Instituto Tecnológico de Buenos Aires
August 2019

Training an Agent on Brainwaves

Using brain signals as feedback for reinforcement learning

Bartolomé Francisco, Moreno Juan, Navas Natalia, Vitali José

Supervisor: Ramele Rodrigo

Final Project

Computer Engineering

Instituto Tecnológico de Buenos Aires

This thesis was written as a part of the Computer Engineering degree at ITBA.

Abstract

The field of Brain Computer Interaction has seen a bloom in the past few years. One of its applications is the training of systems using signals originated from the human brain. Out of all the methods used to acquire information from the central nervous system the one that has gained the most popularity is Electroencephalography (EEG), mainly because of its effectiveness, low cost and because it is noninvasive.

Event-related potential (ERP) are brain signals that are time-locked to a particular event. Error-related potential (ErrP) are a particular type of ERP that are directly connected with error signals. When subjects observe an agent who makes a mistake this potential can be discerned when analyzing the signals which are directly related with the cognitive interpretation of an erroneous outcome or decision.

Reinforcement learning is an approach to learning where an agent tries to maximize the reward obtained while interacting with an unknown environment. It has regained traction in the field of machine learning ever since 2015 when AlphaGo, a reinforcement learning algorithm developed by Google DeepMind, was able to beat the world champion for the complex game Go.

This thesis replicates (I. Iturrate, 2010) and proposes an alternative method to train reinforcement learning algorithms with ErrP signals, captured through EEG, and validate the effectiveness of its use in a prototype application.

Keywords – BCI, EEG, ERP, ErrP, Reinforcement Learning

Contents

List of Figures

List of Tables

1 Introduction

The effectiveness of today's human–machine interaction is limited by a communication bottleneck as operators are required to translate high-level concepts into a machine-mandated sequence of instructions (Thorsten O. Zander and Gramann, 2016). This thesis tackles directly this problem by exploring the use of brain signals as an interface between human and computer, overcoming this limitation by enabling the control of the system without explicit communication from the user.

Using reinforcement learning with reward signals based on brain activity, recorded by an EEG-based BCI system during the task execution, in order to train an agent has been explored in previous research. The paper (I. Iturrate, 2010) has successfully demonstrated that a robot can be controlled by obtaining the reward from brain activity generated while observing the robot solving a task. The present project applies a similar process for making a virtual agent improve its performance using electroencephalography (EEG) signals as feedback for a reinforcement learning algorithm. The idea is that subjects observing a simple simulated system can train such system using only error-related potentials (ErrP) that can be identified in their brain signals. The detection of ErrP signals has been explored in the publication (Ferrez, 2007) which describes the methods that are used in order to identify this specific potential from EEG signals.

In recent years reinforcement learning has seen a come-back. This is mainly because of DeepBrain's AlphaGo who was the first to reach superhuman proficiency in challenging domains, when it won the complex game Go against several world champions. In 2015 AlphaGo won 5 matches against 3-times European Champion, Mr Fan Hui. AlphaGo then went on to compete against Mr Lee Sedol, winner of 18 world titles and considered to be the greatest player until then, and won 4 out of 5 matches in 2016 (David Silver, 2017).

1.1 Brain-Computer Interface

The human brain consists of millions of neurons which control human body behavior relative to internal/external motor/sensory stimuli. Understanding cognitive behaviour of the brain can be done by analyzing either signals or images from it. Human behavior,

such as eye movement, lip movement and so on, are related to specific signal frequency patterns which help understand functional behavior of the complex brain structure.

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 (Wolpaw and Wolpaw, 2012).

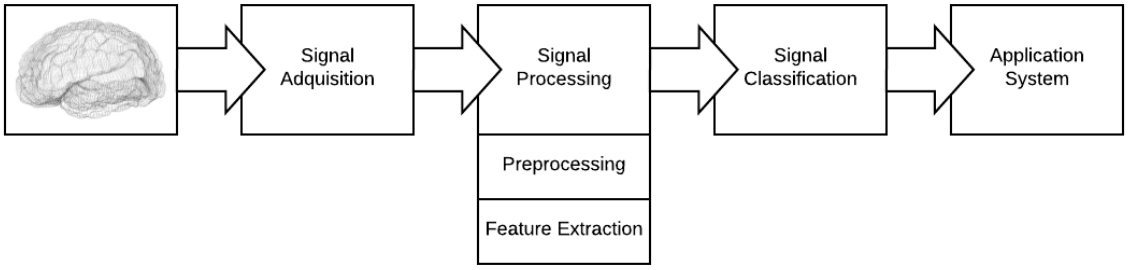


Figure 1.1: BCI system architecture.

A BCI system can be used to control an external device using only brain signals that can be collected with some method of signal acquisition. The architecture is described in Figure 1.1. It has the following 5 main components:

1. An **experiment** that triggers the subject's desired brain activity.
2. A **signal acquisition device** that captures the brain data which is digitized and then transmitted to a computer device for processing.
3. A **signal processing** step which applies necessary filters to eliminate noise and artifacts and that builds a feature in order to distinguish the signals characteristics.
4. A **signal classification** module which infers the information.
5. An **application system** that uses the generated information to affect some external device.

1.1.1 Signal Acquisition

A neural interface is the hardware device used to detect, digitalize, and transmit brain signals measurements. BCI systems can be non-invasive or invasive and can be used to

control different types of effectors (e.g., computer cursor, switch, robotic arm), so their design and functionality varies. The design and functional requirements of a particular BCI are driven by the BCI's intended use and its intended target population. Neural interfaces must be safe, signals must have enough information to support its use, the interface must be reliable and the degree of invasiveness must be as limited as possible (Wolpaw and Wolpaw, 2012). The measuring technique determines the most important taxonomic differentiation in BCI, according to the methodology that is applied to extract the information from the CNS (Ramele, 2018).

BCI neural interfaces currently fall into the following major classes:

- **Electroencephalographic** (EEG): Non-invasive method that allows reading electrical activity of the brain. Described in further detail in section 1.1.1.1.
- **Electrocorticographic** (ECoG): Electrode arrays that are positioned in the brain surface through invasive surgery, with moderate information from small localized sets of neurons and synapses. Signals are believed to reflect mainly synaptic and other source activity occurring over a substantial portion of the depth of local cortex.
- **Miniaturized Microelectodrode Arrays**: Micro-electrode arrays that are inserted into the cerebral cortex through invasive surgery, that record neuronal action potentials from individual. Measures local field potentials (LFPs) that are recorded within brain (usually cortical) tissue and mostly reflect current sources related to synaptic activity.
- **Magnetoencephalography** (MEG): Non-invasive technique for reading magnetical activity in the brain by using sensitive magnetometers. Electrical activity in the brain generates minuscule magnetic fields. It is usually used as a compliment of EEG.

Invasive methods require complex craniotomy surgery. 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 a method for visualizing and recording the electrical activity of the brain in a non-invasive manner with relatively low information. It uses electrodes placed along the scalp to measure the voltage fluctuations of the electrical pulses that result from the activity of the brain cells. Each electrode is called an EEG channel. The variation of the potential difference measured from each channel is displayed as peaks and troughs in a line graph as seen in figure 1.2. It is currently the most used method for gathering information because its simple, non-invasive and inexpensive.

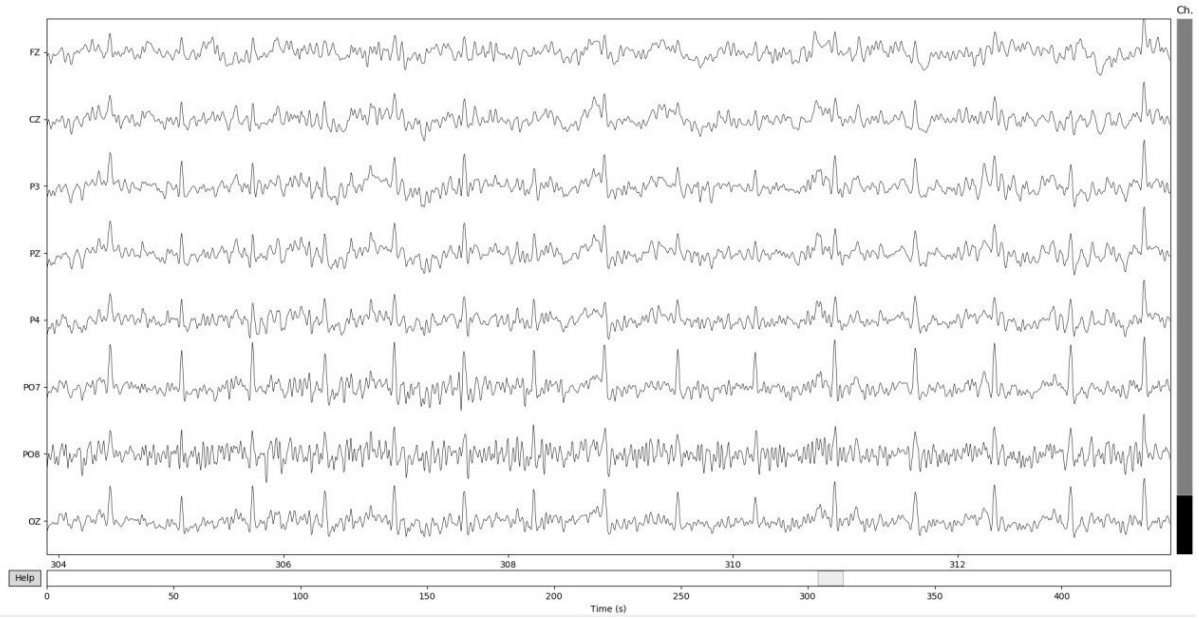


Figure 1.2: Sample EEG signal obtained from (g.Nautilus, g.Tec, Austria) displayed as a peaks and troughs in a line graph from eight channels

1.1.2 Signal Processing

The purpose of a BCI is to detect and quantify characteristics of brain signals and to translate these measurements into the desired device commands. The brain-signal characteristics used for this purpose are called features. 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(Wolpaw and Wolpaw, 2012).

Before the feature extraction step, a signal pre-processing step is applied. This step

enhances the relevant aspects of the signals by preemptively eliminating known interference or irrelevant information (Wolpaw and Wolpaw, 2012), such as noise and artifacts that contaminate the signal. 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 decontamination can be done by applying different filters to the signals such a a band-pass filter or a frequency filter depending on the signal characteristics that are expected to be collected.

The feature extraction step consists of identifying and extracting the important features that make the signals distinct and separate them from irrelevant information in order to build a feature vector made out of various measurements that identifies the signals at the desired moment. After the feature vector is generated, the signals should be classified and transformed into an appropriate command for device control.

1.1.3 Signal Classification

The features described in 1.1.2 represent indirect measurements of the user's intent and they must be translated into appropriate device commands that convey that intent (Wolpaw and Wolpaw, 2012). This is done with a model that processes the feature vector at a given time and outputs the recognized command. In order to identify a command the model should be trained so it knows what a given feature vector represents.

There are some events that trigger an specific kind of EEG signals that can be detected and acted upon. These are called event-related potentials (ERP). They are voltage fluctuations in the ongoing electroencephalogram that are time-locked to an event (Kappenman and Luck, 2011). 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. Error-related potentials (ErrP) are a type of of ERP and are the kind of signals that this project is interested on. This type of signal is describen in futher detail in section 1.1.3.1.

1.1.3.1 Error-Related Potential

ErrP are signals that occur as a response when a subject observes a mistake being made during a task. There are three type of ErrP signals, response ErrP, feedback ErrP and observation ErrP. Response ErrP is measured after 100 ms following the erronueous response. Feedback ErrP is measured 250 ms following presentation of a feedback that indicates that there has been a mistake. Observation ErrP result from the observation errors made by an operator during tasks where the operator needs to respond to stimuli (Ferrez, 2007).

1.2 Reinforcement Learning

Reinforcement Learning is a learning method that is based on the concept of learning from interaction, it automates goal directed learning and decision making. It assigns a numeric reward to every action dependant on a particular state with the objective to maximize said reward. Which actions are better are not directly indicated, instead the learner must discover which are the actions that yield the most reward. This is done by synthesizing a mapping function between situations and actions by maximizing a performance measurement of the desired behavior. Hence, a reinforcement signal provided by the reinforcement function (RF) evaluates the entered situation relative to the desired behavior. After this, the agent receives either positive or negative reinforcements according to the utility i.e. desirability of the situation entered as a consequence of the performed action (Juan Miguel Santos, 1999).

The agent must exploit what it already knows and explore new states. When choosing actions that the agent has chosen in the past and have yielded prizes it is exploiting its previous experiences, but it also needs to explore new options in order to be better experienced in the future. If not, it might not be learning some crucial information of other states which might actually yield higher rewards. The agent must try a myriad of actions and progressively prioritize those that have appeared to be the best in the past (Andrew Barto, 2018).

Reinforcement learning allows, at least in principle, to bypass the problems of building an

explicit model of the behavior to be synthesized or a meaningful learning base needed for supervised learning (Juan Miguel Santos, 1999).

1.2.1 Elements of Reinforcement Learning

The two main elements of reinforcement learning are the **environment** and the **agent**. Apart from those, the other four main elements are a **policy**, a **rewards system**, a **value function** and a **model of the environment**.

The agent is an en entity capable of perceiving its environment and react accordingly in a rational way through effectors, which implies trying to maximize a given result or reward. The interaction between the agent and the environment can be seen in figure 1.3.

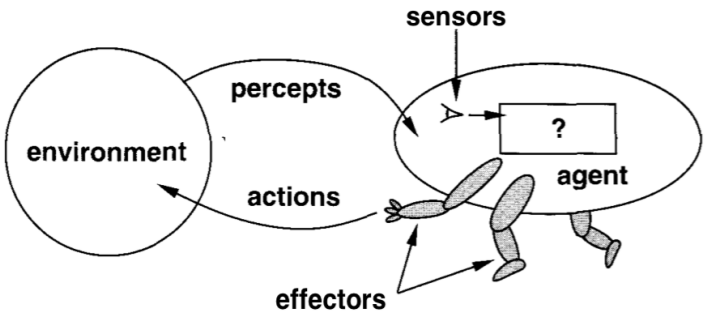


Figure 1.3: Agent interacting with environment (Norvig and Russell, 2009)

The policy is the mapping of the action that an agent is supposed to take at a given state. It might be a table or simple function, but it might also be more sophisticated computation methods. They are usually stochastic, with probabilities for each action.

The reward system determines the goal of the problem, in each step the environment sends the agent a number called the rewards, which corresponds to the value that the agent is trying to maximize. These values are used to determined the optimal policy, when an action is followed with a low reward then the chances of a policy outcome determining that action in the future will become lower. On the other hand, if the reward is higher then the chances of choosing the given action are higher.

The value function determines the accumulation of rewards given an action and a state. It will tell what is good in the long term, not immediately based upon one action. This is used because one action might seem better than another in the immediate step but

when reaching the final goal, another action might have yielded a greater accumulation of rewards. By avoiding to choose the action with the highest immediate result the state reached might have better rewards in the next steps. The value is calculated and re-calculated from the sequences of observations made by an agent in its lifetime. This differs from rewards, which are provided directly by the environment.

The model of the environment mimics the environment's behaviour, and allow inferences to be made about how the environment will behave.

1.2.2 Q-Learning

Q-Learning is a model-free reinforcement learning. In order to represent rewards, a matrix is used, where rows correspond to all the possible states, and columns correspond to all possible actions. This matrix is known as the Q-Table. The reward for an action given a state is the value that can be found for said row and column. In each iteration a value is chosen by picking the action that will maximize the reward for the current state. On each iteration the new Q value is obtained by the following function:

$$Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma Max(s, a) - Q(s_t, a_t)] \tag{1.1}$$

The function takes the current Q value and adds another term. α represents the learning rate, a number between 0 and 1, that determines the proportion of exploration vs. exploitation. When α equals 0 the agent doesn't learn at all and when it equals 1 it only considers the most recent term. γ represents the discount rate, that is a number between 0 and 1 ($0 \leq \gamma \leq 1$), and determines the importance of future results. When γ equals 0 then it will consider only the most recent rewards, instead if it equals 1 it will consider the longer term higher results. $R(s_t, a_t)$ corresponds to the reward obtained when passing from state s_t to s_{t+1} by taking action a_t. An episode of the algorithm ends when the reached state (s_{t+1}) is a terminal state.

The following pseudo-code describes the algorithm used to train the Q-table.

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

Initialize $Q(s, a)$, for all $s \in S$, $a \in A(s)$, arbitrary except that $Q(terminal, \cdot) = 0$

for *each episode* **do**

  Initialize S

  while S *not terminal* **do**

    Choose A from S using policy derived from Q (e.g., $\epsilon - greedy$)

    Take action A, observe R, S'

    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \epsilon \max_a Q(S', a) - Q(S, A)]$

    $S \leftarrow S'$

  end

end

Algorithm 1: Q-Learning (off-policy TD control) for estimating $\pi \approx \pi_*$

2 Materials and Methods

The project consists of collecting signals from a person's brain while they are watching a game where the agent has knowledge of the way it's played but not how to win it. Hence, the agent can learn using the person's feedback to get better at the game. This section describes the process of obtaining the person's brain signals and how they are used to make the agent improve it's performance.

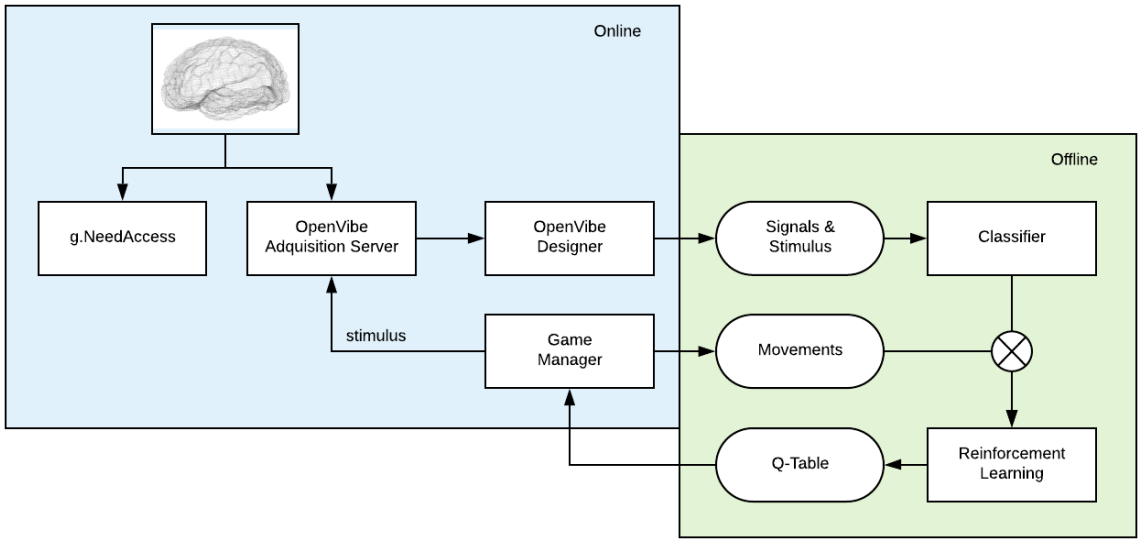


Figure 2.1: Overview of the experimental procedure.

The experimental procedure is summarized in Figure 2.1. The core of the procedure is the retrieval of the subject's brain activity. This process is called brainwave session. In order to fetch this data, subjects use a wireless digital EEG device during the experiment. Once the subject has the headset on, a set up procedure is executed as described in Section 2.1.3 using a program called g.NeedAcceess to do an impedance check and to visually inspect the EEG signals in order to check if it's correctly applied. Subsequently, the OpenVibe Acquisition Server program is launched and configured. This program is part of the OpenVibe platform (Yann Renard and ecuyer, 2010) and has the responsibility of receiving the signals data from the headset and stimulus information from the game and transfers it to the OpenVibe Designer application. After this step, the Game and the OpenVibe Designer programs are launched and configured to communicate with the previously mentioned acquisition server. Once the subject is ready the computer is

positioned in front of them and the brainwave session starts. A brainwave session consists of several experiences, each of them being a game run, and it uses a pseudo-random function to determine the agent's movements. The system is described in more detail in Section 2.1.1 and the brainwave session in Section 2.1.3. Once the experimental procedure is done, the game states and the signals data with the stimuli channel information of each run is stored. For each run, the signals and stimulus information are then passed to the classification module that uses them to train the classifier. The classification step is covered in detail in Section 2.2. After this step is complete, the game movements and the trained classifier are used to update a Q-Table for each experience. Lastly, the calculated Q-Table is used to test if the agent has boosted its performance while playing the game. This module is explained in Section 2.3.

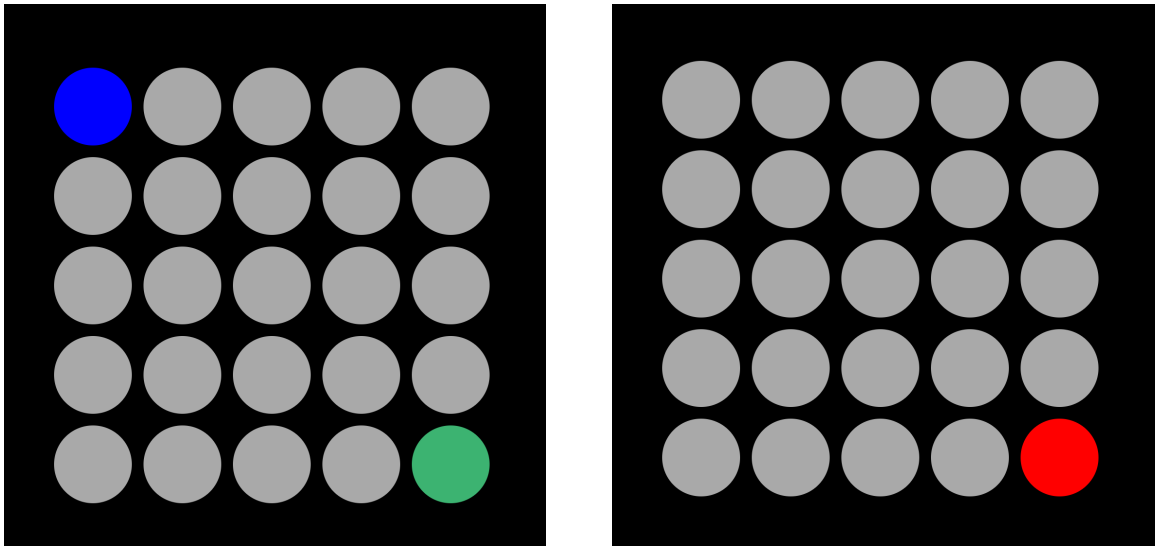
2.1 Cognitive experiment

This section details the game system characteristics, the brainwave session process and the retrieval and analysis of the generated data from the subjects interaction with the system.

2.1.1 Game details

The system consists of a $5x5$ grid where there are two lights, one representing the current position of the agent, and the other one representing the goal, shown in figure 2.2. The objective of the system is for the agent to reach the goal. The light representing the goal remains static at the bottom-right position of the grid, while the one representing the position of the agent always starts at the upper-left position of the grid. The grid is represented by circular spots with a black background. The agent's position has a blue light, the goal position has a green light, and the rest of cells are represented with grey spots. When the agent reaches the goal, the position where the agent and the goal are located turns red, showing that the experience has ended. There are four possible movements that the agent can perform: it can go upwards, downwards, towards the left and towards the right, as long as the move doesn't make the agent leave the grid. The movement direction is selected randomly and is executed once every 2 seconds. When

the experience ends, there is a pause of 5 seconds before the next experience starts. The experience is designed for it to be evident whenever there is an error (when the agent moves away from the objective) so the subject can notice it immediately after the stimulus is presented, possibly triggering a cognitive response.



(a) Initial state of the grid. **(b)** Final state of the grid.

Figure 2.2: Grid system representation used in cognitive experience.

2.1.2 Experimental setup

For the brainwave sessions, subjects are recruited voluntarily. They are given a consent form with questions regarding their health (previous health issues, particular visual sensitivity, etc.), habits (sleeping hours, caffeine consumption, etc.), as well as an approval petition to collect the required data. The brainwave sessions are performed with 8 subjects, 5 males and 3 females, average age 25.125 years, standard deviation 1.54 years, range of 22-28 years. All subjects have normal vision, are right-handed and have no history of neurological disorders.

After the form is filled out, a short description of the brainwave session is given to the subject. They are only told that the objective of the agent is to reach the goal and the four movements that the agent can make. When this concludes, the subject is introduced to the wireless digital EEG device (g.Nautilus, g.Tec, Austria) that they have to wear during the brainwave session. It has 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

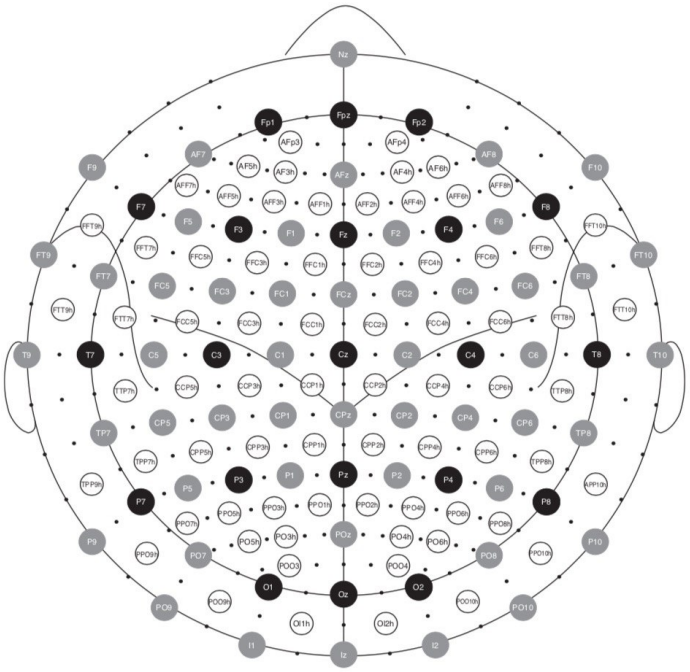


Figure 2.3: The standard 10–20, 10–10, and 10–5 electrode montages. The 10–20 montage is indicated by the 21 electrodes shown as black circles. The 10–10 montage consists of the 21 electrodes of the 10–20 montage plus 53 additional electrodes indicated in gray. The additional electrodes of the 10–5 montage are indicated by the black dots and the open circles.(Wolpaw and Wolpaw, 2012)

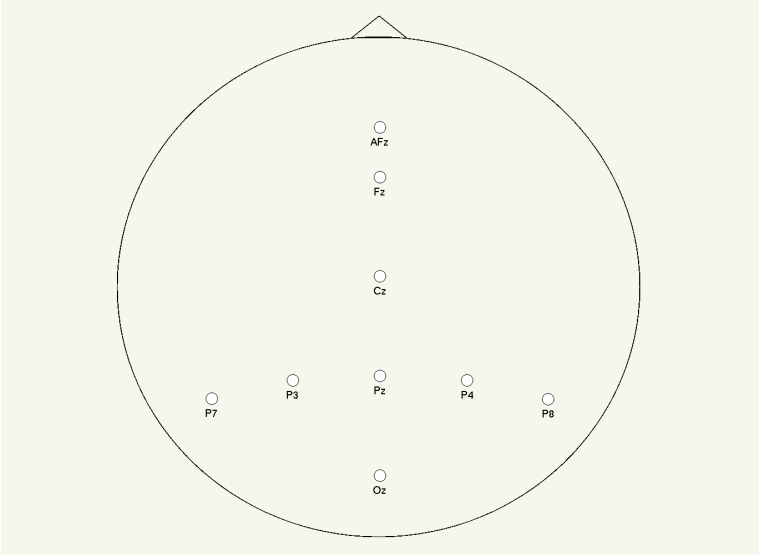


Figure 2.4: 8 channels and ground positions used for the experiment described in 2.1.3

International System, with a reference set to the right ear lobe and ground set as the AFz position. The electrodes contact points are adjusted applying conductive gel until the impedance values displayed by the program g.NeedAccess are within the desired range. This process takes between 10 and 15 minutes. After this step, the subject is instructed

to close their eyes and the same program is used to check the live channel values so that there are no dead ones and the expected values are displayed for eye movements or muscle chewing.

Once the headset is correctly applied, the OpenVibe Acquisition Server program is launched and configured with with a sampling rate of 250Hz and a 50Hz notch filter is applied to filter out power line noise as well as an additional bandpass filter between 0.5Hz and 60Hz. Data is handled and processed with the OpenVibe Designer platform using 8 channels for the brain data (one channel per electrode) and an additional channel for the stimuli. When everything is connected, the subject is seated in a comfortable chair in front of a computer screen. The brightness of the screen is set to the maximum setting to avoid any visual inconvenience in which the subject can not distinguish the components of the game that appear on the screen.

2.1.3 Data acquisition

The data is collected in two brainwave sessions of up to three experiences each. The amount of experiences executed varies depending on the level of fatigue suffered by the subject. After all sessions have concluded, the subject is given another form in which they give feedback of the sessions to check if they found them boring and if they had no difficulty concentrating or understanding it.

The output of each experience is a file in the BrainVision [1] format containing the information of the 8 channels and the stimuli channel. Each channel contains all the samples taken (at 250 Hz) on that given channel throughout the brainwave session. As explained before, 8 of those channels correspond to each of the electrodes of the headset, while the other one corresponds to the stimuli given. Four types of stimuli can be discerned: the start of the experience, the end of the experience, a correct action (when the agent moves closer to the objective), and an incorrect action (when the agent moves further from the objective). All these stimuli are called events, and each of them has a specific numeric value that identifies them. Every other sample that does not correspond to an event has 0 as value.

[1] Link to BrainVision MNE documentation: https://martinos.org/mne/dev/manual/io.htmlbrainvision-vhdr-vmrk-eeg

In order to control the classification of the events, noise signals are created and classified, in order check the obtained results. The classifier is expected to perform poorly using this kind of signal. This in turn would result in the agent not improving (or even decreasing) its performance. This noise signals are generated using the OpenVibe Designer platform and are random values between 0 and 1 with a uniform distribution. The epoch size, sampling frequency, and number of channels is the same as in the experiences conducted with the subjects.

Figure 2.5: Subjects during the experiment.

2.2 Signal processing and classification

The goal of this module is to build a classifier that is able to identify whether the action taken by the agent is an error or not. It is developed in Python using the MNE environment (A. Gramfort, 2013), which is a package designed specifically for processing EEG and MEG data, and the machine learning library Scikit-Learn.

Many classification methods are considered. The best classifier is selected based on a calibration and selection procedure that is further explained in sections 2.2.2.1 and 2.2.2.2. After an optimal algorithm and a configuration are selected, a classifier is trained for each subject and then epochs from their experiences are classified. These classifications are then used by a reinforcement learning algorithm to train the agent. This last algorithm is explained in detail in section 2.3.

2.2.1 Signal processing

This step consists of processing the collected signals in order to train a classifier that can decide whether an error potential is triggered or not. Firstly, the output of a brainwave session is read and a band pass filter of 0.1-20.0Hz is applied to the signal, as seen in Figure 2.7. Samples that correspond to the start of an event are tagged using the data from the stimuli channel.

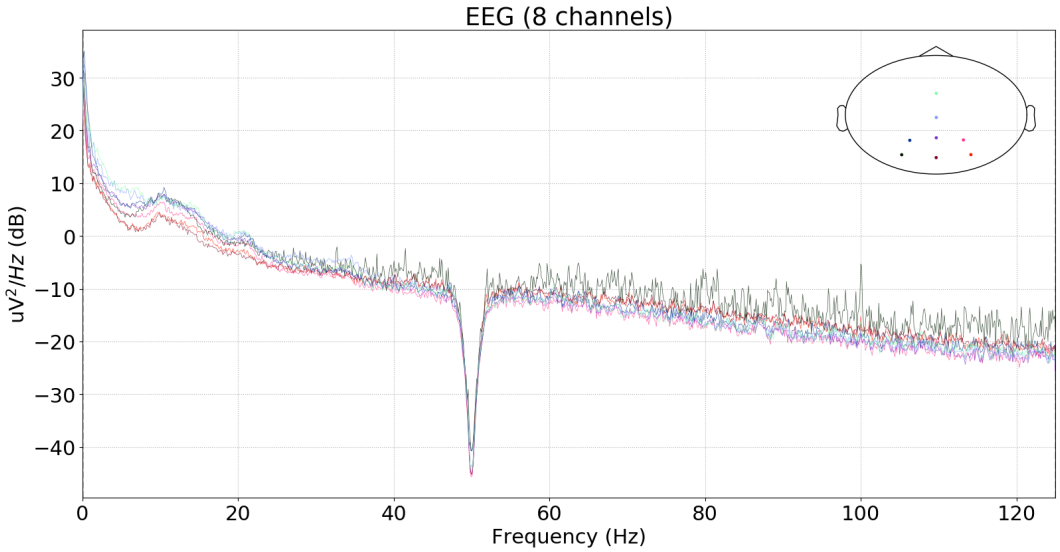


Figure 2.6: The power spectral density (PSD) of the measurements of a raw dataset.

After the data is loaded and tagged, epochs are extracted from the raw data. Epochs consist of all the sample points that take place between the start of the event and 2 seconds later (time for each action to take place), resulting in 500 samples per channel, as the sample frequency is 250 Hz.

Samples that do not correspond to an epoch are not used. Also, epochs referring to the start or finish of the experience are excluded. This is done because the start and the end of the experience doesn't involve the agent taking an action, thus giving signals that should not be considered in the classification process.

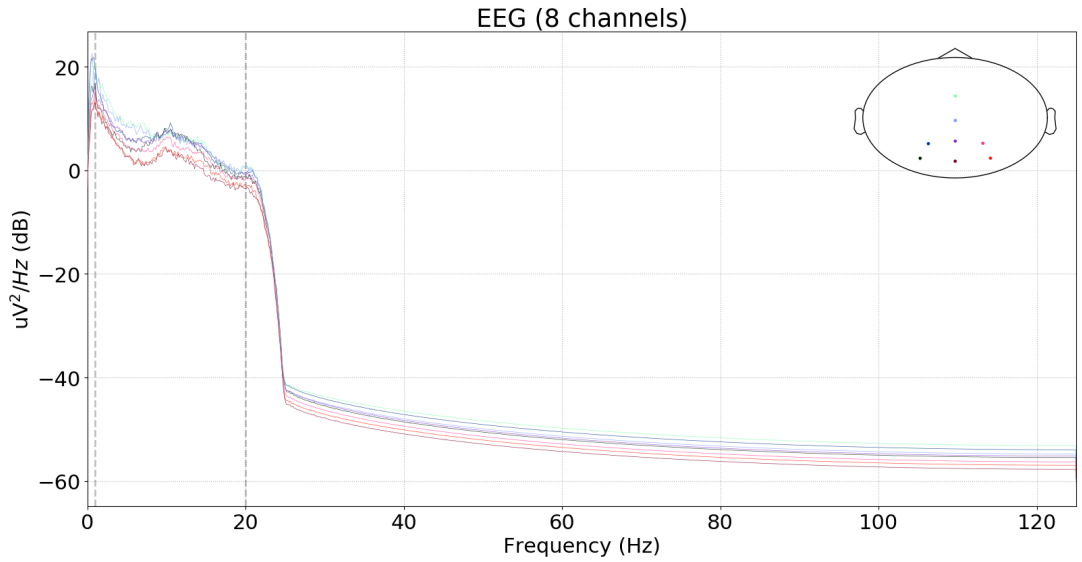


Figure 2.7: The power spectral density (PSD) of the measurements of a raw dataset after applying a band pass filter of 0.1-20.0Hz.

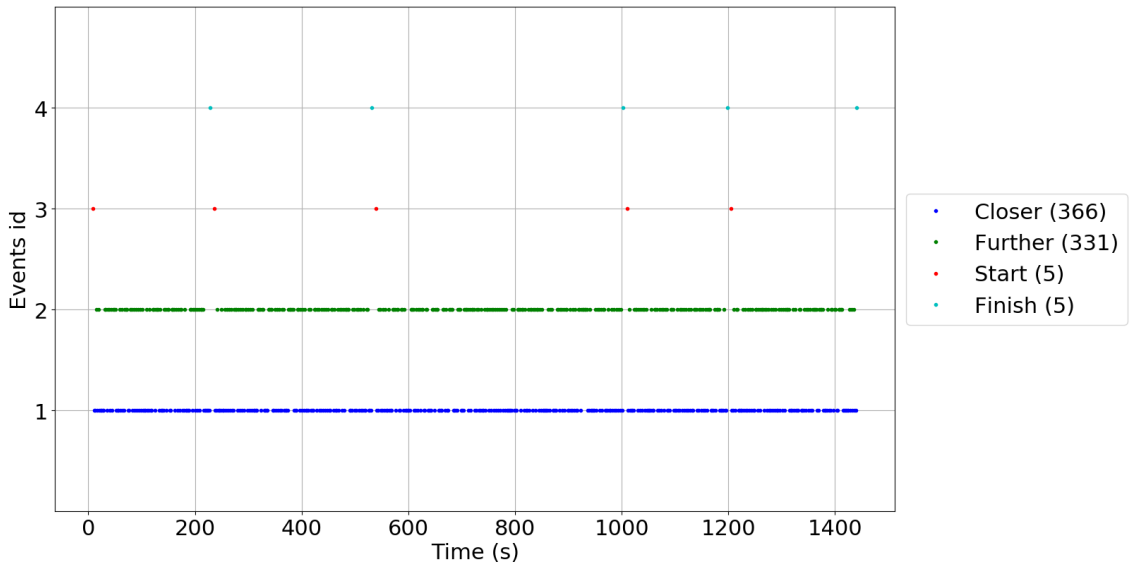


Figure 2.8: Example of a dataset showing all event types.

In this way, the raw data of an entire brainwave session is processed into an array of experiences where each element is an array of epochs tagged with a number specifying if the epoch corresponds to an action that made the agent move further from the goal (hit) or an action that made the agent move closer to the goal (no hit). The ErrP is expected to be found in hits. To get the data ready for classification, the stimuli channel is removed in order to classify the signals using only the EEG data and a mne.decoding.vectorizer [2]

[2]Link to vectorizer documentation: https://mne.tools/dev/generated/mne.decoding.Vectorizer.html

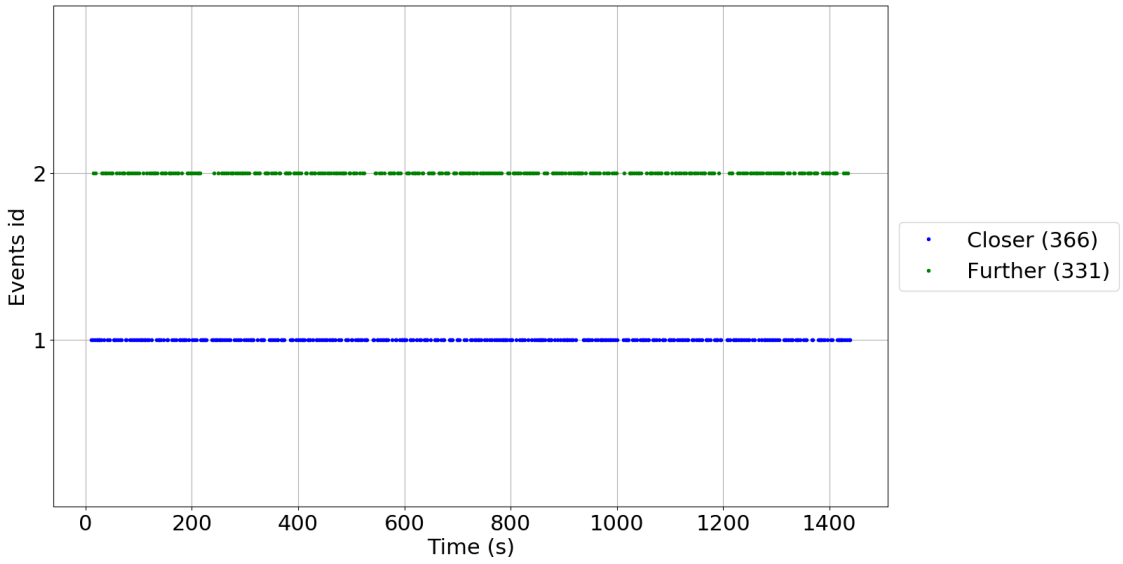


Figure 2.9: Example of a dataset showing only further and closer event types.

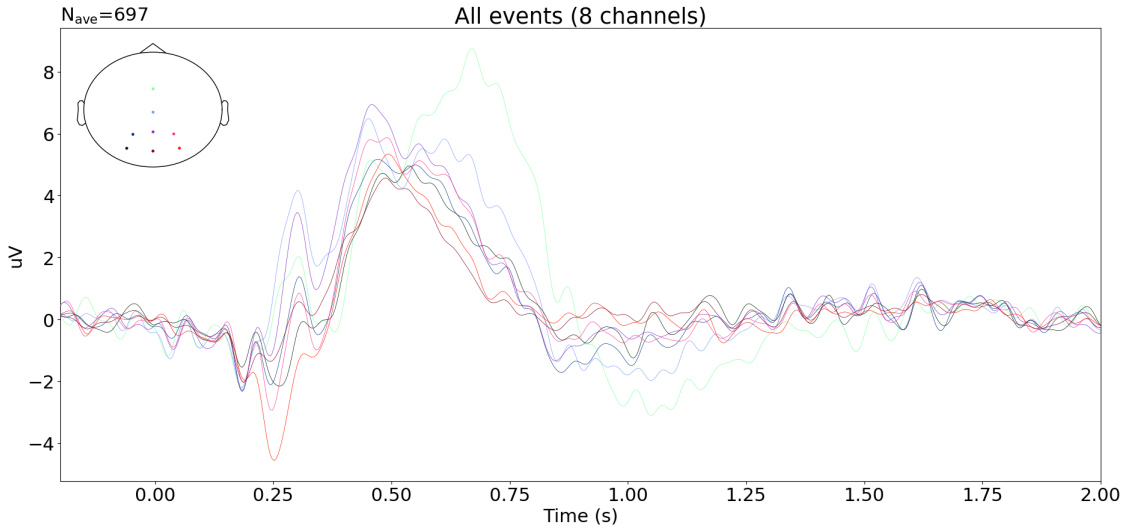


Figure 2.10: Grand average signal values of all epoch types of a measured dataset.

function is applied to transform the data into a single array sample. Lastly, this data is used by the classification module as information to train and test a classifier.

2.2.2 Classification algorithm selection

Different classification algorithms are compared in order to find the one which is best suited for the problem at hand. These are: logistic regression, support vector classifier, random forest, and a multilayer perceptron. In order to choose the ideal algorithm, each one is individually calibrated in order to find the best parameters and then they are

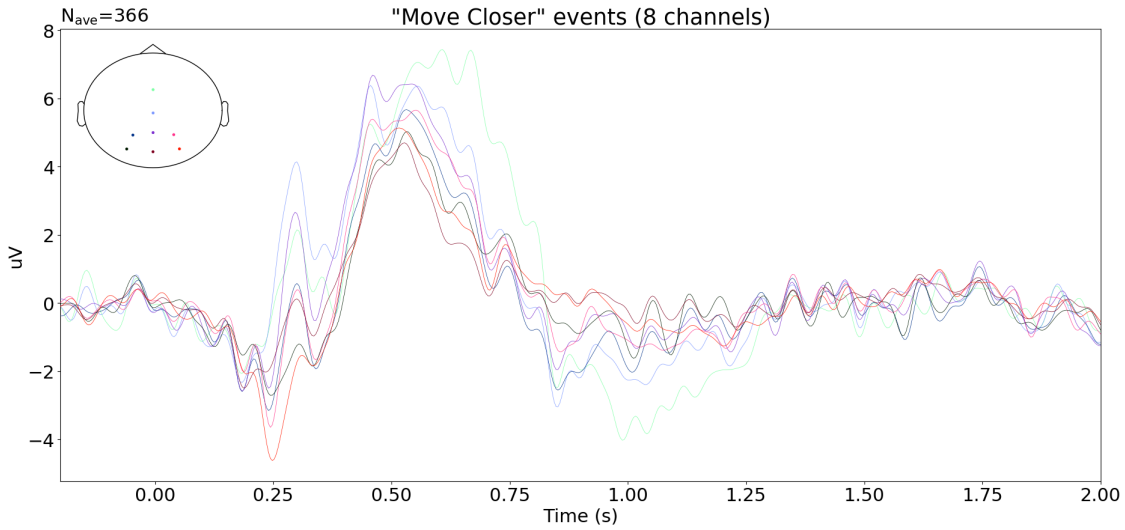


Figure 2.11: Grand average signal values of 'move closer' epoch types of a measured dataset.

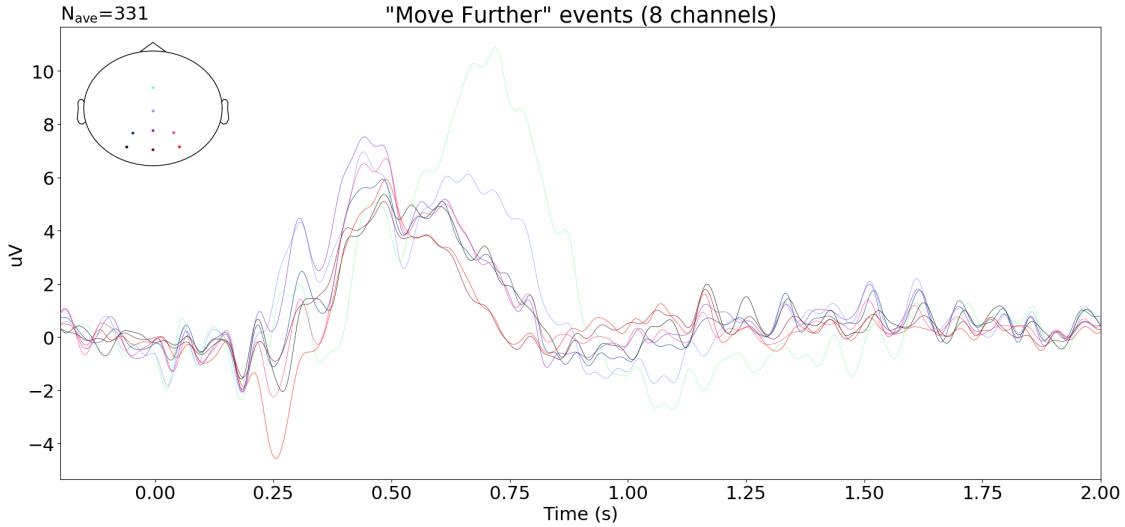


Figure 2.12: Grand average signal values of 'move further' epoch types of a measured dataset.

compared between themselves. Two calibration plots are used to compare the different algorithms and parameters, as seen in section 3.1.2.

2.2.2.1 Calibration

The classifier calibration process consists on finding the set of parameters that yields the best results when classifying. This technique is called hyperparameter optimization. The selected approach for this optimization is grid-search, which performs an exhaustive

search through an specified set of parameters for a given classifier. The process begins by splitting the data into train and test (60% and 40%), where only the training subgroup is used for the grid-search calibration. For scoring each configuration, a process of k-fold cross validation is performed to ensure generalization capability of the training data. Each configuration is scored 10 times, and the average of them is taken as the final score of the configuration. The configuration with the best score is selected for each classification technique.

2.2.2.2 Comparison

Once the best configuration for each classification technique is calculated, the resulting classifiers are trained using the training samples and then compared against each other. With these results, the optimal configuration is used in subsequent tests. For this comparison, the remaining test data from the initial split is used (40% of the original data set).

2.2.2.3 Performance

When the optimal classifier is found and built, it is trained for each subject and tested. When classifying, a confusion matrix is calculated alongside a receiver operating characteristic (ROC) curve to show the performance of the classification.

A confusion matrix provides a visual way of assessing the classifiers performance and how well it classifies particular labels. It shows the percentage of events with a certain predicted label that actually belong to that group, and the percentage of events with a predicted label that belong to the other classification group. This is shown for each of the two possible labels (hit and no hit). This type of graph is helpful to identify both type of classification errors:

- Type I Error: Samples are classified as positive when they are actually negative. (Top-right of confusion matrix)
- Type II Error: Samples are classified as negative when they are actually positive. (Bottom-left of confusion matrix)

A ROC curve shows the rate of false positive classification versus the rate of true positive classification. If the curve is above the identity function, the classifier is considered to perform better than a pure chance classifier. In contrast, if the curve is below the identity then the classifier is under-performing. It should be noted that in that case, the classifier could be modified to give the inverted prediction, resulting in a good classifier. This is why the desired result is a curve that maximizes the distance to the identity.

These results can be seen in section 3.1.3.

2.2.2.4 Implementation testing

In order to validate that the classifier implementation works according to expectations, a sample data set is used. The sample data-set used is the one provided in (Spüler and Niethammer, 2015b). The paper describes an experiment where the subject has to play a simple video game (depicted in Figure 2.13). The subject uses the right thumbstick of a gamepad to control the angle in which the cursor moves on the screen. The task consists on avoiding collisions of the cursor with blocks dropping from the top of the screen with a constant speed. The speed of the falling blocks is set to a level so that the game is challenging and the agent collides with a block from time to time. In case of a collision, the game continues for 1 second and then stops. The delay of 1 second is introduced to make sure that the reaction measured in the EEG originates from the subject recognizing the collision (outcome error) and not from the game stopping or restarting. To study the execution error, which is happening when the interface delivers erroneous feedback, the angle of the cursor movement is modified for the duration of 2 seconds. The degree of modification is randomized (45°, 90°, 180° to either the left or the right side). The time between two execution errors is randomized to be between 5 and 8 seconds. (Spüler and Niethammer, 2015a).

2.2.3 Rewards generation

The set of experiences of each subject is split into training and testing, so that the results of the classification can then be used to improve the performance of the agent. After training the classifier, the testing experiences are classified. Then, for each experience a

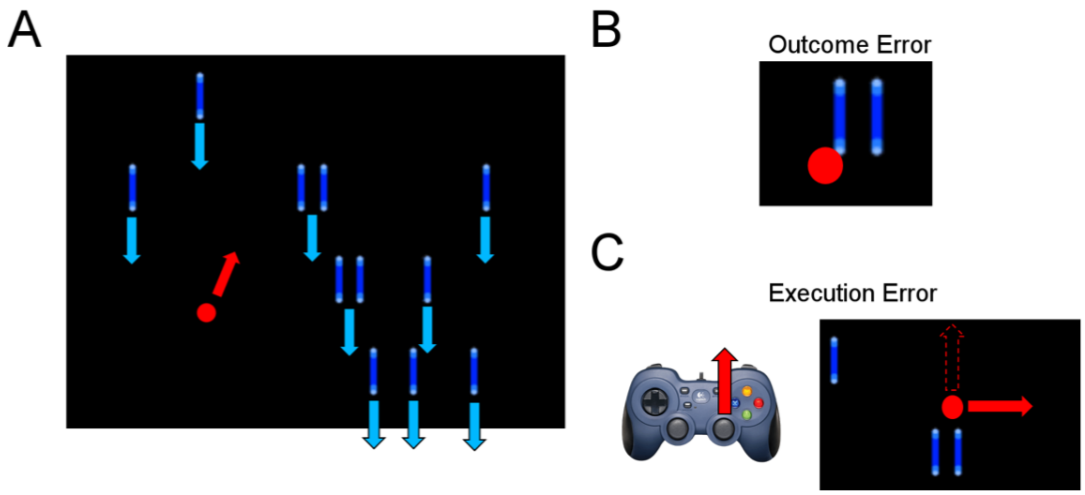


Figure 2.13: Dataset experiment representation.

reward file is generated. For this purpose, game states files, which are submitted by the game when an experience takes place, are used. These files contain an ordered list of the states of the game throughout an experience. With this file and the classification of the test data of an experience, a reward file is submitted. This file specifies a reward for each movement in the game, based on the classification of the event that corresponds to that movement. 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 this rewards depends on the performance of the classifier. This reward file is used by the Q-Learning algorithm to improve its performance, as detailed in section 2.3.

2.3 Q-Learning

A Q-Learning algorithm is used to train the agent. It is developed in Python and uses the OpenAI Gym toolkit. Gym is a toolkit for developing and comparing reinforcement learning algorithms. It makes no assumptions about the structure of an agent, and is compatible with any numerical computation library, such as TensorFlow or Theano.

2.3.1 Algorithm implementation

The Q-Table is initialized with zeros, unless a preexisting Q-Table is passed as a parameter. In order for the agent to learn from the feedback generated by the subject, the Q-Table is not used to determine the action to take at a given state. Instead, the policy which determines which action to take in a particular state is given by the steps taken from the brainwave session results. This allows to train the Q-Table based on the subject's feedback from the movements the agent took, which are chosen pseudo-randomly, while executing the brainwave session. The previously mentioned feedback is not explicit as it comes from the interpreted brain signal data, which is collected while the agent is executing the brainwave session and then each action is classified as an error or not. This implies that the reward is determined by the subject's brain activity. This is covered in further detail in 2.3.2.

While using the previously mentioned step function, the Q-Table is updated in each iteration. This is done following the Equation 2.1.

$$Q(state, action) \leftarrow Q(state, action) + \alpha[reward + \gamma * Max(state, action) - Q(state, action)] \tag{2.1}$$

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

2.3.2 Environment

The environment is the representation of the game mentioned in 2.1.1. It defines the game's states and actions and keeps track of the current state. It determines the next state given the current one and an action, and decides the corresponding reward.

The Gym library provides an easy way of developing an environment. It only requires the implementation to inherit from Gym's class *Gym.Env*. The main methods that have to be implemented are step, reset and render. Reset resets state to the initial position and sets the amount of steps to zero. Render prints the matrix representing the state to

simple output. The *step* function iterates from one state to the next. The implementation returns four variables:

- Observation (object): agent's observation of the current environment, the current state of the environment.
- Reward (float): Reward returned after previous action takes place.
- Done (boolean): Indicates if the state reached is a final state, if *step* is called again after this state has been reached, calls will return undefined
- Amount of steps (integer): Amount of steps since the beginning of the experience.

The custom environment developed is called $MentalChaseEnv$. In this implementation the actions and rewards are given by the rewards file mentioned in 2.2.3. This implies that 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.

2.3.3 Testing Implementation

In order to test that the algorithm mentioned in 2.3.1 is working correctly, a similar algorithm is developed with a different step function. In this implementation the Q-Table is used to determine which action should be chosen given a state. In this case the action is chosen greedily with some noise, as shown in equation 2.2.

$$action \leftarrow Max(sort(Q[state,:] + random(actionSpace) * (1/(iterationStep + 1)))) \quad (2.2)$$

This version of the Q-Learning implementation is first tested using one of the available environments in OpenAI Gym, $FrozenLake - v0$. This environment is further described in the apendix section A.

After this test, a custom environment called $ChaseEnv$ is developed and used to test the Q-Tables that are trained using $MentalChaseEnv$.

For this environment the *step* method receives the action as a parameter, and evolves the state depending on it.

The reward function is calculated based on the current distance to the goal, if it has increased compared to the previous step then the rewards is negative, if not it returns zero. In order to test how the Q-Table is trained when the accuracy is not perfect the reward function doesn't always return the correct reward. It calculates a random number between 0 and 1, and if it is larger than the accuracy then it will return 0 even if it should return -1 as the reward.

The reward function is described in algorithm 2.

$reward \leftarrow 0$
if *currentDistanceToGoal* $>$ *previousDistanceToGoal* **then**
    $reward \leftarrow -1$
else if *random.uniform(0, 1)* $<$ *accuracy* **then**
    return reward
else
    return 0
end

Algorithm 2: Reward Calculation for *ChaseEnv*

3 Results

3.1 Signal Classification

3.1.1 Algorithm Calibration

The first step in order to be able to find ErrP signals is to choose the most efficient algorithm, and the proper calibration. In order to do this different parameters are tested for a set of algorithms and for each individual subject. Initially a sub-selection of experiments is used to define a subset of parameters to test with, and then all the data is tested with this subset of parameters.

3.1.1.1 Logistic Regression

For the logistic regression algorithm the parameters that are tested are C and the Solver. The C parameter is the inverse of the regularization strength. The values this parameter can take are 0.001, 0.01, 0.1, 1 and 10. The solver parameter refers to the algorithm that is used in the optimization problem, the ones that are tested are $lbfgs$ and $saga$. The different configurations that resulted as optimal for subjects can be seen in the apendix section B.1.

Logistic Regression		
Subject	Score	Configuration Number
1	0,672	5
2	0,639	6
3	0,631	4
4	0,626	4
5	0,610	1
6	0,594	2
7	0,588	3
8	0,524	6

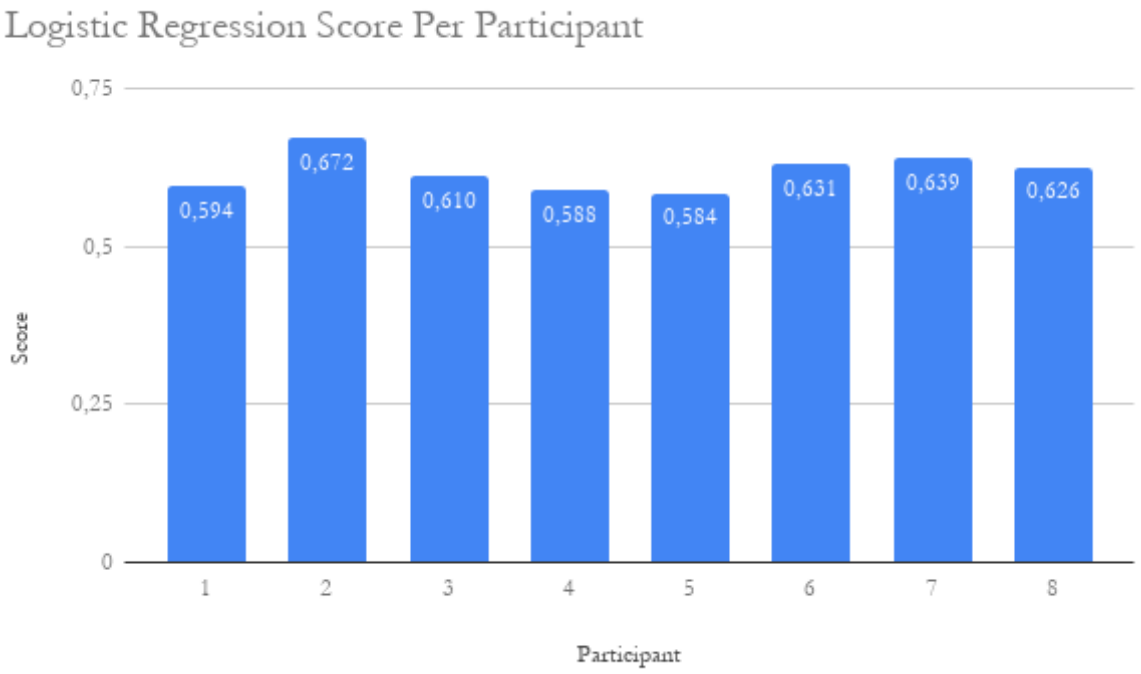


Figure 3.1: Logistic Regression Algorithm Calibration

3.1.1.2 Multi-layer Peceptron

For the multi-layer perceptron the tested parameter is α, which refers to the regularization term parameter. This parameter can equal 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}, 10^{-9} or 10^{-10}. The configurations that resulted optimal for subjects can be seen in the appendix Section B.2.

Multi-layer Perceptron		
Subject	Score	Configuration Number
1	0,591	2
2	0,625	3
3	0,591	3
4	0,627	1
5	0,520	3
6	0,600	3
7	0,605	1
8	0,621	2

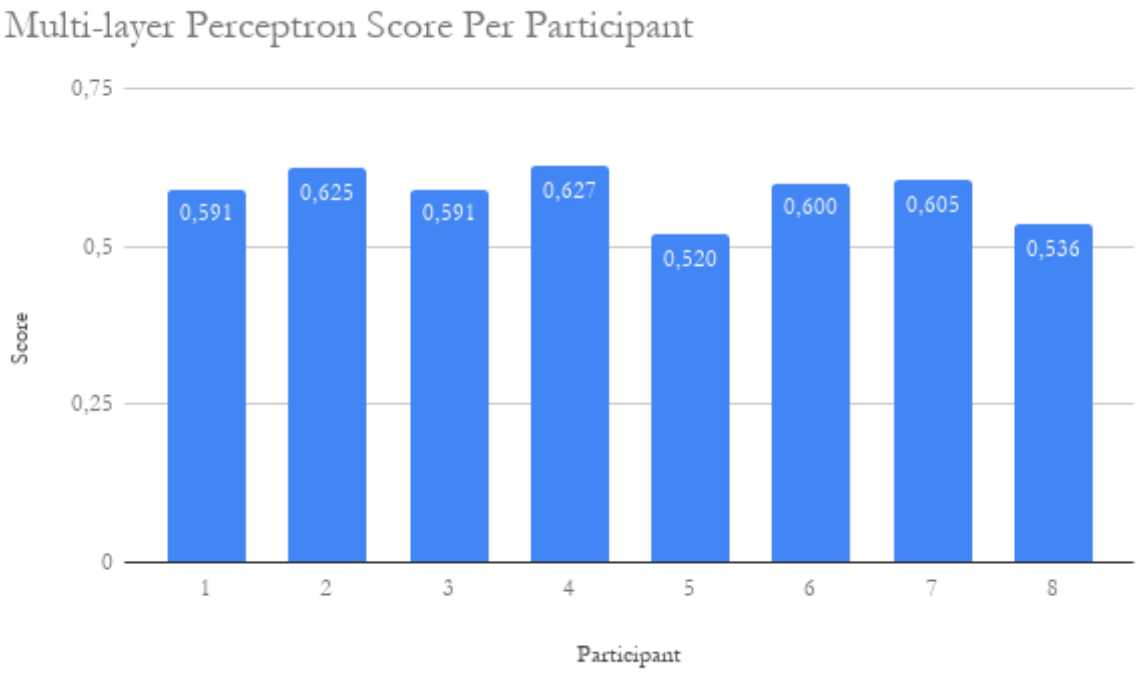


Figure 3.2: Multi-layer Perceptron Algorithm Calibration

3.1.1.3 Random Forest

When testing the random forest classifier the parameters that are tested are the maximum amount of features and the criterion. Regarding the maximum amount of features the parameter values that are tested are auto and $log2$. This parameter refers to the number of features that are considered when looking for the optimal split. Regarding the criterion the ones that are tested are *gini* and *entropy*. This is the criterion that is used in order to qualify the quality of a split. These are the only two options for this parameter, *gini* refers to the Gini impurity and entropy to the information gain. The possible configurations that resulted optimal for subjects can be seen in the appendix section B.3.

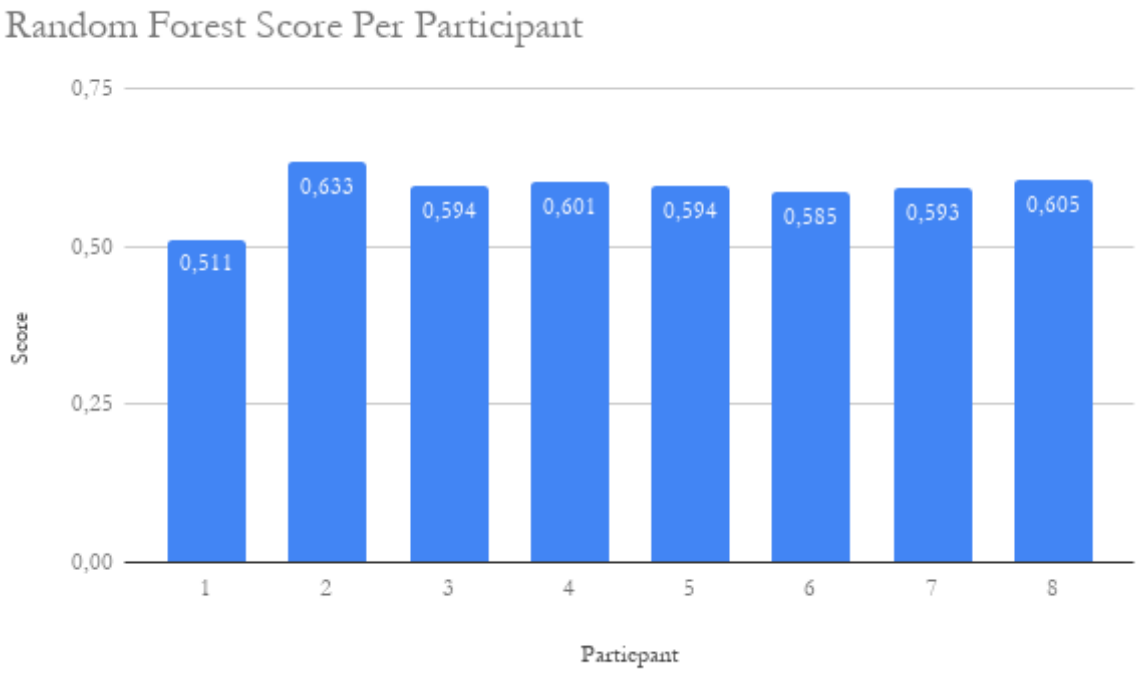


Figure 3.3: Random Forest Algorithm Calibration

Random Forest		
Subject	Score	Configuration Number
1	0,511	1
2	0,633	2
3	0,594	1
4	0,601	1
5	0,594	1
6	0,585	3
7	0,593	3
8	0,605	3

3.1.1.4 Support Vector Classifier

In the support vector classifier the parameter that is tested is C, which is the penalty parameter of the error term. The possible values C can take are 0.001, 0.01, 0.1, 1 and 10. The configurations that resulted ideal for subjects can be seen in appendix section B.4.

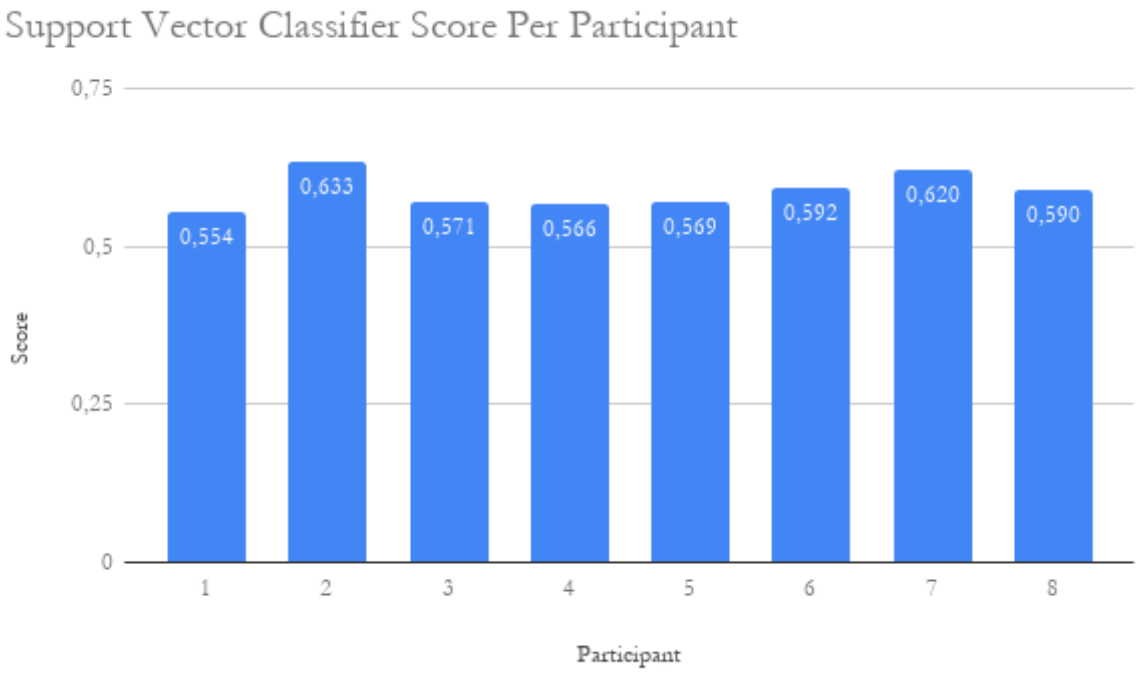


Figure 3.4: Support Vector Classifier Algorithm Calibration

Support Vector Classifier		
Subject	Score	Configuration Number
1	0,554	2
2	0,633	2
3	0,571	2
4	0,566	2
5	0,569	2
6	0,592	2
7	0,620	2
8	0,590	1

3.1.1.5 Score Obtained For Optimal Configuration Per Algorithm

Figure 3.5 shows the score for each algorithm for each subject using the optimal configuration.

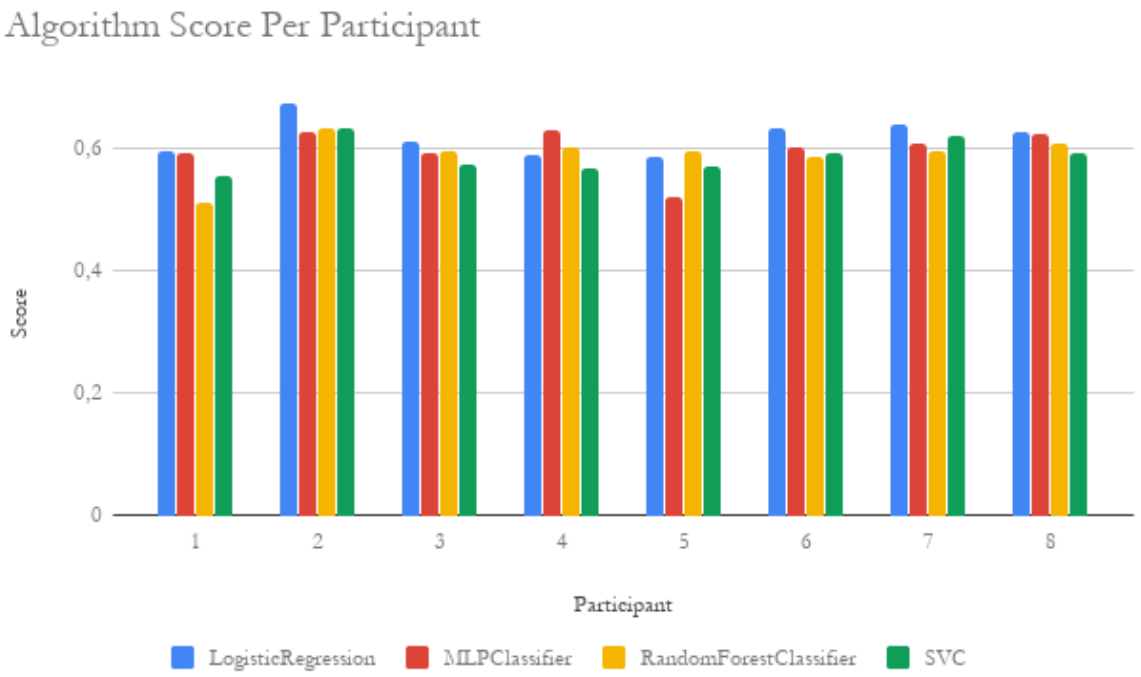


Figure 3.5: Score Obtained With The Optimal Calibration Per Algorithm Per Subject

3.1.2 Algorithm Selection

In order to select the best algorithm to use when training, another comparison is made between algorithms. In this instance the comparison is made for each test subject, using the optimal set of parameters for each algorithm, which where found in the previous subsection. This comparison consists on validating the percentage of samples correctly classified for a given prediction probability. For instance, a well calibrated classifier would classify correctly 60% of the samples that were classified with a prediction probability of 0.6. This gives a sense of how reliable the classifier is when giving a prediction probability.

Also, a comparison regarding the amount of samples classified given a prediction probability is made, to see how many samples are classified with a very high or low level of confidence.

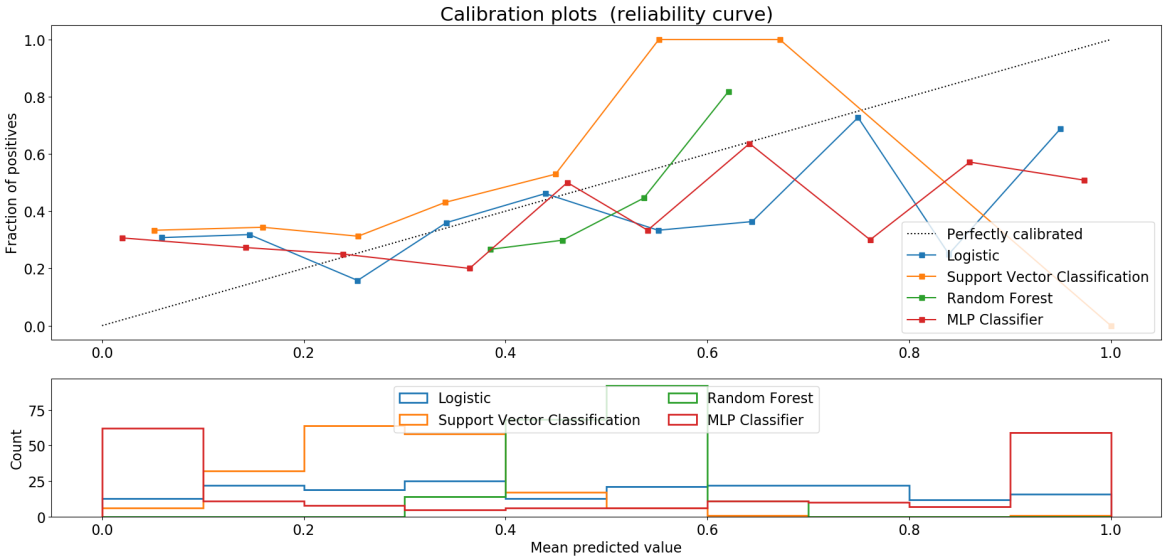


Figure 3.6: Algorithm Classification: Subject 1

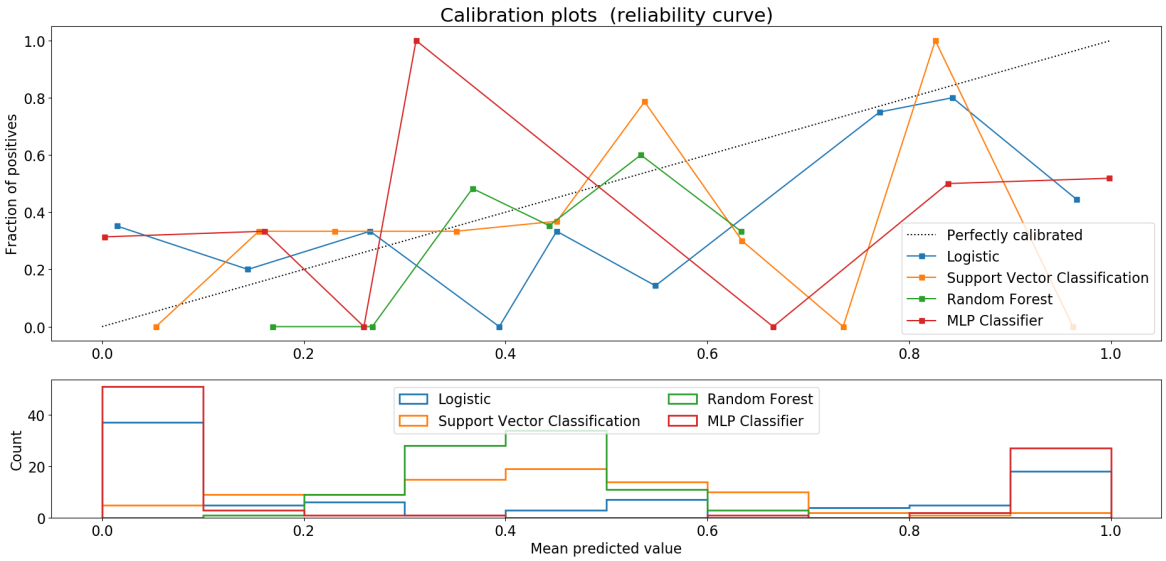


Figure 3.7: Algorithm Classification: Subject 2

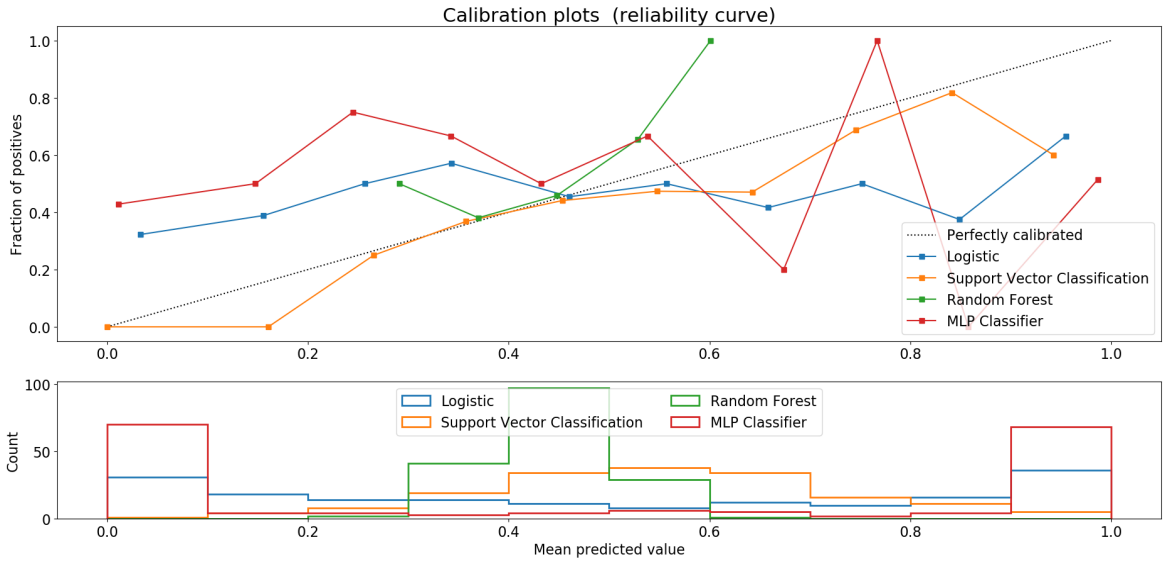


Figure 3.8: Algorithm Classification: Subject 3

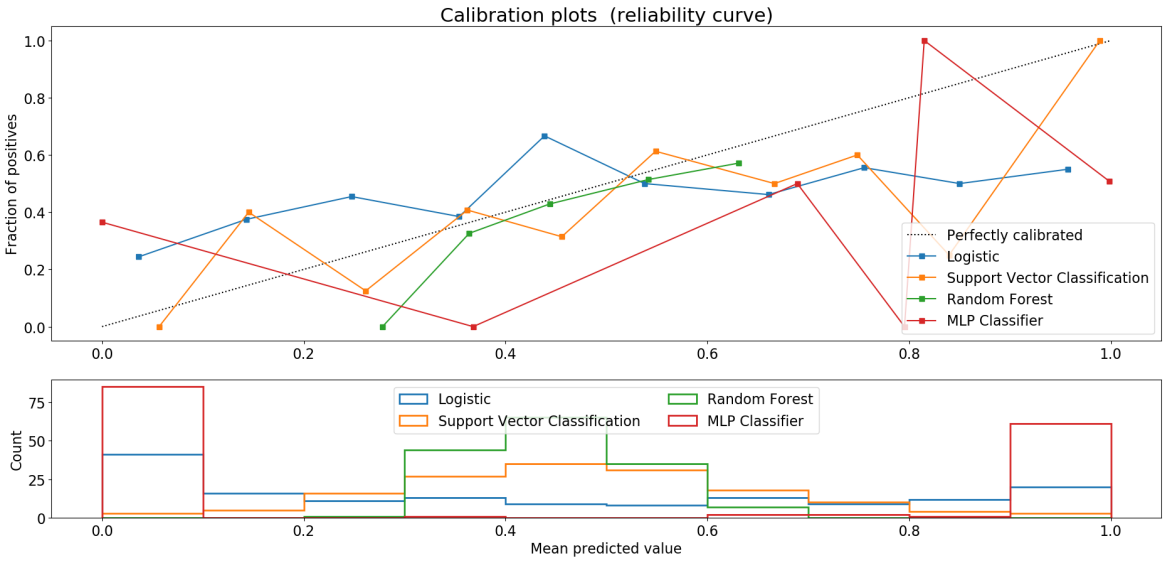


Figure 3.9: Algorithm Classification: Subject 4

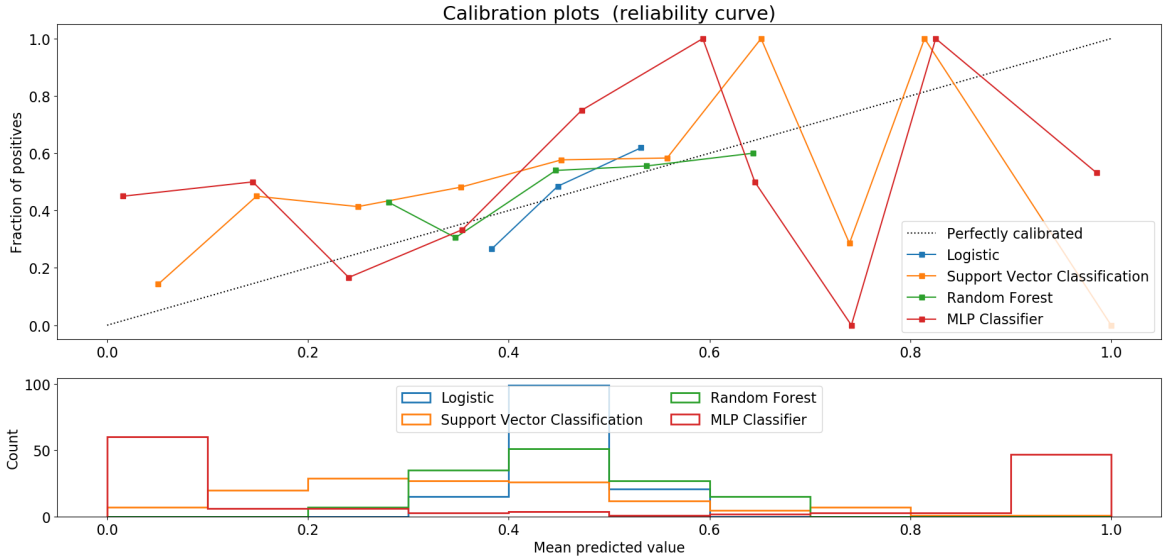


Figure 3.10: Algorithm Classification: Subject 5

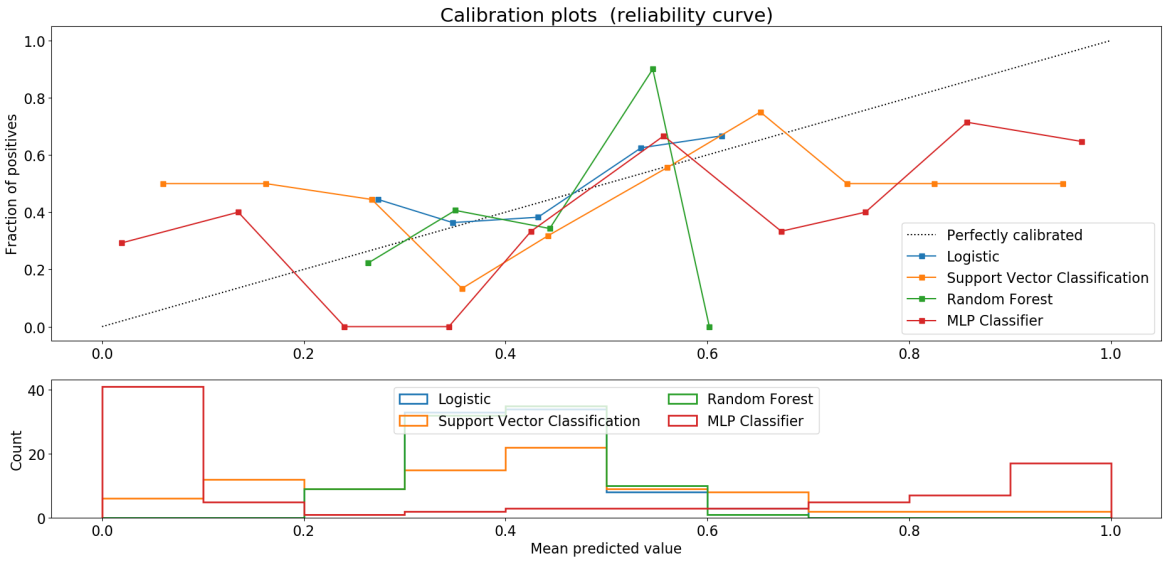


Figure 3.11: Algorithm Classification: Subject 6

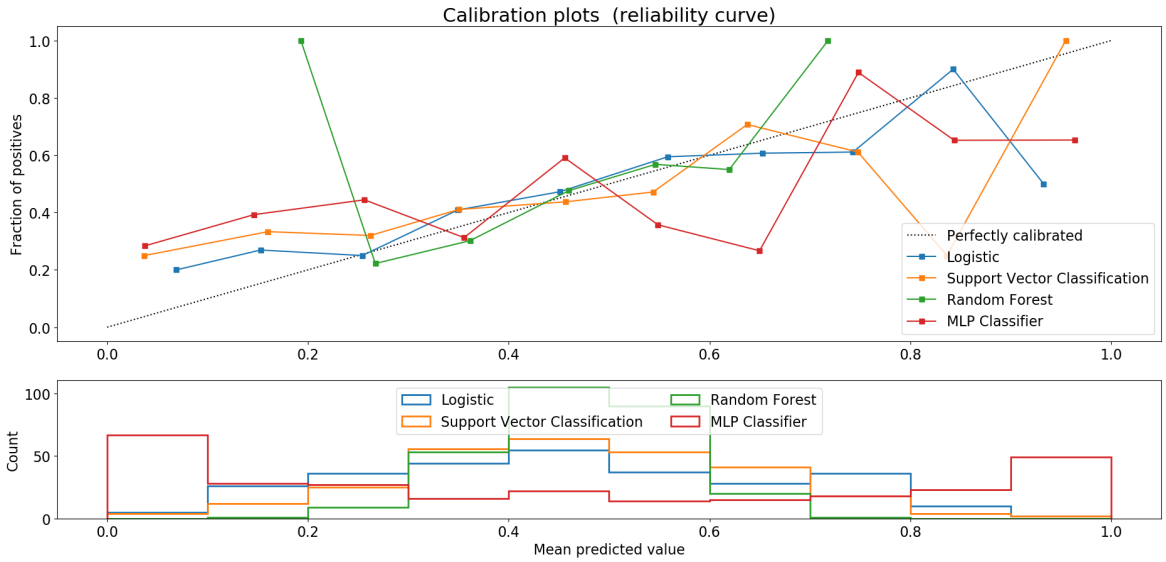


Figure 3.12: Algorithm Classification: Subject 7

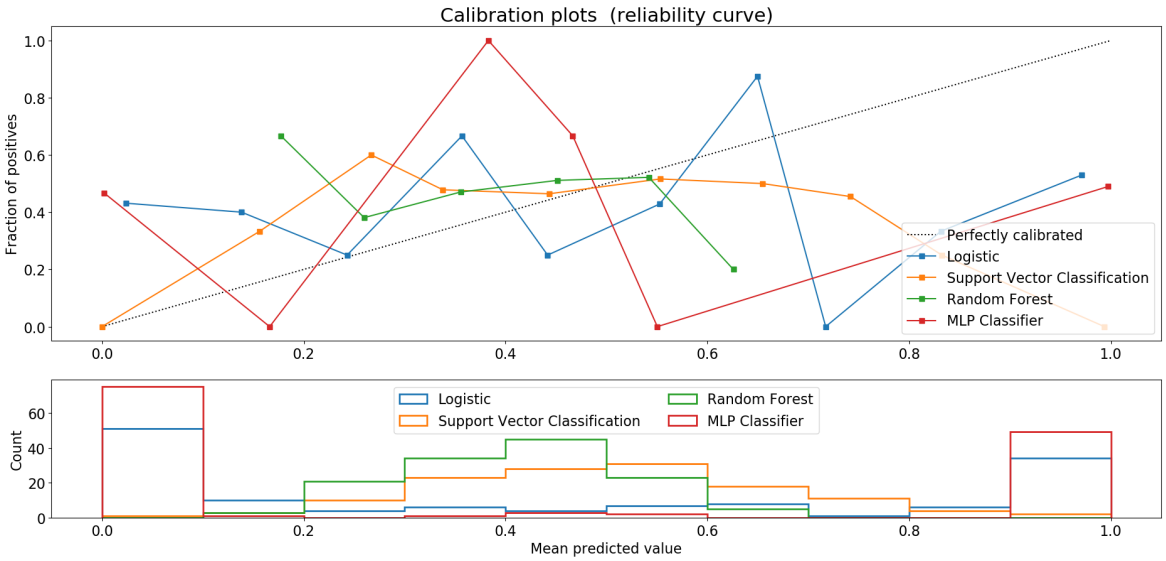


Figure 3.13: Algorithm Classification: Subject 8

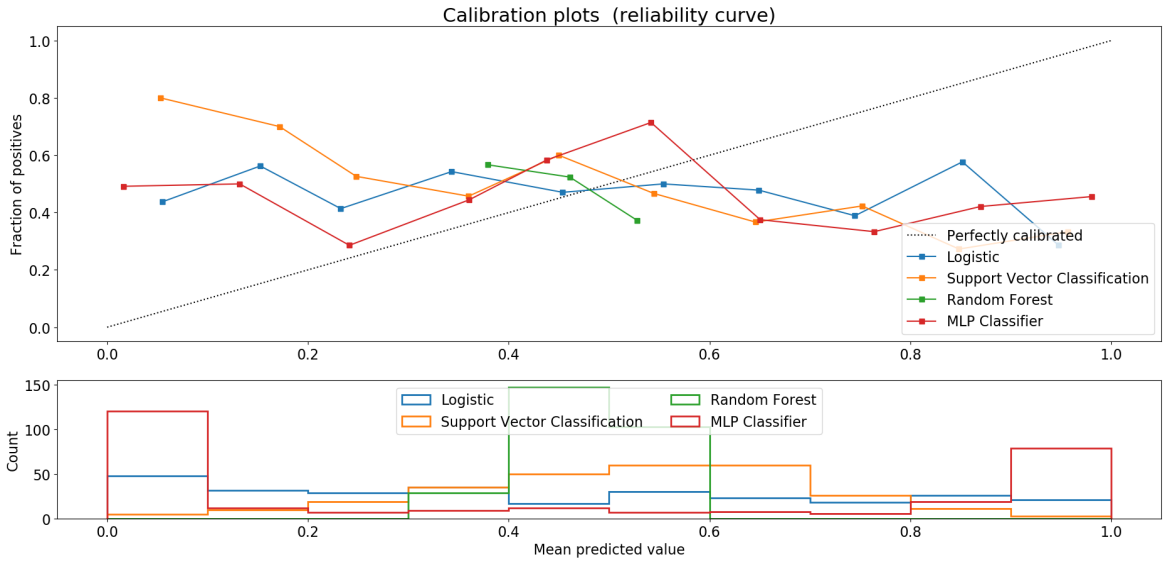


Figure 3.14: Algorithm Classification: Noise

3.1.2.1 Optimal Algorithm and Configuration Per Subject

Looking at figures 3.6 to 3.14 it can be seen that Logistic Regression is the most consistent classifier regarding classification confidence. Table 3.1.2.1 shows the optimal configuration and algorithm per subject. Given this results, Logistic Regression is the algorithm selected to continue with further analysis.

Optimal Algorithm And Configuration Per Subject			
Subject	Algorithm	Configuration Number	Score
1	Logistic Regression	5	0.594
2	Logistic Regression	6	0.672
3	Logistic Regression	4	0.594
4	Multi-Layer Perceptron	1	0.627
5	Random Forest Classifier	1	0.594
6	Logistic Regression	2	0.631
7	Logistic Regression	3	0.639
8	Logistic Regression	6	0.626

3.1.3 Classification Results

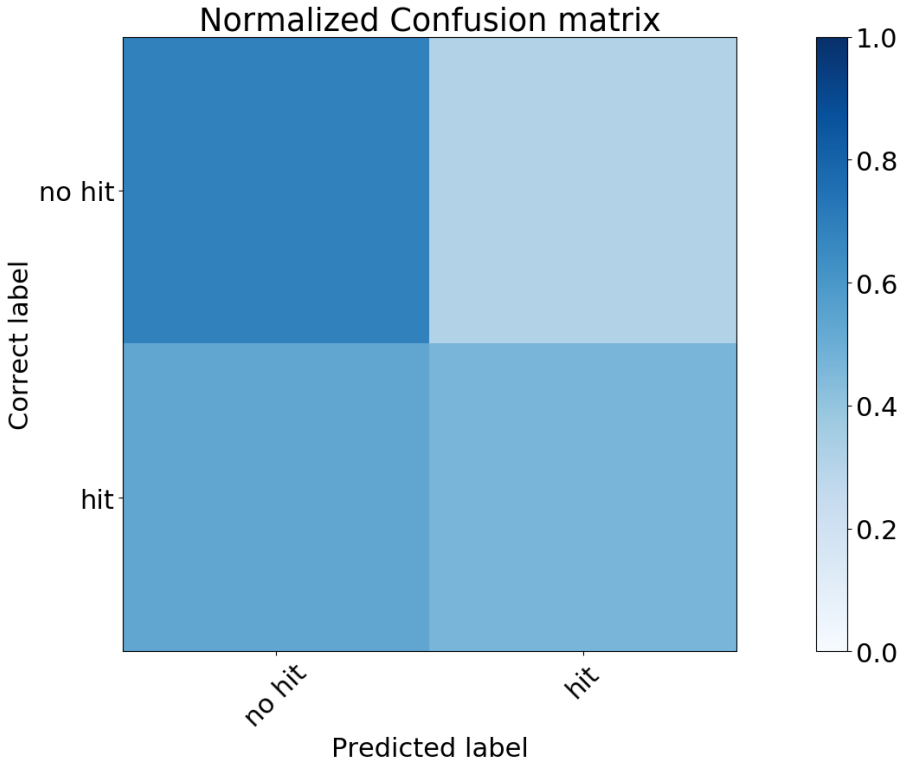
Figure 3.15 to Figure 3.22 show the Confusion Matrix and ROC curve for each test subject and 3.23 shows them for a classifier trained with a noisy signal.

Looking at the confusion matrix, the percentage of samples classified as a hit that are in fact not a hit (false positive) is consistently low for every subject. In most cases Type I error tends to be lower than Type II error for all subjects.

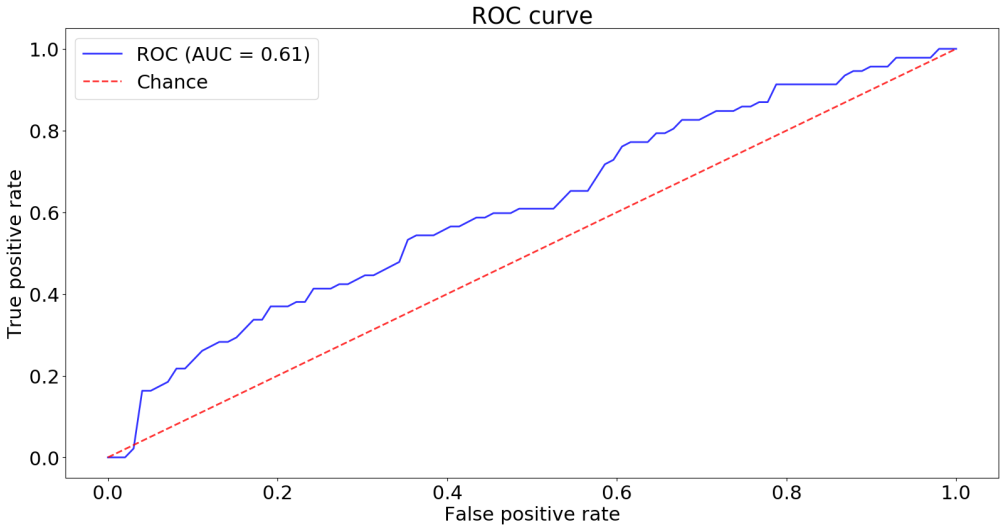
It can be seen that for most subjects the ROC curve is above the identity function, meaning that the classifier performs well enough when classifying the subject's data.

The ROC curves corresponding to subject 5 and 6 are very similar to the identity function, meaning that the rewards originated from those classifiers are expected to have a less meaningful impact when training a Q-Table.

Lastly, it should be noted that when classifying a noise signal, the confusion matrix shows a higher percentage of Type I error compared to the subject's classification.

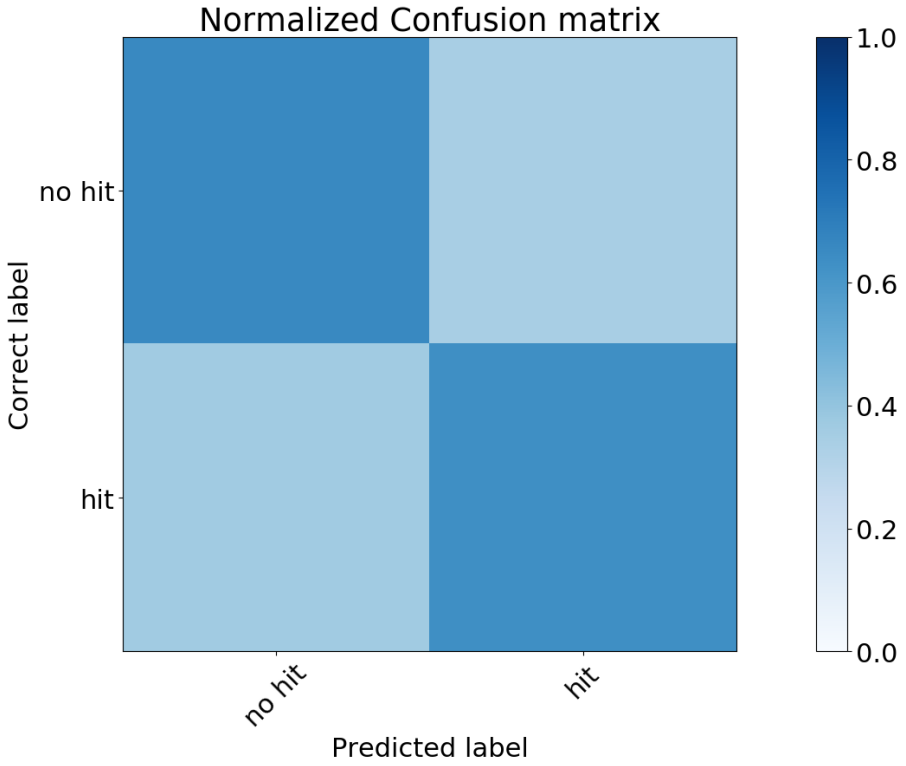


(a) Confusion Matrix: Subject 1. A hit represents the player moving away from the goal

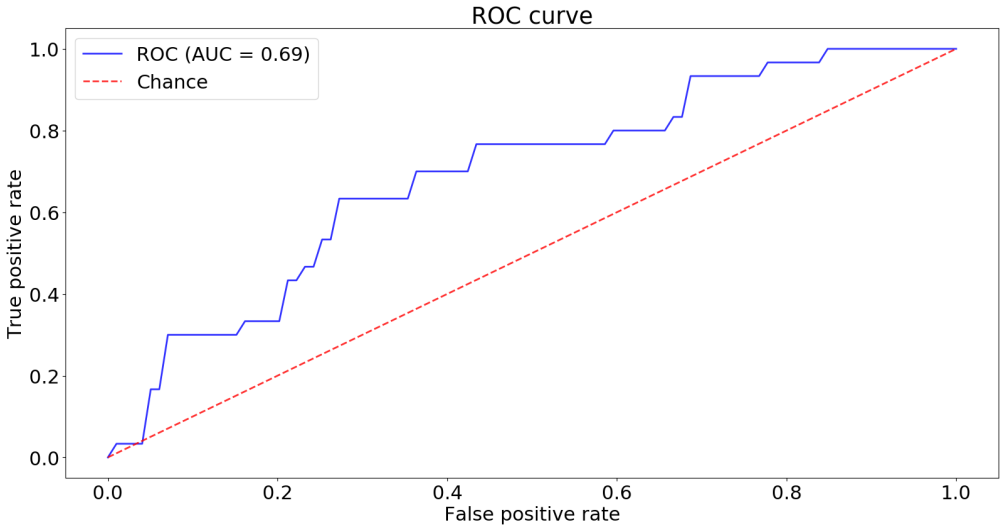


(b) ROC: Subject 1

Figure 3.15: Classification results: Subject 1

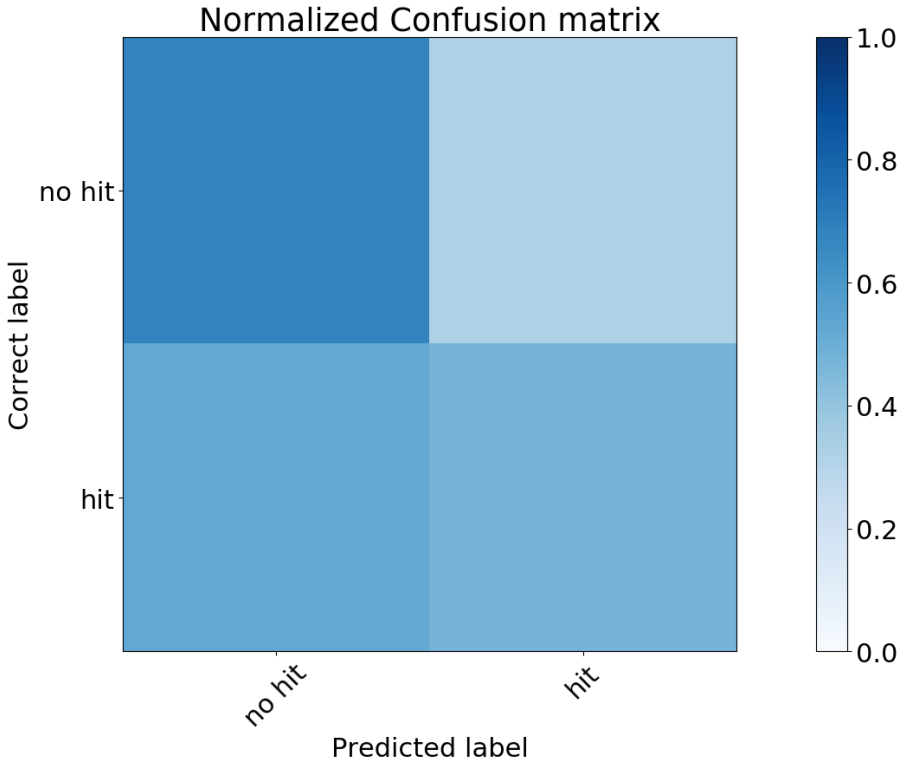


(a) Confusion Matrix: Subject 2. A hit represents the player moving away from the goal

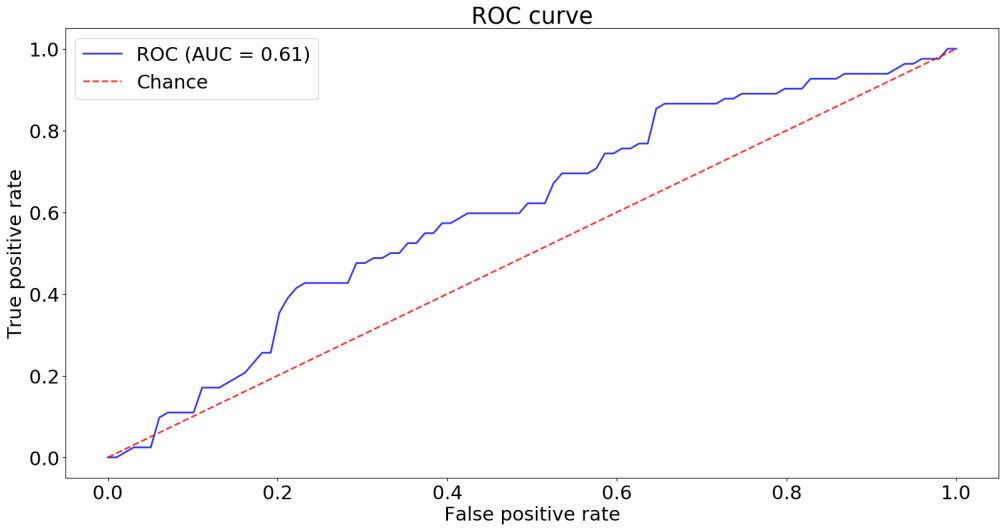


(b) ROC: Subject 2

Figure 3.16: Classification results: Subject 2

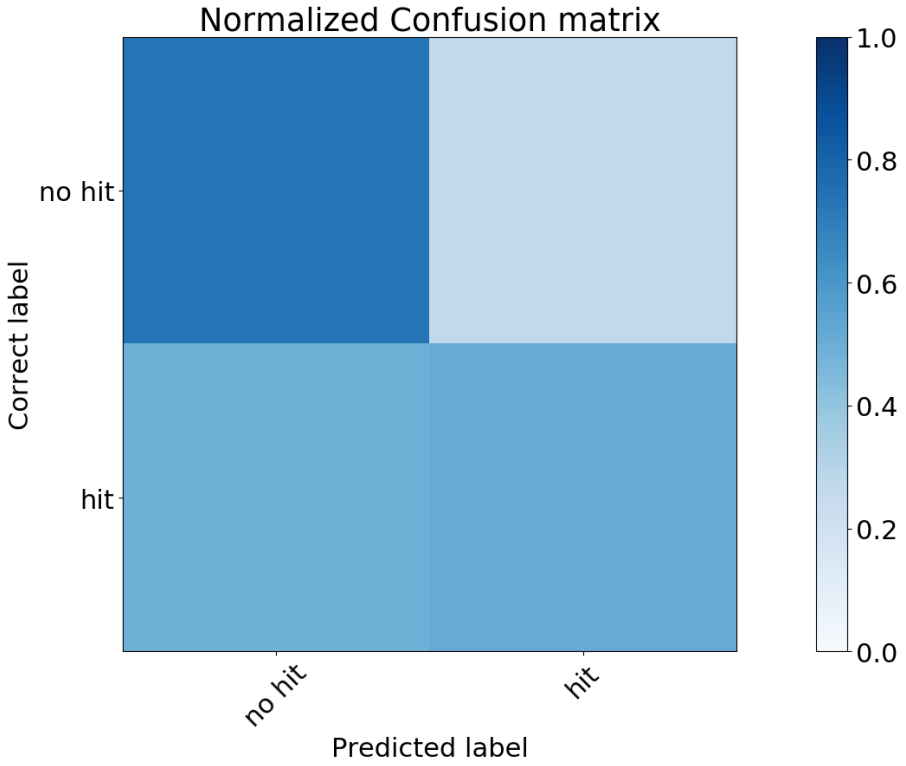


(a) Confusion Matrix: Subject 3. A hit represents the player moving away from the goal

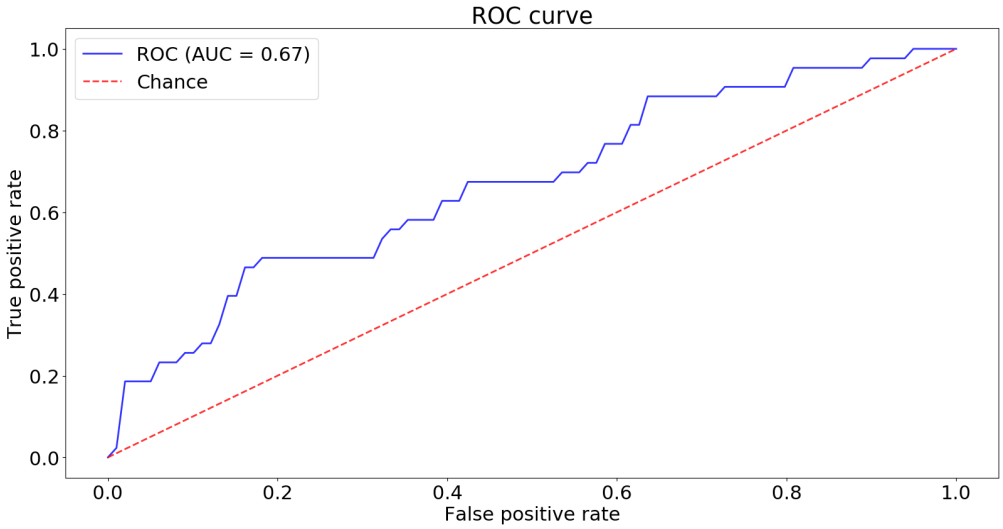


(b) ROC: Subject 3

Figure 3.17: Classification results: Subject 3

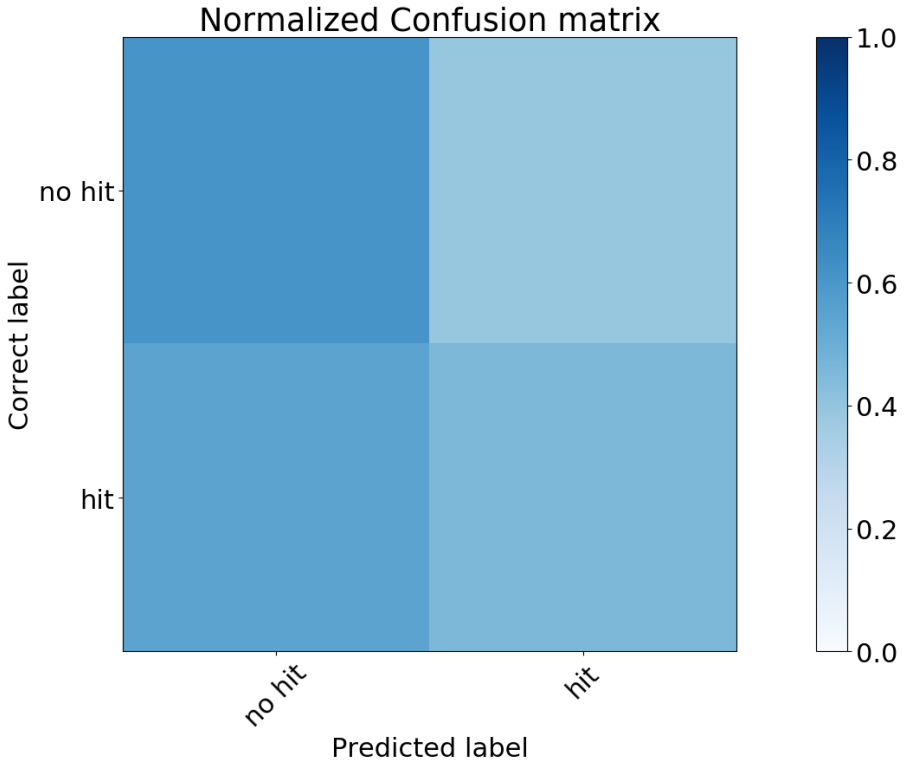


(a) Confusion Matrix: Subject 4. A hit represents the player moving away from the goal

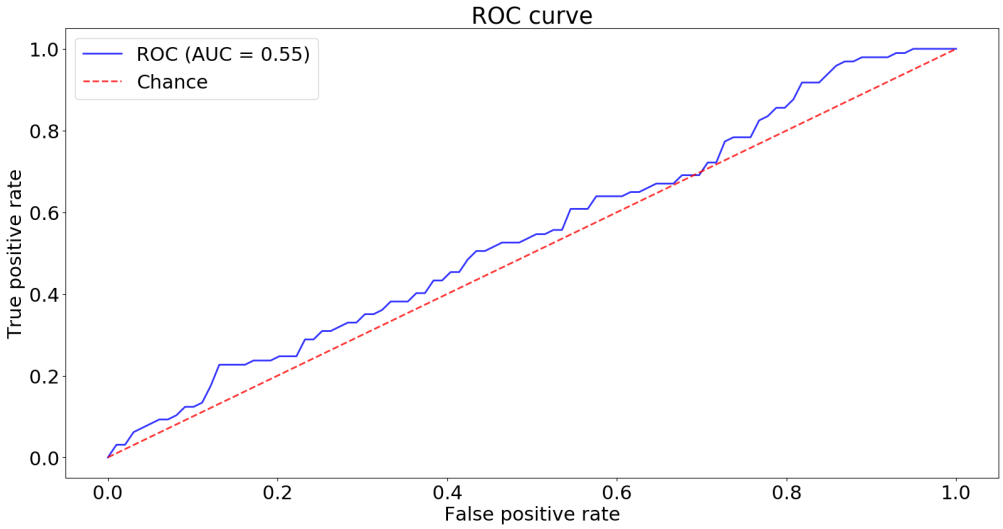


(b) ROC: Subject 4

Figure 3.18: Classification results: Subject 4

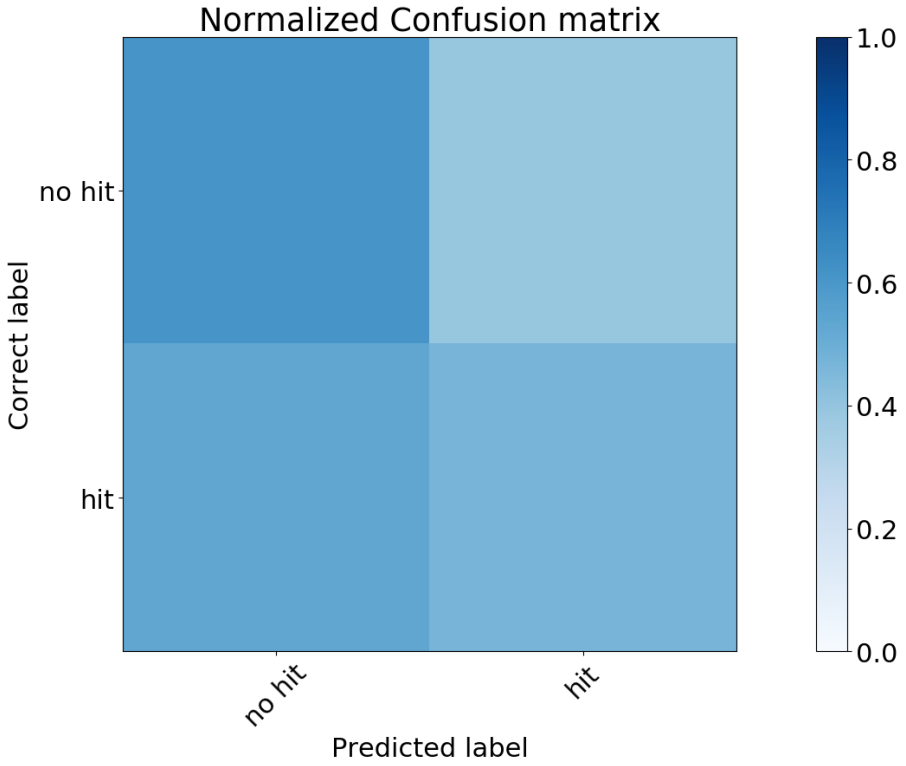


(a) Confusion Matrix: Subject 5. A hit represents the player moving away from the goal

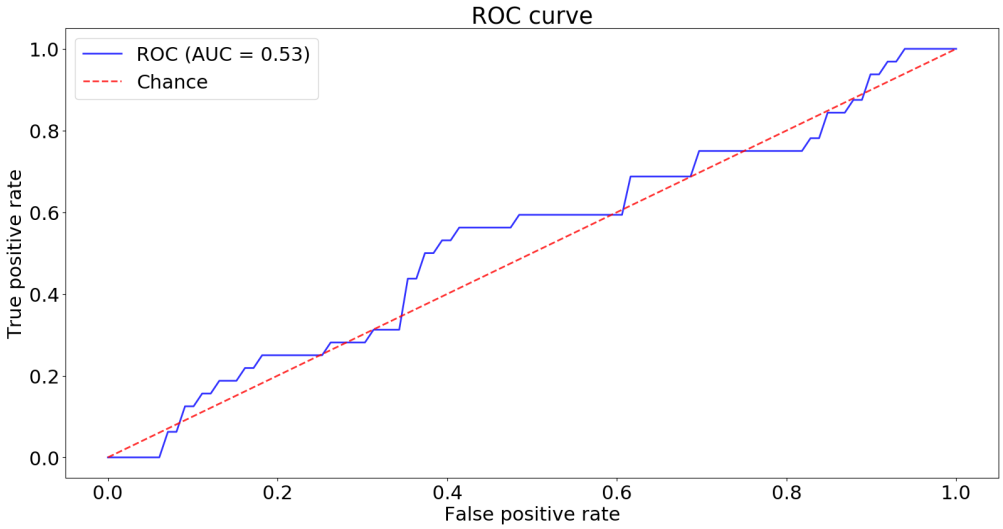


(b) ROC: Subject 5

Figure 3.19: Classification results: Subject 5

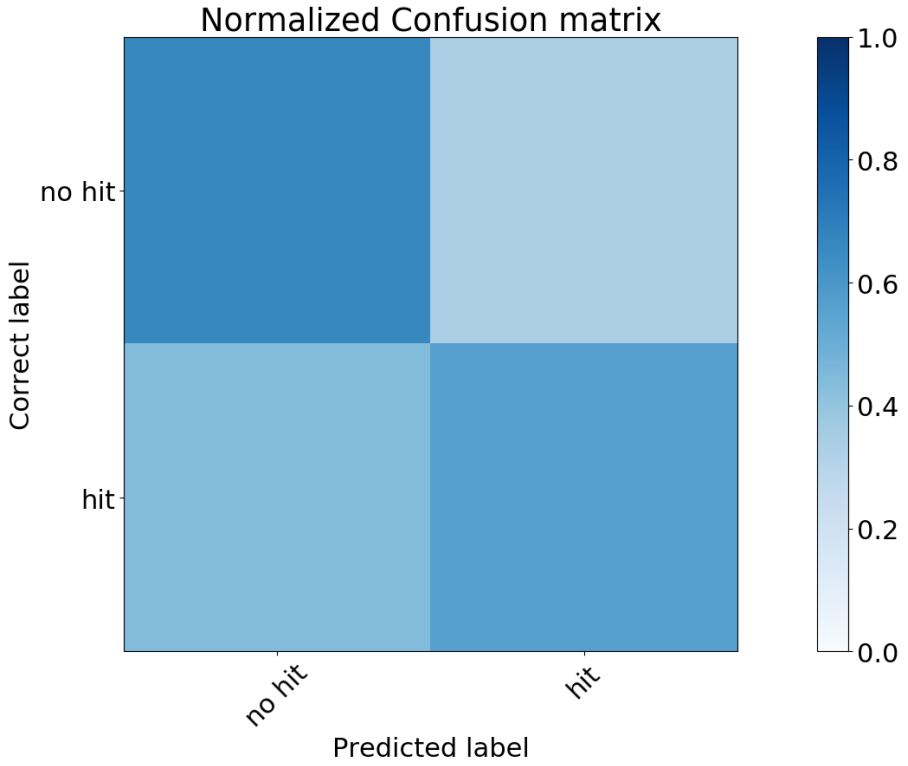


(a) Confusion Matrix: Subject 6. A hit represents the player moving away from the goal

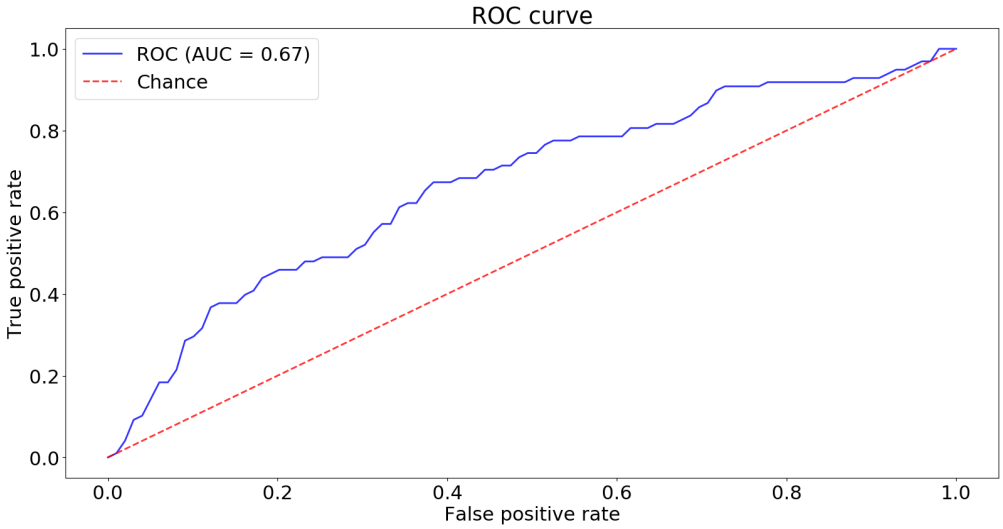


(b) ROC: Subject 6

Figure 3.20: Classification results: Subject 6

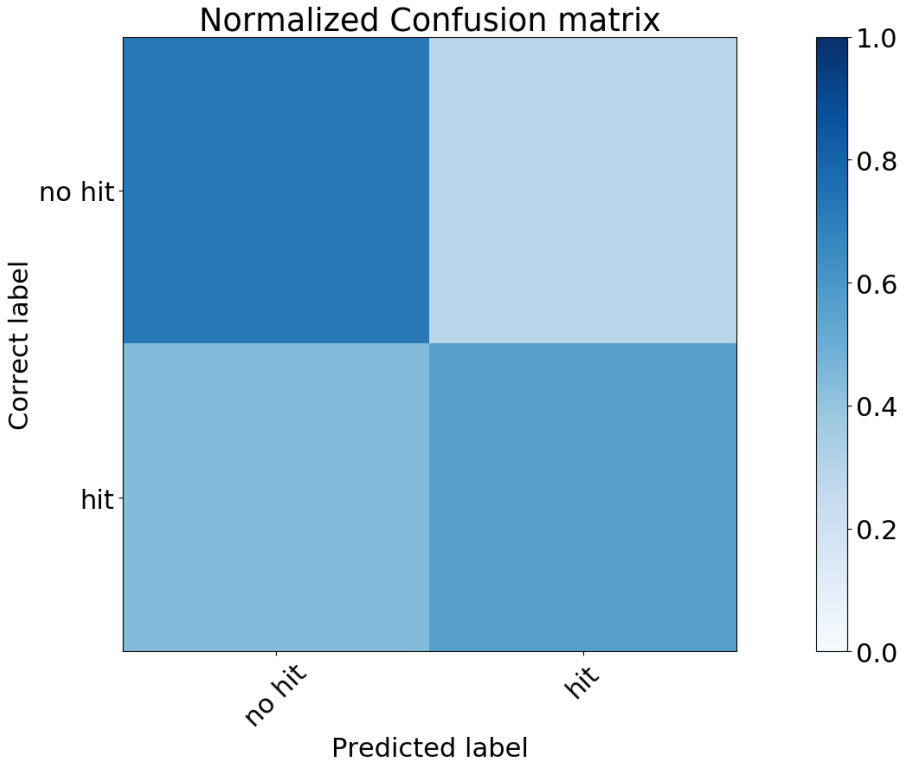


(a) Confusion Matrix: Subject 7. A hit represents the player moving away from the goal

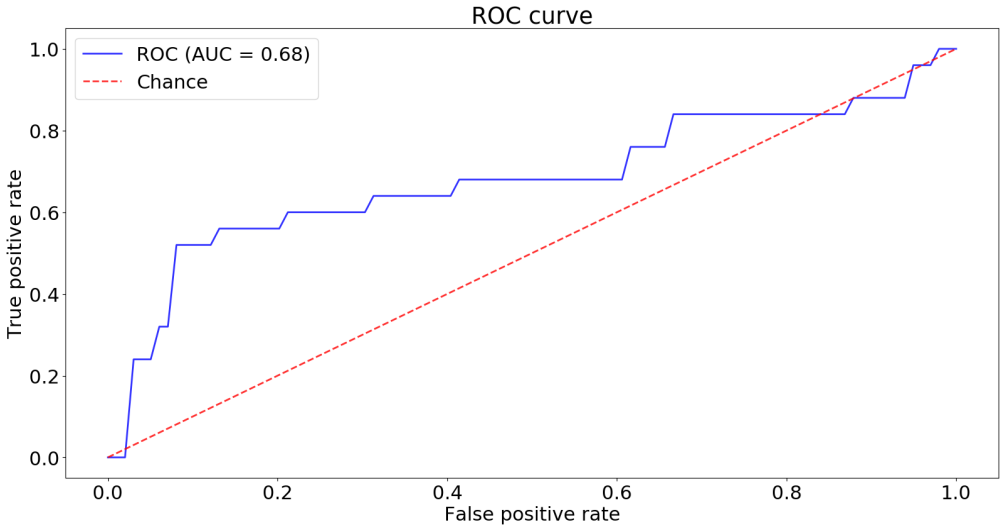


(b) ROC: Subject 7

Figure 3.21: Classification results: Subject 7

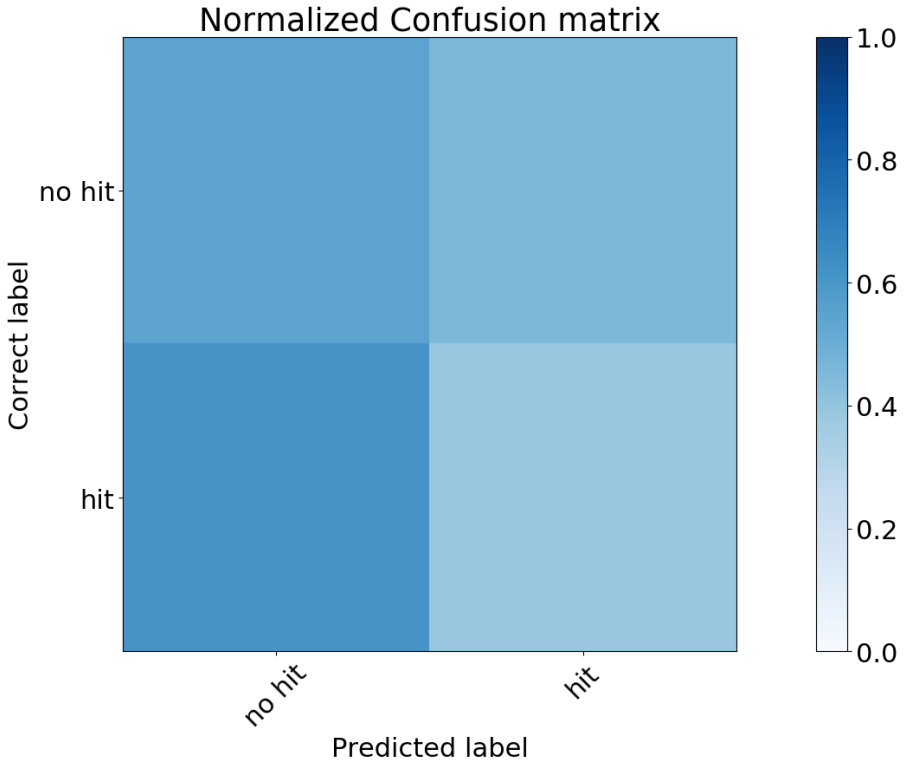


(a) Confusion Matrix: Subject 8. A hit represents the player moving away from the goal

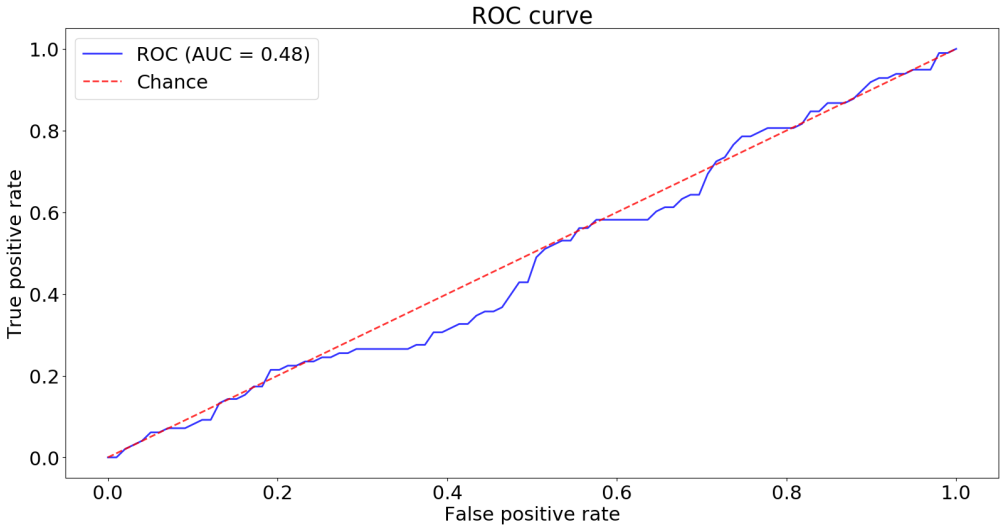


(b) ROC: Subject 8

Figure 3.22: Classification results: Subject 8



(a) Confusion Matrix: Noise. A hit represents the player moving away from the goal



(b) ROC: Noise

Figure 3.23: Classification results: Noise

3.2 Reinforcement Learning

3.2.1 Average steps to goal

Figure 3.24 shows the average amount of steps it takes to reach the goal as the Q-Table is progressively trained using the reward information obtained for each subject. Each point corresponds to the average amount of steps it takes for the agent to reach the goal for a specific Q-Table in 200 iterations. The first point represents the amount of steps it takes to reach the goal for a Q-Table that hasn't been trained at all, where movements are decided randomly. The next point corresponds to the amount of steps it takes to reach the goal using a Q-Table trained with one experience, and so on.

The results show that as the Q-Table is progressively trained the average amount of steps decreases, meaning that the agent learns. However, the rate at which it learns varies per subject, certain subjects have more effective experiments than others, for example results for subject 1 (fig 3.24a) show faster learning than those of subject 8 (fig 3.24h).

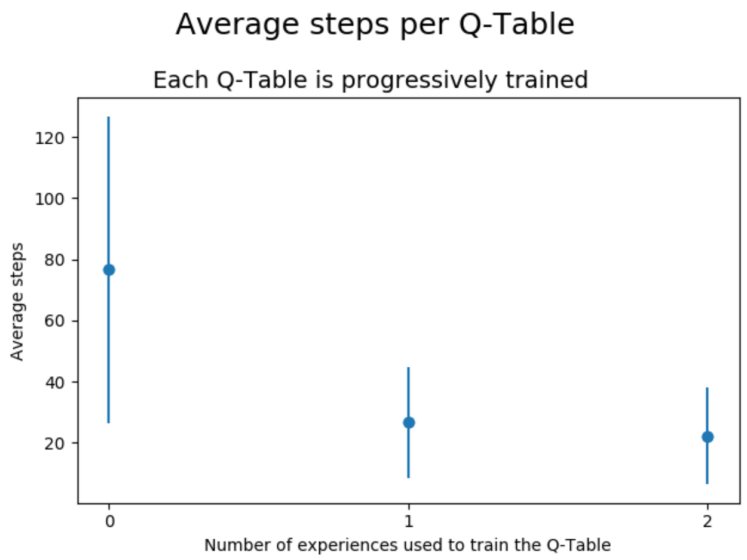
In the case for subject 5 and 6, the reward information obtained from the brainwaves is not enough to train the agent effectively. Figures 3.24e and 3.24f show no apparent learning, as the amount of steps to reach the goal doesn't decrease when trained. Both subjects have less recorded data from the sessions in comparison to the rest of the subjects. In particular, subject 5 has less recorded experiences, so it is respective graphic shows the amount of steps for an empty Q-Table and for a Q-Table trained with only one experience. These results are consistent with fig. 3.19b and fig. 3.20b, which show that the signal classification for these subjects hasn't been particularly successful. The average steps result for these subject are similar to that of fig. 3.23b which is constructed from noise, so the generated Q-Table would be similar to one generated with noise.

Figure 3.25 shows how the amount of steps evolves when training a Q-Table with one experience of each subject progressively, excluding subject 5 & 6 which showed poor results. Using experiments from different subjects the agent still improves its performance, showing that the reinforcement learning algorithm is independent from the subjects. Considering that the agent is trained from a file containing rewards, the only thing that is relevant when training is that this file is correctly constituted. If the signal classification is correct

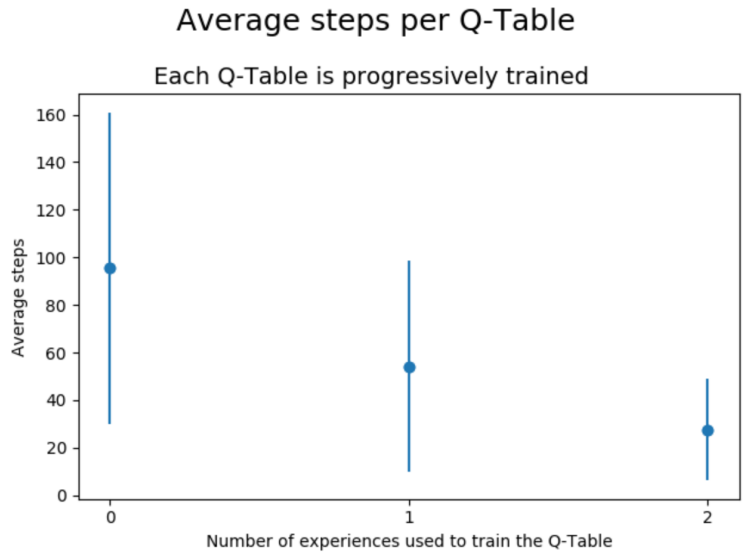
then when training the Q-Table different subjects can be used.

Figure 3.26 shows the average amount of steps for Q-Tables that are progressively trained with noise. It shows that in this case the agent doesn't learn, since the amount of steps doesn't decrease as the Q-Table is trained, whereas when using the Q-Tables that are trained with the subjects experiences the amount of steps does decrease. This shows that is very likely that in the other cases the agent learns in fact by the rewards produced by reading the signals from the person.

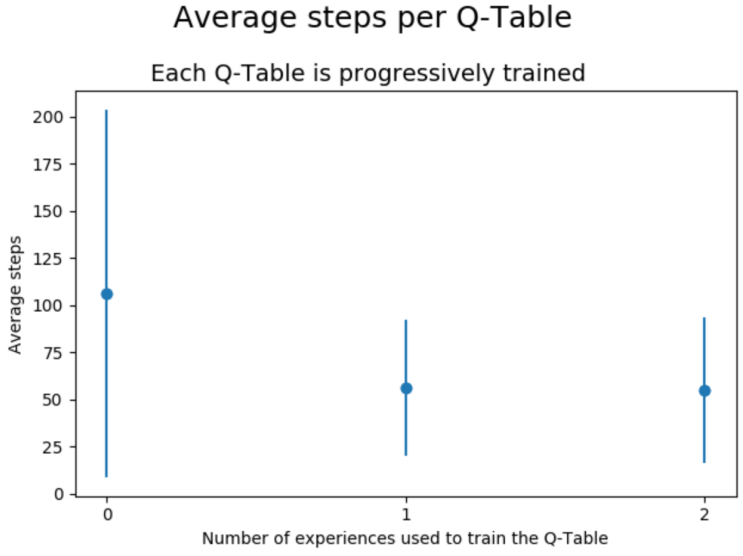
In order to test if the information obtained from brainwaves for a subject can be used to classify signals from other subjects, which is called transfer learning (Jayaram, 2015), game data collected from the brainwave sessions of certain subjects is classified with classifiers trained from other subjects. The results are shown on figure 3.27. In figure 3.27b the results correspond to the experiments from subject 1 classified using a classifier trained exclusively with data from subject 3 (two of the subjects that show the best results), where the next one shows the results of the subject 1 experiments classified with the classifier of subject 3. Both graphics show similar results from those of figure 3.26, where noise is used. In both cases, the amount of steps for the agent to reach the goal does not decrease from those of the initial empty Q-Table, which means that the agent is unable to learn with data classified by a classifier corresponding to other subject. This means that the experiment is not suited for transfer learning.



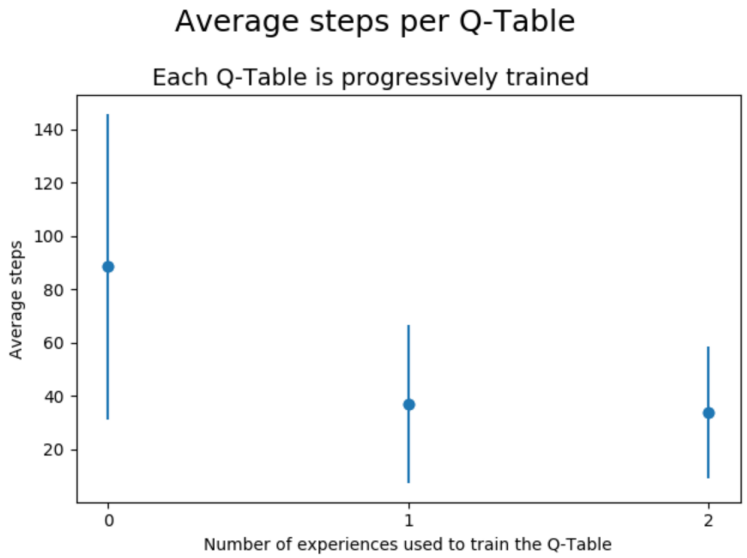
(a) Average steps using Q-table trained with the results of subject 1



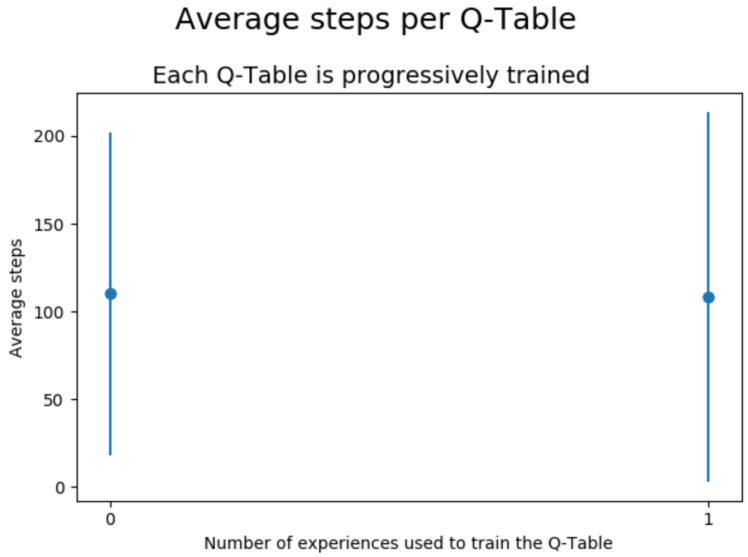
(b) Average steps using Q-table trained with the results of subject 2



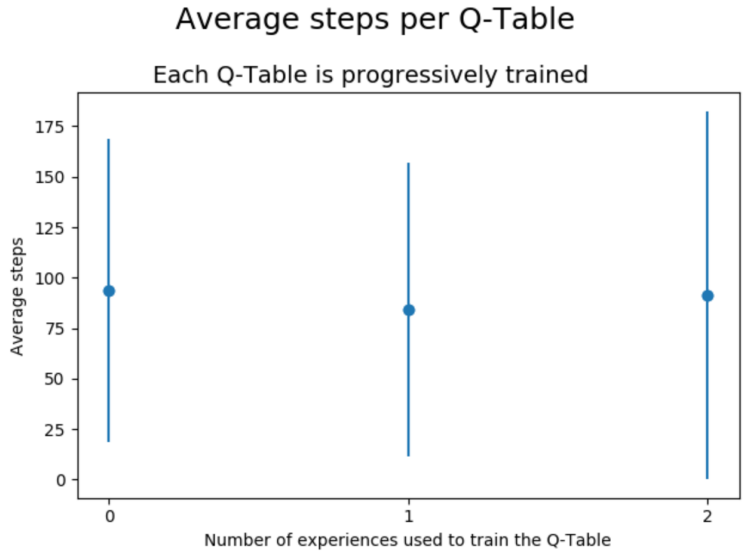
(c) Average steps using Q-table trained with the results of subject 3



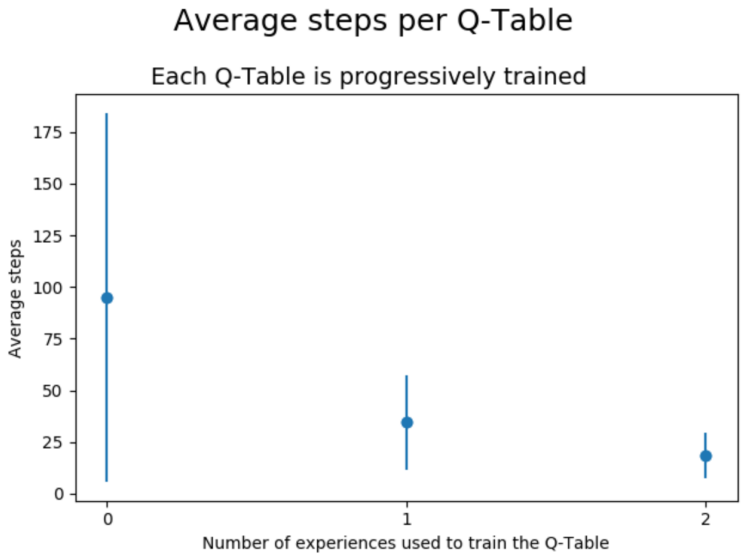
(d) Average steps using Q-table trained with the results of subject 4



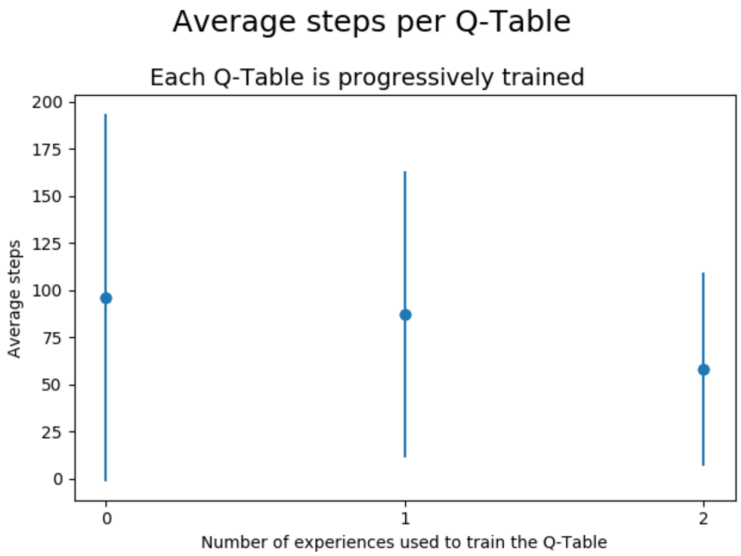
(e) Average steps using Q-table trained with the results of subject 5



(f) Average steps using Q-table trained with the results of subject 6



(g) Average steps using Q-table trained with the results of subject 7



(h) Average steps using Q-table trained with the results of subject 8

Figure 3.24: Average steps for each subject

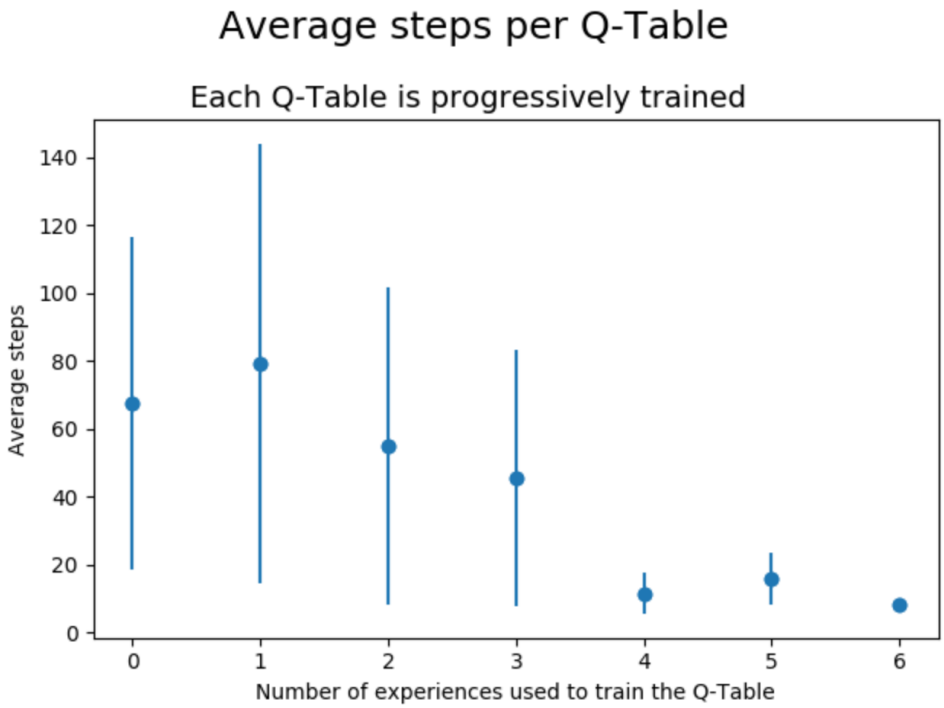


Figure 3.25: Average steps using Q-Table trained with one experience per subject.

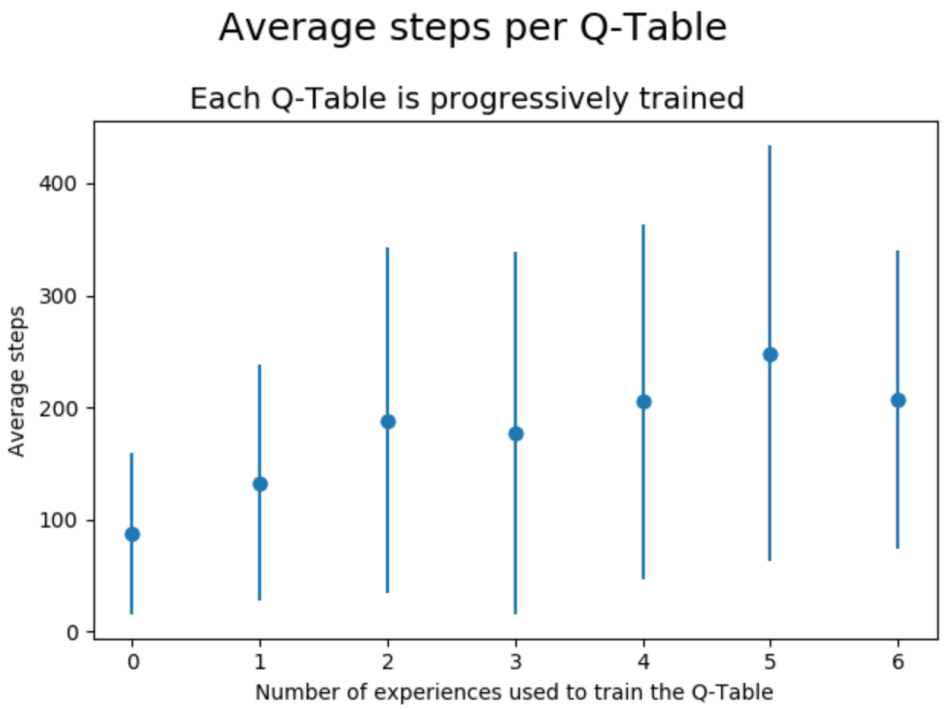
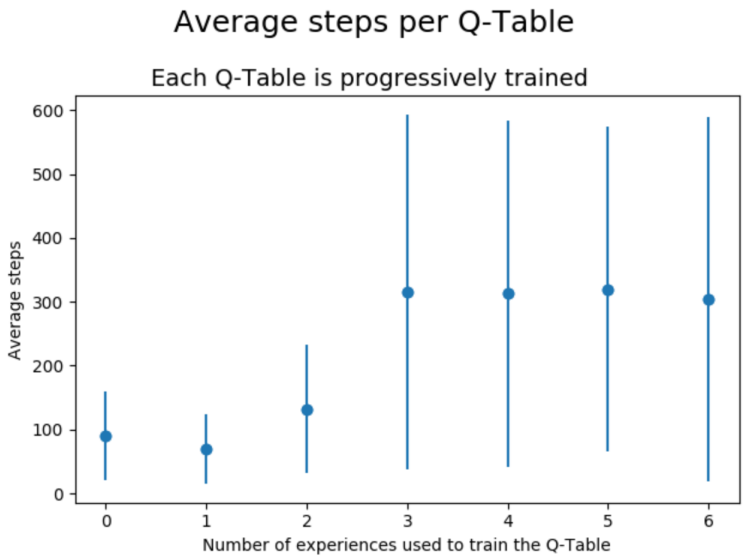
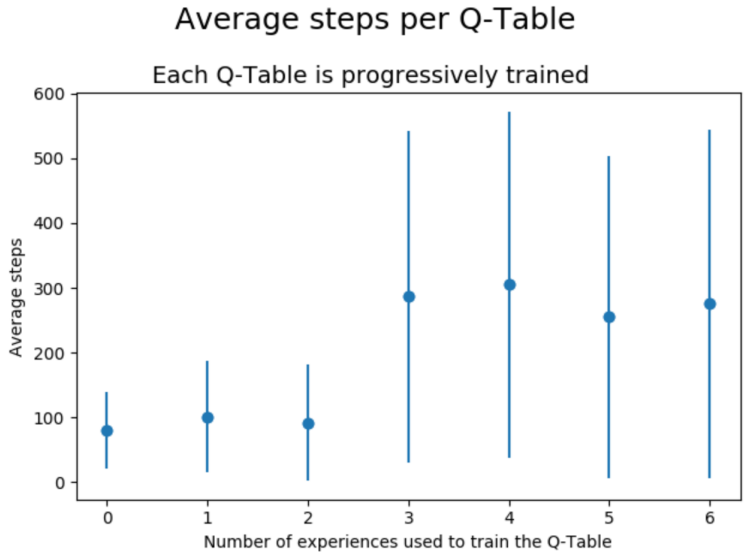


Figure 3.26: Average steps using Q-Table trained with noise.



(a) Average steps using Q-table trained with experiences from subject 8 classified with a classifier trained with data from subject 6



(b) Average steps using Q-table trained with experiences from subject 1 classified with a classifier trained with data from subject 3

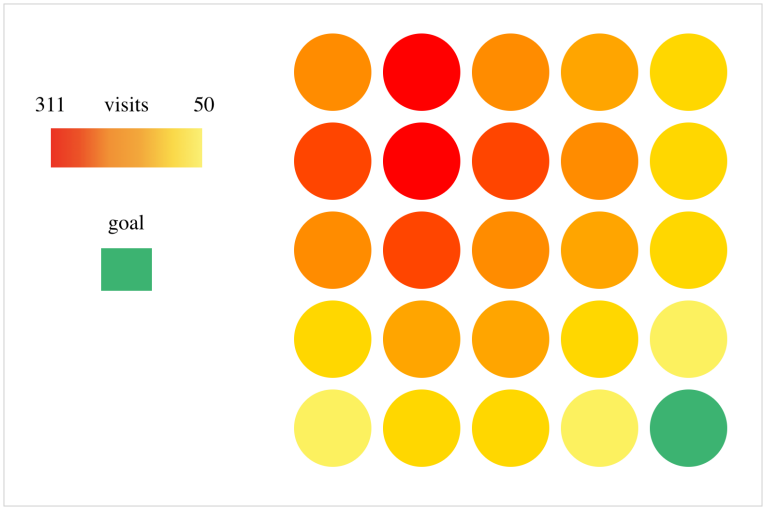
Figure 3.27: Average steps using Q-table trained with experiences from one subject, but classified with a classifier trained with data from other subject

3.2.2 Heat maps of Learned Policies

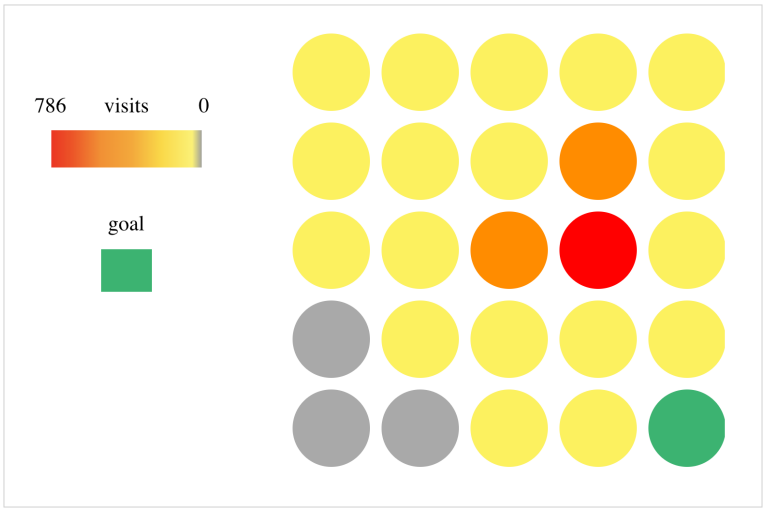
Figure 3.28 shows three heat maps, each one corresponding to 50 cumulative experiences with a specific Q-table. The state of the player has higher frequency on red cells, and lower on yellow ones. The first heat map, 3.28a, shows the results using an empty Q-table, hence, the movement of the player is always random. An oscillation with centre on row 2 and column 2 can be seen, where frequency lowers as cells are further from the oscillation centre. The average amount of actions taken to reach the goal is 78.56. The second map, 3.28b, shows the results of using a Q-table trained with the results from subject 3. The map shows that the frequency values on each cell are more even, than those in the previous map. However there is a peak on row 3 column 4, which could be caused by a classification error or by a tendency to explore certain states that were not explored during the experience. It can also be seen that the states that correspond to the lower-left section of the matrix are not explored, as the Q-table seems to consider moving to those states as a negative action. The average amount of actions taken to reach the goal is 43.08, which shows a substantial improvement.

Figure 3.28c shows the results of using a Q-table trained with the results of both subjects 2 and 3. The average amount of actions taken to reach the goal is 8.32, which is almost the optimal, 8. In this map a path from the starting point to the goal can be seen, which shows a tendency to learn one specific path. This tendency can only be explained by classification errors that lead to penalizing certain paths more than others, as a perfect classification would lead to an equality on the probabilities between choosing two actions that bring the player closer to the goal. The yellow cells can be explained by the probability of choosing a random action, which makes always the process non-deterministic, thus opening a possibility to take an alternative path.

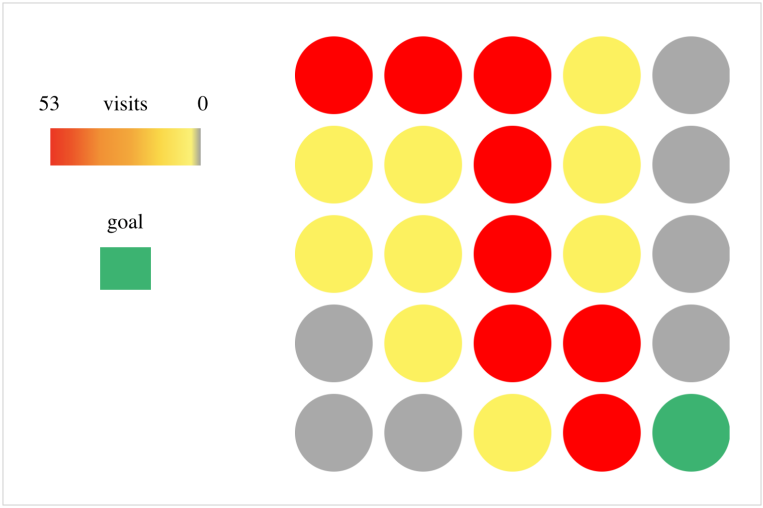
The last heat map, fig. 3.28d, shows the results of using an optimal Q-table. This means that the Q-table always chooses the best action to take for a given state. The results contrasts with those of fig.3.28c, as it does not show a dominant path, instead, they appear to be more evenly distributed. The cells closer to the starting point and goal are more common than the further ones (cells on the up-right and down-left), as those are included in more possible optimal paths.



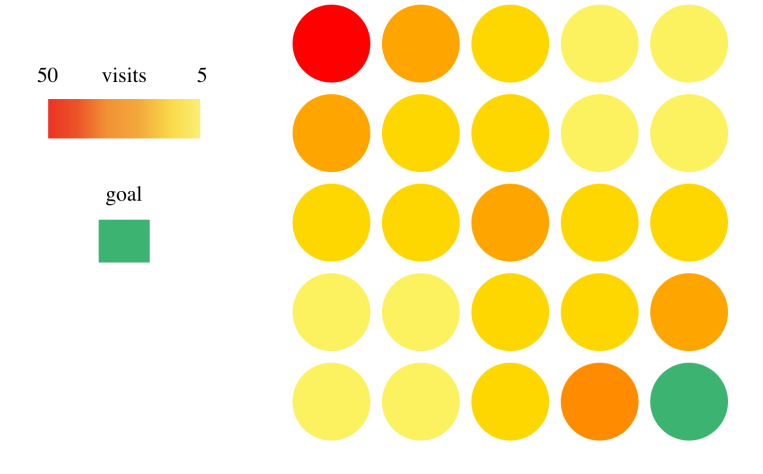
(a) Empty Q-table



(b) Trained Q-table 1



(c) Trained Q-table 2



(d) Optimal Q-table

Figure 3.28: Heat maps

4 Conclusion

The present research work set out to validate if ErrP signals could be used to train an agent using reinforcement learning. The collected data show that ErrP signals can in fact be classified and used to train an agent effectively.

When classifying, the better performing classifier is Logistic Regression. One important aspect of the classification results is the low percentage of false positives, meaning that it is not common that the agent learns that an action is wrong when in fact it is an action that takes it closer to the goal. On the other hand, the percentage of false negatives is generally higher, but this is not a serious issue since missing out on learning that an action is wrong does not lower the performance of the agent, but only means it will take more experiences to learn a correct path.

Once ErrP signals are identified they can be used to train a reinforcement learning algorithm. However, this can only be achieved by generating a specific classifier for each subject and using it for giving rewards for the corresponding subjects. Results show that training a classifier with data of one subject, but using it to classify the events of experiences of another subject does not lead to an improvement on the performance of the agent, concluding that the experiment is not suited for transfer learning. Despite that, the rewards generated from different subjects can be used to train the same Q-Table to improve its performance.

Brainwave sessions have a low amount of experiences in order to reduce fatigue from the subjects. However data suggests that longer sessions are required in order to reach better classification scores, since more data is available in order to train the classifier. It can be seen that subjects with the largest amounts of data have the best classification. This can also be achieved designing a bigger game system that generates more samples with every session.

Finally, this research verifies that brain signals can be used as an interface between human and computer enabling the control of the system without explicit input from the user.

5 Future Work

There are potential areas that could improve the presented methodology and other which could be an interesting complement to what has been achieved. These are presented as follows:

1. Online learning
2. Increase The Number Of EEG Channels
3. Improve Classification Score
4. Explore More Complex Game Models
5. Try Other Reinforcement Learning Algorithms

5.1 Online Learning

In the current implementation the experiment has to end in order to use it to train a Q-Table. A more useful implementation should do the training at the same time as the experiment takes place.

5.2 Increase The Number Of EEG Channels

The version of the used EEG cap has 8 channels. For the used agent the information that could be gathered sufficed, but it would be interesting to use other additional channels to see if the classification score that is obtained can be increased, as more valuable data could be obtained from them. There are up to 64 channels in the same type of EEG cap.

5.3 Improve Classification Score

The classification methods used sufficed for detecting the ErrP in the brainwaves collected, but more complex classification methods could be used in order to increasing the classification score. This way, less false positives could affect the training of the agent.

5.4 Explore More Complex Game Models

The reason why the designed agent was chosen was that the subject that was examining the agent had to be able to easily distinguish whenever a mistake was being made. If the agent had been playing a game of chess for example, then the subjects should have been proficient playing the game in order to identify whenever the agent didn't make the right move.

5.4.1 Non-Deterministic Environment

It would be interesting to experiment within a more complex environment, such as a non-deterministic one. The possibilities given in this kind of scenarios is richer as the decision making is much less obvious. Some ideas considered include making the goal move, which would transform the experiment much more into a chase rather than a goal, and making a competitive environment such as a chess with simpler rules.

5.4.2 Robotic Agent

Initially the plan was to use a robotic agent, it could either be a robotic hand playing some sort of game or a more simpler modification to the existing agent using a robot, that would to have to go through a given path, but would choose it's direction at random. Because of complications that arose during the implementation this couldn't be achieved but it would be an interesting future work.

5.5 Try Other Reinforcement Learning Algorithms

For the given agent the Q-Table reinforcement learning algorithm sufficed, but in case that a more complex agent has been modeled, then it would be relevant to test other types of reinforcement learning algorithms.

6 Acknowledgements

First and foremost we would like to extend our candid appreciation to our tutor, Dr. Rodrigo Ramele, for his assistance, guidance, and expertise during the whole process of this thesis.

This project would not have been possible without the cooperation of ITBA University and several individuals. We would like to extend the deepest appreciation to the robotics and artificial intelligence department, to Dr. Juan Miguel Santos, the head of the laboratory, who lent us all the necessary materials and installations in order to be able to carry out the experiments. We also would like to thank the subjects who volunteered and were key to the development of this project.

Finally, we would like to thank everybody who supported us during these last months and years, our friends and families, without whom this would have never been possible.

6.1 Funding

This was possible thanks to the grant ITBACyT-15 issued by the ITBA University.

References

A. Gramfort, M. Luessi, E. L. D. E. D. S. C. B. R. G. M. J. T. B. L. P. M. H. (2013). *MEG and EEG data analysis with MNE-Python, Frontiers in Neuroscience, Volume 7.*

Andrew Barto, R. S. S. (2018). *Reinforcement Learning: An Introduction.* Cambridge, MA : The MIT Press.

David Silver, Julian Schrittwieser, K. S. I. A. A. H. A. G. T. H. L. B. M. L. A. B. Y. C. T. L. F. H. L. S. G. v. d. D. T. G. D. H. (2017). Mastering the game of go without human knowledge.

Ferrez, P. (2007). Error-related eeg potentials in brain-computer interfaces.

I. Iturrate, L. Montesano, J. M. (2010). Robot reinforcement learning using eeg-based reward signals.

Jayaram, V. (2015). Transfer learning in brain-computer interfaces.

Juan Miguel Santos, C. T. (1999). Exploration tuned reinforcement function.

Kappenman, E. S. and Luck, S. J. (2011). Erp components: The ups and downs of brainwave recordings.

Norvig, P. and Russell, S. (2009). *Artificial Intelligence: A Modern Approach.* Prentice Hall Press Upper Saddle River.

Ramele, R. (2018). Histogram of gradient orientations of eeg signal plots for brain computer interfaces.

Spüler, M. and Niethammer, C. (2015a). Dataset description: Error-related potentials (errps) during continuous feedback.

Spüler, M. and Niethammer, C. (2015b). Error-related potentials during continuous feedback: using eeg to detect errors of different type and severity.

Thorsten O. Zander, Laurens R. Krol, N. P. B. and Gramann, K. (2016). *Neuroadaptive technology enables implicit cursor control based on medial prefrontal cortex activity.*

Wolpaw, J. and Wolpaw, E. W. (2012). *Brain-Computer Interfaces: Principles and Practice.* Oxford University Press.

Yann Renard, Fabien Lotte, G. G. M. C. E. M. V. D. O. B. and ecuyer, A. L. (2010). Openvibe: An open-source software platform to design, test and use brain-computer interfaces in real and virtual environments.

Appendix

A OpenAI Gym FrozenLake Description

The agent controls the movement of a character in a grid world, representing a frozen lake. The agent can only walk over some of the tiles of the grid, while others lead to the agent falling into the ice water. Additionally, the movement direction of the agent is uncertain and only partially depends on the chosen direction, since the agent might slip, causing it to go in an undesired direction. The agent is rewarded for finding a path that reaches a goal tile. The episode ends when you reach the goal or fall in a hole. You receive a reward of 1 if you reach the goal, and zero otherwise. The grid can be seen in the table A.1, where S represents a starting point, which is a safe spot, F represents a frozen surface which is also safe, H represents a hole where if you land on you lose and G represents the goal, which is the point where you ultimately want to reach.

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

Table A.1: Frozen Lake Grid Representation

B Algorithm Selection Configuration

Different parameters are tested in order to choose the best configuration for each algorithm for each experiment subject. Some parameters are kept constant per algorithm, these are specified, and the ones that aren't are described in each different configuration.

B.1 Logistic Regression

- penalty = l2
- max-iter = 300

Logistic Regression		
Configuration Number	C	Solver
1	0.001	lbfgs
2	0.01	saga
3	0.1	saga
4	1	lbfgs
5	1	saga
6	10	lbfgs

B.2 Multi-Layer Peceptron

- solver = lbfgs
- max-iter = 100

Multi-Layer Perceptron	
Configuration Number	α
1	1e-09
2	0.0001
3	0.1

B.3 Random Forest Classifier

- n-estimators = 1000

Random Forest Classifier		
Configuration Number	Max Features	Criterion
1	log2	entropy
2	log2	gini
3	auto	gini

B.4 Support Vector Classifier

Support Vector Classifier	
Configuration Number	C
1	0.001
2	10