

[40] Lukas Schlobmuller Frank Kirchner Sy Kyoung Kim Elsa Andrea Kirchner. "Errors in Human-Robot Interactions and Their Effects on Robot Learning". In: (2020). URL: https://www.dfki.de/fileadmin/user_upload/import/11096_20200825_Errors_in_human-robot_interactions_and_their_effects_on_robot_learning.pdf.