UNIVERSITAT DE BARCELONA
Facultat de Matemàtiques i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

Leaky echo state network for brainstates classification

Autor: Clàudia Boixader Garcia

Directors: Dr. Josep Vives i Santa Eulàlia
i Dr. Ignasi Cos Aguilera
Realitzat a: Departament de Matemàtiques i Informàtica
Barcelona, June 13, 2023

Contents

Abstract

The usage of neural networks for classification tasks has gained significant attention in recent years due to its potential in various domains, including medicine, finances or even in social media. In this particular project, based on the previous study realised by Xènia in [11], we will take advantage of these computational models in order to investigate the utility of temporal dynamics in electroencephalogram (EEG) signal classification. Also we aim to evaluate the influence of different classifier methods when classifying those EEG signals.

The research employs Leaky Echo State Networks (ESNs), a type of recurrent neural network, as the main tool for extracting temporal dynamics from EEG signals. As classifiers, two distinct methods will be used to evaluate their impact on the classification task: Ridge Regression and Logistic Regression classifier.

The script starts with a theoretical introduction to neural networks, with a particular focus on Leaky Echo State Networks. Subsequently, a concise overview of the two classification methods employed to construct our network architecture is presented. The final chapter is dedicated to define the aforementioned architecture and revealing the outcomes derived from the application of said network to real EEG data.

Resum

L'ús de xarxes neuronals per a tasques de classificació ha guanyat força protagonisme durant els últims anys degut al seu gran potencial en àmbits com la medicina, les finances o, fins i tot, les xarxes socials. En aquest treball, doncs, farem ús d'aquests models computacionals per tal d'investigar l'utilitat de l'informació temporal, proporcionada per senyals d'electroencefalogrames (EEG), en la classificació d'aquestes. Cal mencionar que el treball és una ampliació del previ estudi realitzat per la Xènia en [11]. Altrament, també es pretén evaluar l'influència de l'ús de diferents classificadors a l'hora de classificar les senyals d'EEG.

Durant el treball utilitzarem les Leaky Echo State Networks (LESNs), un tipus de xarxes neronals, com a eina principal per extreure l'informació temporal de les senyals d'EEG. Com a classificadors, compararem la precisió proporcionada pel classificador de regressió en cresta enfront el de regressió logística.

El treball comença amb una introducció teòrica a les xarxes neuronals, fent especial èmfasi a les Leaky Echo State Networks. A continució es fa una breu pinzellada sobre els classificadors que s'utilitzaran més endevant per a la construcció de la xarxa neuronal, tema que es tractarà a l'últim capítol. En aquest, també es discutiràn els resultats obtinguts d'aplicar la xarxa descrita a dades d'EEG reals.

Acknowledgments

First of all, I would like to thank my family for the emotional support and encouragement given to every choice I made that that led me to become who I am today.

Furthermore, I would also like to thank my tutors Josep Vives and Ignasi Cos for their time and guidance during these last months and also, special thanks to Gabriel Vayá for being my peer throughout the project. Without their support and the valuable discussions held during these months, it would not have been possible to carry it out.

Lastly, I would not want to forget all the people I have met and shared these last years with. Each of them helped with making everything easier and, undoubtedly, much better.

2022 Mathematics Subject Classification. 68T07, 62J07

Chapter 1

Introduction to Neural Networks

The upcoming chapter provides a concise introduction to neural networks, exploring their fundamental concepts and classifying them into main types. We based it, mainly, on [9], pages 3-12 and [10].

1.1 History

A neural network is a type of machine learning model that is inspired by the structure and operation of the human brain. It has a long history with the first artificial neuron described in 1943 by the mathematician Walter Pitts and the neurophysiologist Warren McCulloch. They wrote a paper, *The Logical Calculus of the Ideas Immanent in Nervous Activity*, about how neurons might work, and also developed a simple neural network using electrical circuits.

During next years, its popularity continued to grow reaching 1958, the year when the psychologist Rosenblatt came up with the perceptron, one of the first computational models of a neuron, mostly used for classification or regression problems.

Definition 1.1. *A **perceptron** is defined as follows:*

$$y(x) = f(b + \sum_{i=1}^{n} w_i x_i) \tag{1.1}$$

where b is the threshold (or bias), $x \in \mathbb{R}^n$ is a set of inputs, each of them associated with a weight w_i and $y(x) \in \{0, 1\}$ is the output value. Finally, the function f is defined as:

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{1.2}$$

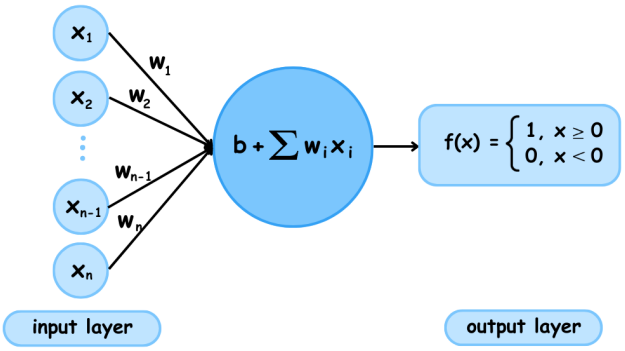


Figure 1.1: Architecture of a perceptron

After an initial period of enthusiasm, progress in the field experienced a difficult phase until the end of the century. However, renewed interest emerged with the celebration of the nowadays annual meeting of *Neural Networks for Computing*, as well as the publication of new and rediscovered concepts, such as the back propagation algorithm.

Nowadays, neural networks has become a very popular term due to its great achievements in many domains such as computer vision, speech recognition or natural language processing. However, all the progress would not have been possible without the appropriated computing power and, for that very reason, I would venture to say that the future of neural networks lies in the development of hardware.

Despite all the success of neural networks, there are still a lot of challenges to face as explainability of models or large datasets processing.

1.2 Basics and notation

As previously mentioned, a neural network is a machine learning model which is composed by one or multiple neurons organized into layers. Therefore, in order to better understand what neural networks are and their functionality, we are going to invest some time into knowing about the fundamental concept of aneuron, before jumping to the next topic.

Definition 1.2. *A **neuron** is nothing more than an improvement of the definition of perceptron we saw previously. We define it, given an n dimension input vector and using the same notations used before as:*

$$y(x) = f(b + \sum_{i=1}^{n} w_i x_i) \tag{1.3}$$

where the output $y(x) \in \mathbb{R}$ and the function f differs from the definition we had before for the perceptron, as it can take more than two possible output values. It is called the ***activation function*** *and it is the one in charge of representing convoluted random functional mappings between input and output.*

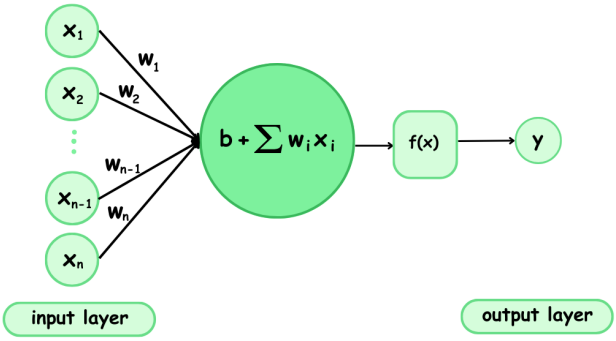


Figure 1.2: Architecture of a neuron

Remark 1.3. We are considering that every component of the input vector is contributing additively to the neuron where it is connected, which is the most widely used rule, but there are other approaches such as the Sigma-Pi-Sigma rule which may also be useful in some specific cases such as for function approximation.

Remark 1.4. It is interesting to see that a neuron without an activation function will work as a linear model, which is just a polynomial of degree one. A very similar thing happens when using linear activation functions, which make the network to behave as a linear regression model.

Regarding the previous remark, when considering only linear activation functions the possible applications of neural networks remain quite limited. Consequently, most neural networks end up using non-linear activation functions. From [10] we know that some of the most widely used are:

- Sigmoid: it transforms the values it gets into values within the range (0,1). This helps the user to understand the output values as they can be interpreted as probabilities.

$$f(x) = \frac{1}{1 + e^{-x}}$$

- Hyperbolic Tangent: the main advantage of using tanh to sigmoid is that its gradient around 0 it is almost four times bigger than the gradient of the

sigmoid function also around 0. This property might be useful if we want to make big learning steps when training the network, i.e, when learning the best weights for the network.

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- ReLU: it is more efficient than other functions because it allows us to do not activate all the neurons at the same time. This is a consequence of the gradient function that, in some cases, its value is zero, which leads the weights and biases to not be updated during the learning processes when training the network.

$$f(x) = max(0, x)$$

Having established an understanding of the nature of a neuron and some of the most used activation functions, we may now proceed to formally define what is a neural network.

Definition 1.5. *A **Neural Network (NN)** consists of one input and one output layer with any or some hidden layers in between. When there is no hidden layers, we call them **single-layer NN**. Otherwise, we call them **multi-layer NN**. Each of those hidden layers contains one or more neurons connected to other neurons, not necessarily from the same layer.*

Depending on how this connections are made, we have different types of neural networks. Actually regarding how neurons connect to each other, we can split them in two big categories: **Feed Forward Neural Networks (FFNN)** and **Recurrent Neural Networks (RNN)**. Let's see in the next section which is its main difference and also what similarities they have.

1.3 Types of neural networks

When talking about neural networks, there is one big feature they all share indifferently of its type. This particular feature is the capability to learn from its environment, which at the same time is the characteristic that distinguishes them from conventional computers. Some different learning algorithms exist such as Back Propagation or the Hebbian learning rule.

The key difference between **Feed Forward Neural Networks (FFNN)** and **Recurrent Neural Networks (RNN)** lies in how neurons from different layers are connected to each other. For FFNN we have neurons from different layers which

connects to neurons from other layers without creating cycles, thus restricting the connection to go forward to the following layers. As neuron activation spreads through the network starting from input layer and reaching the output layer going through the hidden layers, if the input layer is inactive, subsequent layers will also be inactive. We can see in the figure 1.3 an example of the basic architecture of a FFNN with two hidden layers.

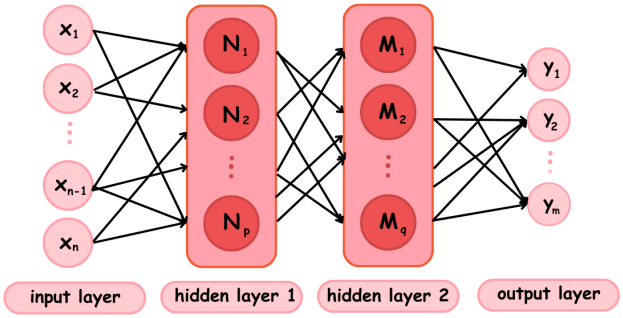


Figure 1.3: Architecture of a FFNN with two hidden layers

Oppositely to FFNN, RNN require to have at least one cycle, either with itself or with neurons from other layers. This feature allows the output of some neurons to affect, directly or not, the input of the same neuron. Mathematically, we can see RNN as dynamical systems while FFNN can be thought of as functions.

A significant advantage of RNN is that they preserve in its internal state a nonlinear transformation of the input history (in case the activation function used is non-linear). In other words, they have a dynamical memory and are able to process temporal information. For that very reason, later on the practical application, we are going to use a RNN as we will be dealing with time series data. In particular, we will use an RNN based on Reservoir computing.

1.4 Reservoir Computing

In the following chapter we used as a reference chapters 2 from [6] and [8]. We also used the results from [5] and based our section Echo State Poperty in [7].

Reservoir computing (RC) is a family of RNN models which achieved excellent performance in time series forecasting and process modelling in the last few years. In machine learning, RC techniques were originally introduced under the name Echo State Networks (ESNs). Those networks were designed to efficiently process and model temporal data. The key characteristic that differentiates them from the

others is that the reservoir layer will not be updated during training. In other words, the weights defining the connections between the neurons in the reservoir layer will remain the same during all the training process. The reservoir layer serves as a dynamic memory thus allowing the network to be able to process temporal context information in the following layer. Therefore, the model can be summarized in three main steps:

- Firstly, a recurrent neural network is created and remains unchanged during training. This layer is called the reservoir. It is passively excited by the input signal and maintains in its state a nonlinear transformation of the input history.
- Secondly, the weights obtained from the reservoir layer are trained, usually by direct methods, in order to map them to the desired output. This layer is called the readout layer and it is, usually, composed by linear units.
- Finally, the output weights acquired in step two are used together with new input data in order to predict new final outputs.

In this project, we will be focusing in one particular Reservoir Computing model, named Echo State Network. This will, actually, be the model used later in the practical part since it has been shown in [5] to have clear superiority over standard RC models in problems requiring effective propagation of input information over multiple time-steps, which is the particular case of the data we will be dealing with.

1.4.1 Echo State Networks

All over this chapter we will be assuming our network to have:

- K input values
- N neurons in the reservoir layer
- L output values
- T the size of each input value

We will be focusing on the Echo State Network (ESN) model and in particular, to the general ESN formulation with leaky integrator neurons.

Definition 1.6. *A **Leaky Echo State Network (LESN)** is modeled with the following ODE:*

$$\dot{x}(n) = -x(n) + f(W^{in}u(n) + Wx(n)) \tag{1.4}$$

where x and u, respectively, denote the state and the driving input for each time step n, W is the recurrent weights matrix, W^{in} is the input weights matrix and f is the activation function defined previously.

From this ODE, let's approximate its solution by discretizing the continuous domain into small time intervals using the forward Euler method. Taking

$$\frac{\Delta x}{\Delta t} = \frac{x(n+1) - x(n)}{\Delta t} \approx \dot{x} \tag{1.5}$$

where Δt is the step size and replacing it in (1.4), we get the following equation:

$$\begin{aligned}
&\frac{x(n+1) - x(n)}{\Delta t} = -x(n) + f(W^{in}u(n) + Wx(n)) \\
\iff\ & x(n+1) - x(n) = \Delta t \cdot (-x(n) + f(W^{in}u(n) + Wx(n))) \\
\iff\ & x(n+1) - x(n) = -\Delta t \cdot x(n) + \Delta t \cdot f(W^{in}u(n) + Wx(n)) \\
\iff\ & x(n+1) = (1 - \Delta t) \cdot x(n) + \Delta t \cdot f(W^{in}u(n) + Wx(n))
\end{aligned}$$

Definition 1.7. *Using the same notation as in the previous definition, the **state transition equation** for each time step n is defined as:*

$$x(n) = F(u(n), x(n-1)) = (1-\alpha)x(n-1) + \alpha f(W^{in}u(n) + Wx(n-1)) \tag{1.6}$$

where $x(n) \in \mathbb{R}^N$, $u(n) \in \mathbb{R}^K$, $W \in M(\mathbb{R}^{N\times N})$, $W^{in} \in M(\mathbb{R}^{N\times K})$ and $\alpha \in (0, 1]$ is the leaking rate.

Remark 1.8. Although the most common activation function is the hyperbolic tangent, other functions can be applied. Even so, in our future work in chapter 3 and from now on, when talking about activation function we will be referring to the hyperbolic tangent.

Also it is worth noticing than the model can be used without the leaking rate if setting $\alpha = 1$ thus turning it into an "ordinary" ESN model.

Definition 1.9. *The (commonly linear) readout layer is defined as:*

$$y(n) = f(W^{out}[1 : x(n)] \tag{1.7}$$

where $W^{out} \in M(\mathbb{R}^{L\times(N+1)})$ being N the number of neurons and L the number of output units. $[a; b]$ corresponds to the concatenation of vectors a and b and f, which is the activation function, may not be used in some cases. The column of ones added to the state input matrix x, refers to the bias of the model.

Remark 1.10. Some variants of the output equation exists. For instance,

$$y(n) = f(W^{out}x(n)) \qquad \text{or} \qquad y(n) = f(W^{out}[1 : u(n) : x(n)]).$$

However, in the later application of the network, we will be using equation (1.7) to compute the predicted output of the network.

1.4.2 Echo State Property

As we just saw, in a LESN, the network dynamics are divided into two main components: the input weights and the reservoir dynamics. The input weights are responsible for transforming the input signals, while the reservoir dynamics capture the temporal dependencies in the data. That's why each state $x(n)$ from the reservoir dynamics can be understood as an "echo" from the input history. It is also worth remembering that the only values that are going to be trained are the ones on the readout layer while the ones from the reservoir remain untrained from their initialization. As a consequence, Echo State Property play a very important role in this type of networks as it holds that the reservoir state should asymptotically depend only on the driving input signal while the influence of initial conditions should progressively vanish over time. Next, we will see a formal definition of such property as well as some sufficient and necessary conditions for the Echo State Property to hold.

Definition 1.11. *Assume a LESN whose global dynamics are ruled by a equation as the one seen in 1.6. The network satisfies the* ***Echo State Property (ESP)*** *if* $\forall x, x' \in \mathbb{R}^N$ *states and* $\forall s_T(u) = (u(1), \ldots, u(T)) \in \mathbb{R}^{K \times T}$ *input,*

$$||\tilde{F}(s_T(u), x) - \tilde{F}(s_T(u), x')|| \longrightarrow 0 \quad when \quad T \to \infty$$

In other words, ESP holds that the distance between the states in which the LESN is driven after being fed by the same input sequence, but starting from different initial conditions, approaches 0 as the length of the input sequence goes to infinity for all every initial condition.

Now, let's go ahead and provide a theorem that will gives us a necessary condition for LESN to have echo states. Before that, though, let's see one previous result regarding the stability of the system that will help us when proving the just mentioned theorem. To do so, it will be useful to start by linearizing our system as it will help to capture the local behavior of the system around a specific operating point. Using Taylor series expansion for $x(n) = F(u(n), x(n-1))$ around state $x_0 \in \mathbb{R}^N$ we get:

$$x(n) \approx F(u(n), x_0) + J_{F,x}(u(n), x_0)[x(n-1) - x_0] \tag{1.8}$$

where $J_{F,x}(u(n), x_0)$ denotes the Jacobian matrix of $x(n)$ evaluated in x_0.

Lemma 1.12. *Consider the linearized system in equation (1.8) and assume null input sequence as an admissible input. Then, a necessary condition for the stability of the system dynamics around zero state is given by*

$$\rho(J_{F,x}(\bar{0_u}, \bar{0})) < 1 \tag{1.9}$$

where $\bar{0_u}$ and $\bar{0}$ are the null vectors of dimensions K and N, respectively.

Proof. Assuming a constant zero input $\bar{0_u}$ for the equation (1.8), the first-order approximation of the LESN dynamics around the zero state $\bar{0} \in \mathbf{R}^N$ is:

$$x(n) = J_{F,x}(\bar{0_u}, \bar{0})x(n-1) \tag{1.10}$$

from which it is easy to see that the zero state is a fixed point of the linearized system. Moreover, the stability of the zero state as a fixed point of equation (1.10) determines the asymptotic behavior of the trajectories starting from a state in a small neighborhood of 0. We also know that the spectral radius determine the rate of growth or decay of perturbations from the equilibrium states, in our case, the zero state. Therefore, if $\rho(J_{F,x}(\bar{0_u}, \bar{0})) \geq 1$ then we cannot guarantee that the zero state is stable. Hence, a necessary condition for the stability of the linearized system around the zero state is given by $\rho(J_{F,x}(\bar{0_u}, \bar{0})) < 1$. □

Theorem 1.13. *Consider a LESN whose dynamics are defined by equation (1.6) where the activation function is $f(x) = tanh(x)$, and assume a null sequence as admissible input for the system. Then, a necessary condition for the ESP of the LESN dynamics around the zero state is given by:*

$$\rho((1-\alpha)Id + \alpha W) < 1 \tag{1.11}$$

where α is the leakying rate.

Proof. Using the previous lemma, we just need to prove that

$$J_{F,x}(\bar{0_u}, \bar{0}) = (1-\alpha)Id + \alpha W. \tag{1.12}$$

Therefore,

$$\begin{aligned} J_{F,x}(\bar{0_u}, \bar{0}) &= \frac{\partial F(u(n), x(n-1))}{\partial x(n-1)}(u(n), x_0) \\ &= \frac{\partial[(1-\alpha)x(n-1) + \alpha f(W^{in}u(n) + Wx(n-1))]}{\partial x(n-1)} \\ &= (1+\alpha)Id + \alpha\frac{\partial f(W^{in}u(n) + Wx(n-1))}{\partial x(n-1)}W \end{aligned}$$

Considering zero input and state, and being $1 - tanh(x)^2$ the derivative of $tanh(x)$, we finally reach the (1.12) expected expression. □

To conclude this section, we will see one last theorem which gives us a sufficient condition for Echo States Property, which depends on a Lipschitz property of the reservoir weights matrix.

Theorem 1.14. *Assume we have a LESN whose dynamics are defined by equation (1.6) and where the activation function is $f(x) = tanh(x)$. Let the reservoir weights matrix W satisfy $\sigma(W) = ||W||_2 < 1$ where $\sigma(W)$ denoted the largest singular value of W. Then,*

$$||\tilde{F}(s_T(u), x) - \tilde{F}(s_T(u), x')||_2 < C||x - x'||_2$$

for all inputs $s_T(u) = (u(1), \dots, u(T)) \in \mathbb{R}^{K \times T}$, for all states $x, x' \in [-1, 1]^N$ and for some $C < 1$. This implies ESP for all inputs $s_T(u)$ and for all states $x, x' \in [-1, 1]^N$.

Proof. We need to prove that $\exists C < 1$ such that

$$||\tilde{F}(s_T(u), x) - \tilde{F}(s_T(u), x')||_2 < C||x - x'||_2.$$

From (1.6) we have:

$$\begin{aligned}
||\tilde{F}(s_T(u), x) - \tilde{F}(s_T(u), x')||_2 &= ||(1-\alpha)x(n-1) + \alpha f(W^{in}u(n) + Wx(n-1)) \\
&\quad - (1-\alpha)x'(n-1) - \alpha f(W^{in}u(n) + Wx'(n-1))||_2 \\
&\leq ||(1-\alpha)(x(n-1) - x'(n-1))||_2 \\
&\quad + ||\alpha f(W^{in}u(n) + Wx(n-1)) - \alpha f(W^{in}u(n) + Wx'(n-1))||_2 \\
&= (1-\alpha)|||x - x'||_2 + \alpha||f(W^{in}u(n) + Wx(n-1)) \\
&\quad - f(W^{in}u(n) + Wx'(n-1))||_2.
\end{aligned}$$

Taking the second term of the previous expression and remembering that $f(x) = tanh(x)$ we get:

$$\begin{gathered}
||tanh(W^{in}u(n) + Wx(n-1)) - tanh(W^{in}u(n) + Wx'(n-1))||_2 \\
\leq max(|tanh'|)||W(x - x')||_2 \leq ||W||_2||x - x'||_2.
\end{gathered}$$

Hence, we finally obtain:

$$\begin{aligned}
||\tilde{F}(s_T(u), x) - \tilde{F}(s_T(u), x')||_2 &\leq (1-\alpha)||x - x'||_2 + \alpha||W||_2||x - x'||_2 \\
&\leq (1 - \alpha + \alpha\sigma(W))||x - x'||_2
\end{aligned}$$

and since, from hypothesis, we have:

$$\begin{gathered}
1 - \alpha + \alpha\sigma(W) = 1 + \alpha(\sigma(W) - 1) < 1 \\
\iff \alpha(\sigma(W) - 1) < 0 \\
\iff \sigma(W) < 1
\end{gathered}$$

from what we can state that $\exists C < 1$ that satisfies the initial equation. □

Remark 1.15. Note that the previous argumentation is valid for any norm in which the states transition equation is a contraction, hence contractivity of such equation, in any norm, is a sufficient condition for the ESP.

We can make a quick resume of the chapter in two points:

- When using a LESN with the ESP we do not have to worry about the initial values of the input and reservoir matrices
- When using a LESN, if $\rho(W) < 1$ we can ensure that the ESP holds

Chapter 2

Classification algorithms

In the field of machine learning, classification algorithms plays a vital role in solving problems where the objective is to assign input data points to predefined categories or classes. These algorithms analyze patterns and relationships within the data to make accurate predictions or decisions about unseen or future instances.

For the later application we will be using the Ridge Regression and Logisitic Regression models. The first one is particularly handy in the mitigation of problems with multicollinearity, which occasionally occurs in models with a large number of features as the one we will be working with. On the other hand, Logistic model, it is less sensitive to outliers compared to some other algorithms and also has the advantage that can be trained relatively quickly. Let's see in more detail what these classifiers consists on. We will start introducing the ordinary least squares model in order to have a better understanding of the other models.

This section is based on [2], pages 19-22 for the Ordinary Least Squares classifier, on [4] for the Logistic classifier. Moreover, we will use some results that can be found in [12], [13] and in Lesson 5.1 from [3].

2.1 Ordinary Least Squares Classifier

Definition 2.1. *Being the residual the difference between an observed value and the fitted value provided by a model, the* ***ordinary least squares method (OLS)*** *is a standard approach in regression analysis to approximate the solution of over determined systems (sets of equations in which there are more equations than unknowns) by minimizing the sum of the squares of the residuals (SSR) made in the results of each individual equation.*

Definition 2.2. *Being n the amount of independent variables and m the size of those*

variables, a ***multivariate linear regression model*** *is defined by*

$$Y_i = \beta_0 + \sum_{j=1}^{m} x_j \beta_j + \epsilon_i \tag{2.1}$$

for $1 \leq i \leq n$ *where* $\epsilon_i \sim N(0, \sigma^2)$, $\beta = (\beta_0, \ldots, \beta_m)^T \in \mathbb{R}^{m+1}$, $Y_i \in \mathbb{R}$ *is the* i^{th} *component of* $Y \in \mathbb{R}^n$ *and* x_j *denotes the* j^{th} *row of the independent variables matrix* $X \in \mathbb{R}^{n \times m}$.

Denoting $x_j = (1, x_1, \ldots, x_m)$ for $1 \leq j \leq n$, we observe the following properties from the model:

- $E(Y) = X\beta$
- $Cov(Y_i, Y_j) = 0$ if $i \neq j$ since we suppose the variables to be independent
- $Var(Y_i) = \sigma^2$ since the variance of each ϵ_i is the same

We also observe from the second and third property that $Cov(Y) = \sigma^2 Id$.

From definition 2.1 and being $Y, \hat{Y} \in \mathbb{R}^n$ the expected and predicted values respectively, using a multivariate linear regression model, we have:

$$SSR = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 = \sum_{i=1}^{n} (Y_i - \hat{\beta_0} - \sum_{j=1}^{m} x_{i,j} \hat{\beta_j})^2 \tag{2.2}$$

which is the error we want to minimize in order to reach the best prediction for $\hat{Y}$.

Proposition 2.3. *The value of* $\hat{\beta} = (\hat{\beta_0}, \ldots, \hat{\beta_m})^T$ *that minimizes the value of SSR in (2.2) is given by the equation:*

$$\hat{\beta} = (X^T X)^{-1} X^T Y \tag{2.3}$$

assuming $X^T X$ *is nonsingular.*

Proof. We will start by seeing that, effectively, $\hat{\beta} = (X^T X)^{-1} X^T Y$ minimizes equation (2.2) and later we will see that there is not any other alternative estimate who minimizes SEE with a smallest value than $\hat{\beta}$.

Denoting as $x_i = (1, x_1, \ldots, x_m) \in \mathbb{R}^{m+1}$ the i^{th} independent variable, we can write equation (2.2) as:

$$\begin{aligned} SSR &= \sum_{i=1}^{n} (Y_i - x_i \hat{\beta}) = ||Y - X\beta||^2 = (Y - X\beta)^T (Y - X\beta) \\ &= Y^T Y - Y^T X\beta - \beta^T X^T Y + \beta^T X^T X\beta. \end{aligned}$$

Now, we will find the minimum of SSR by setting the gradient to zero. Therefore,

$$\frac{\partial SSR}{\partial \beta} = \frac{\partial Y^T Y - Y^T X\beta - \beta^T X^T Y + \beta^T X^T X\beta}{\partial \beta} = -2X^T Y + 2X^T X\beta = 0$$
$$\Rightarrow \quad X^T Y = X^T X\beta$$
$$\Rightarrow \quad \beta = (X^T X)^{-1} X^T Y$$

Next, let's suppose $b = (b_0, \ldots, b_m)^T$ to be an alternative estimate of β, different from $\hat{\beta}$, which minimizes SSR with a smallest value than $\hat{\beta}$, i.e:

$$SSR(b) = (Y - Xb)^T (Y - Xb) < (Y - X\hat{\beta})^T (Y - X\hat{\beta}) = SSR(\hat{\beta})$$

but expanding SSR(b) we get:

$$\begin{aligned} SSR(b) &= (Y - Xb)^T (Y - Xb) = (Y - X\hat{\beta} + X\hat{\beta} - Xb)^T (Y - X\hat{\beta} + X\hat{\beta} - Xb) \\ &= ((Y - X\hat{\beta}) + X(\hat{\beta} - b))^T ((Y - X\hat{\beta}) + X(\hat{\beta} - b)) \\ &= (Y - X\hat{\beta})^T (Y - \hat{\beta}) + [X(\hat{\beta} - b)]^T (Y - X\hat{\beta}) \\ &+ (Y - X\hat{\beta})^T X(\hat{\beta} - b) + [X(\hat{\beta} - b)]^T X(\hat{\beta} - b) \\ &= (Y - X\hat{\beta})^T (Y - \hat{\beta}) + [X(\hat{\beta} - b)]^T X(\hat{\beta} - b) + 2[X(\hat{\beta} - b)]^T (Y - X\hat{\beta}) \\ &= (Y - X\hat{\beta})^T (Y - \hat{\beta}) + (\hat{\beta} - b)^T X^T X(\hat{\beta} - b) + 2(\hat{\beta} - b)^T X^T (Y - X\hat{\beta}). \end{aligned}$$

Now, taking the third term of the last expression and using $\hat{\beta} = (X^T X)^{-1} X^T Y$ we see that:

$$\begin{aligned} 2(\hat{\beta} - b)^T X^T (Y - X\hat{\beta}) &= 2[X(\hat{\beta} - b)]^T (X^T Y - X^T X\hat{\beta}) \\ &= 2[X(\hat{\beta} - b)]^T (X^T Y - X^T X(X^T X)^{-1} X^T Y) = 0. \end{aligned}$$

Therefore, as the second term holds $\hat{\beta} \neq b$, $(\hat{\beta} - b)^T X^T X(\hat{\beta} - b) = ||(\hat{\beta} - b)X||^2 > 0$, we get the following:

$$SSR(b) = (Y - Xb)^T (Y - Xb) > (Y - X\hat{\beta})^T (Y - X\hat{\beta}) = SSR(\hat{\beta})$$

which refutes our initial hypothesis thus proving that $\hat{\beta} = (X^T X)^{-1} X^T Y$ is a global minimum. □

Theorem 2.4. *(**Gauss-Markov Theorem**) Using a linear regression model defined by $Y = X\beta + \epsilon$ which holds that:*

- *the matrix of independent variables X has full-rank*
- $E(\epsilon) = 0$
- $Var(\epsilon) = \sigma^2 Id$

the ordinary least squares estimator $\hat{\beta} = (X^T X)^{-1} X^T Y$ is the best linear unbiased estimator (BLUE), that is, the estimator that has the smallest variance among those that are unbiased and linear in the observed output variables.

Getting back to the model defined by equation (2.1), we already saw that $E(Y) = X\beta$ and $Cov(Y) = \sigma^2 Id$. Therefore, we can say that the least squares estimator $\hat{\beta}$ verify the following properties:

- $\hat{\beta}$ is an unbiased estimator of β
  $E(\beta) = (X^T X)^{-1} X^T E(Y) = (X^T X)^{-1}(X^T X)\beta = \beta$
- $Cov(\hat{\beta}) = (X^T X)^{-1} X^T Cov(Y)((X^T X)^{-1} X^T)^T = \sigma^2 (X^T X)^{-1} X^T (XX^T)^{-1} X = \sigma^2 (X^T X)^{-1}$
- From Gauss-Markov theorem, $\hat{\beta}$ is a best linear unbiased estimator (BLUE)

Although most of the times we can use $\hat{\beta}$ as an estimator of β, there are some cases, when X is a singular matrix, where $\hat{\beta}$ cannot be computed. Consequently, another estimator exists, called Ridge estimator which allows to compute an estimator of β even if X is a singular matrix. Moreover, this new estimator controls the amount of regularization applied to the model, a technique used to control overfitting and underfitting. It is also a good option when dealing with multicollinearity (high correlation) among input variables. However, one of the requirements the estimator will have to abandon is to be unbiased. We will see, though, that Ridge estimator is a good alternative even being biased.

2.2 Ridge Regression Classifier

Using the same definition of the model as in definition 2.2, ridge regression estimator solves a slightly modified minimization problem since we look for a $\tilde{\beta}$ which minimizes the sum of squared residuals plus the squared norm of of the vector of coefficients, $\tilde{\beta}$, i.e:

$$SSR = \sum_{i=1}^{n} (Y_i - x_i \tilde{\beta}) + \lambda \sum_{j=1}^{m} \beta_j^2 = ||Y - X\beta||^2 + \lambda ||\beta||^2 \tag{2.4}$$

where $\lambda > 0$ is a positive constant, named penalty. In order to find the best value for λ, what is done most times is to try a large amount of values for it and keep the best one.

Proposition 2.5. *The value of* $\tilde{\beta} = (\tilde{\beta_0}, \ldots, \tilde{\beta_m})^T$ *that minimizes the value of SSR in (2.4) is given by the equation:*

$$\tilde{\beta} = (X^T X - \lambda Id)^{-1} X^T Y \tag{2.5}$$

for some penalty $\lambda > 0$.

Proof. We will follow the same idea as before. Firstly, we will see that $\tilde{\beta} = (X^T X - \lambda Id)^{-1} X^T Y$ solves the minimization problem

$$SSR = Y^T Y - Y^T X\beta - \beta^T X^T Y + \beta^T X^T X\beta + \lambda\beta^T\beta.$$

Now, setting the gradient to 0, we find that, effectively, $\tilde{\beta}$ minimizes the value of SSR.

$$\begin{aligned} \frac{\partial SSR}{\partial \beta} &= -2X^T Y - 2X^T X\beta + 2\lambda\beta = 0 \\ \Rightarrow \quad & X^T Y = X^T X\beta - \lambda\beta \\ \Rightarrow \quad & \beta = (X^T X - \lambda Id)^{-1} X^T Y \end{aligned}$$

Next, let's see that $\tilde{\beta}$ is indeed a global minimum. If we compute the Hessian matrix of SSR, we get:

$$\frac{\partial^2 SSR}{\partial \beta^2} = 2(X^T X + \lambda Id) > 0$$

since $X^T X + \lambda Id$ is a positive definite matrix for any $\lambda > 0$. Hence, we can state that $\tilde{\beta}$ is a global minimum.

We need to prove that $X^T X + \lambda Id$ is positive definite. Being a vector $v \in \mathbb{R}^{n+1}$ not null,

$$v^t(X^T X + \lambda Id)v = (Xv)^t(Xv) + \lambda v^t v = \sum_{i=0}^{m}(x_i v)^2 + \lambda \sum_{j=0}^{m} v_j^2 > 0$$

since there must exist at least one j such that $v_j \neq 0$ in order to be $\tilde{\beta} \neq 0$. □

As before, using properties defined for the multivariate linear regression model, we get the following:

- $E(\tilde{\beta}) = (X^T X - \lambda Id)^{-1} X^T X\beta$
- $Bias(\tilde{\beta}) = E(\tilde{\beta}) - \beta = [(X^T X + \lambda Id)^{-1} - (X^T X)^{-1}] X^T X\beta$
- $Cov(\tilde{\beta}) = \sigma^2 (X^T X + \lambda Id)^{-1} X^T X (X^T X + \lambda Id)^{-1}$

The following result proves that the covariance matrix of ridge estimator is lower than the covariance matrix of the OLS estimator. It does not contradict the Gauss-Markov theorem since the ridge estimador is biased.

Corollary 2.6. *Being $\hat{\beta}$ the estimator obtained using OLS method and $\tilde{\beta}$ the ridge estimator, then*

$$Cov(\tilde{\beta}) < Cov(\hat{\beta})$$

Proof. Firstly, let's see that two covariance matrices can be compared by checking whether their difference is positive definite.

$$\begin{aligned} Cov(\tilde{\beta}) < Cov(\hat{\beta}) &\iff Cov(a\tilde{\beta}) < Cov(a\hat{\beta}), \forall \text{ constant vector } a \\ &\iff aCov(\tilde{\beta})a^t < aCov(\hat{\beta})a^T \\ &\iff a(Cov(\tilde{\beta}) - Cov(\hat{\beta}))a^T > 0 \\ &\iff Cov(\tilde{\beta}) < Cov(\hat{\beta}) \end{aligned}$$

is positive definite.

Denoting $W = X^T X(X^T X - \lambda Id)^{-1}$ we have:

$$\begin{aligned} Cov(\hat{\beta}) - Cov(\tilde{\beta}) &= \sigma^2(X^T X)^{-1} - \sigma^2 W^T (X^T X)^{-1} W \\ &= \sigma^2[W^T (W^T)^{-1}(X^T X)^{-1} W^{-1} W - W^T (X^T X)^{-1} W] \\ &= \sigma^2 W^T[(W^T)^{-1}(X^T X)^{-1} W^{-1} - (X^T X)^{-1})]W \\ &= \sigma^2 W^T[(X^T X)^{-1}(X^T X + \lambda Id)(X^T X)^{-1}(X^T X + \lambda Id)(X^T X)^{-1} \\ &- (X^T X)^{-1}]W \\ &= \sigma^2 W^T[(Id + \lambda(X^T X)^{-1})(X^T X)^{-1}(Id + \lambda(X^T X)^{-1}) - (X^T X)^{-1}]W \\ &= \sigma^2 W^T[((X^T X)^{-1} + \lambda(X^T X)^{-2})(Id + \lambda(X^T X)^{-1}) - (X^T X)^{-1}]W \\ &= \sigma^2 W^T[((X^T X)^{-1} + \lambda(X^T X)^{-2} + \lambda(X^T X)^{-2} + \lambda^2(X^T X)^{-3} \\ &- (X^T X)^{-1}]W \\ &= \sigma^2 W^T[2\lambda(X^T X)^{-2} + \lambda^2(X^T X)^{-3}]W \\ &= \sigma^2(X^T X + \lambda Id)^{-1} X^T X(2\lambda(X^T X)^{-2} + \lambda^2(X^T X)^{-3})X^T X(X^T X + \lambda Id)^{-1} \\ &= \sigma^2(X^T X + \lambda Id)^{-1}[2\lambda Id + \lambda^2(X^T X)^{-1}](X^T X + \lambda Id)^{-1} = M \end{aligned}$$

If $\lambda > 0$, then $M = Cov(\hat{\beta}) - Cov(\tilde{\beta})$ is positive definite since for any $v \neq 0$ we have:

$$z = (X^T X - \lambda Id)^{-1} v \neq 0$$

and

$$v^T M v = \sigma^2 x^T(2\lambda Id + \lambda^2(X^T X)^{-1})z = \sigma^2\lambda z^T z + \sigma^2\lambda^2 z^T(X^T X)^{-1}z > 0$$

because $X^t X$ and its inverse are both definite positive. □

All the previous results may lead us to discuss how different $\tilde{\beta}$ and $\hat{\beta}$ are in terms of the MSE.

Proposition 2.7. *The **mean squared error (MSE)** of an estimator β is written as:*

$$MSE(\beta) = trace[Cov(\beta)] + ||Bias(\beta)||^2$$

Therefore, we have $MSE(\tilde{\beta}) = trace[Cov(\tilde{\beta})] + ||Bias(\tilde{beta})||^2$ and $MSE(\hat{\beta}) = trace[Cov(\hat{beta})]$.

Then,

$$\begin{aligned} MSE(\hat{\beta}) - MSE(\tilde{\beta}) &= trace[Cov(\hat{beta}) - Cov(\tilde{\beta})] - ||Bias(\tilde{beta})||^2 \\ &= trace(N) - ||Bias(\tilde{beta})||^2 \end{aligned}$$

where $N = Cov(\hat{beta}) - Cov(\tilde{\beta})$

Since we already proved that N is strictly positive definite in the proof of the last Corollary, its trace will also be strictly positive and the square of the bias will also be positive. This means that the difference can be positive or negative. It is possible to prove that whether the difference is positive or negative depends on the penalty parameter λ, and it is always possible to find a value for λ such that the difference is positive.

Thus, there always exists a value of the penalty parameter such that the ridge estimator has lower mean squared error than the OLS estimator.

This result is very important because even having the OLS estimator the lowest variance (and the lowest MSE) among the estimators that are unbiased, there exists a biased estimator (ridge estimator) whose MSE and variance is lower than the one of the OLS estimator.

2.3 Logistic Regression Classifier

Definition 2.8. *Being n the amount of independent variables, m the size of those variables and given $X \in \mathbb{R}^{n\times(m+1)}$ the input matrix with an added column of ones, the **Logistic regression model** is defined as:*

$$P(Y_i|\beta) = \frac{1}{1 + e^{-Y_i(\beta^T X_i)}} \tag{2.6}$$

where $\beta \in \mathbb{R}^{m+1}$ are the weights of the model.

As it can be seen, the model is no more than a linear model to which it has been applied the non-linear sigmoid function.

Definition 2.9. *In statistics, **maximum likelihood estimation (MLE)** is a method of estimating the parameters of an assumed probability distribution, given some observed data. This is achieved by maximizing a likelihood function so that, under the assumed statistical model, the observed data is most probable.*

Remark 2.10. Because the logarithmic function is monotonic, maximizing the likelihood is the same as maximizing the log of the likelihood (i.e., log-likelihood). That is exactly the same to take the negative of the log-likelihood and minimize that, resulting in the well known Negative Log-Likelihood Loss.

From the definitions above, we can define our loss function as:

$$L(\beta) = \sum_{i=1}^{n} log(\frac{1}{1 + e^{-Y_i \beta^T X_i}}). \tag{2.7}$$

Moreover, to obtain good generalization abilities, one adds a regularization term to the loss function. In our case, we will be adding the term $\lambda \sum_{i=1}^{m+1} \beta_i^2$ for $\lambda > 0$.

Therefore, the loss function can be rewrote as:

$$L(\beta) = \frac{1}{2} \sum_{i=1}^{m+1} \beta_i^2 + C \sum_{i=1}^{n} log(\frac{1}{1 + e^{-Y_i \beta^T X_i}}) \tag{2.8}$$

where $C = \dfrac{1}{2\lambda} > 0$ is a parameter to be determined by the user.

In order to find the value, $\bar{\beta}$, that minimizes the equation (2.8) there are different optimization methods we can use. Among all of them, we will use the simplest Newton method, which uses the following expression to update the weights at every iteration:

$$\beta^{k+1} = \beta^k + s^k \tag{2.9}$$

where k is the iteration which we are computing and s^k, the search direction values, is the solution of the following linear system:

$$\nabla^2 L(\beta^k) s^k = -\nabla L(\beta^k). \tag{2.10}$$

However, when using the previous equation, there is no guarantee that the values of $\bar{\beta}$ converge. To avoid this problem, there are some techniques that can be applied such as the line search. This consists on calculating, at each iteration, a suitable steplength $a^k > 0$ so that $L(\beta + a^k s^k) < L(\beta^k)$.

Another problem we might face is due to the computations needed when trying to solve the system defined in (2.10). For instance, we know that:

$$\nabla L(\beta) = \beta + C \sum_{i=1}^{n} \frac{Y_i X_i}{1 + e^{-Y_i \beta^T X_i}}$$
$$\text{and}$$
$$\nabla^2 L(\beta) = Id + C X^T D X$$

where $D \in \mathbb{R}^{n \times n}$ is a diagonal matrix the components of which are defined as:

$$D_{ii} = \frac{1}{1 + e^{-Y_i \beta^T X_i}} (1 - \frac{1}{1 + e^{Y_i \beta^T X_i}}).$$

When computing the Hessian matrix, we can find a dense matrix for the value of $X^T DX$ which might be too large to store it. One possible solution for this problem, and also to make the computations faster, is not to calculate the Hessian matrix but to compute the left side of equation (2.10) as follows:

$$\nabla^2 L(\beta^k) s^k = (Id + CX^T DX) s^k = s^k + CX^T (D(Xs^k)).$$

Chapter 3

Application of a leaky echo state network to EEG signal classification

3.1 Experiment explanation and data provided

Following the results achieved in article [1], the aim of the project is to determine if temporal data can help us to determine if we, humans, are influenced by social pressure by looking at some time series.

EEG signals are inherently temporal in nature, representing the electrical activity of the brain over time. By incorporating Leaky Echo State networks as a feature extraction technique, we aim to leverage the temporal dynamics and memory properties of the network to capture important temporal dependencies within the EEG signals. By comparing the performance of the Ridge regression and Logistic regression classifiers, we seek to determine the impact of temporal data on the classification accuracy and assess whether capturing temporal dynamics improves the classification of EEG signals into distinct brainstates.

In particular, we will be looking to the electroencephalogram signals human brain generates when doing the same activity in three different sates of social pressure. The complete sessions of the experiments are described in detail in the mentioned article but for a lighter follow-up of the incoming section, I will start by providing a quick resume of the those experiments and a description of the data obtained from them.

Although several subjects participated on the study, we will only be focusing on the data generated by one of them. Along it, each subject was exposed to three different situations, therefore, three different states of social pressure, when

carrying out the same activity. In each of them, the subject was asked to be as fast and precise as possible at hitting the center of a provided target.

During the first situation, the subjects performed repeatedly the activity alone, and were shown, at the end of each trial, their respective accuracy. From now on, we will name this state as solo. For the two other states, two types of simulated players were created: one who would perform the activity with less accuracy than the subject did in the first state and one who would perform it much better than the subject on its first state. In this project, we will be naming those states easy and hard, respectively. It is also worth mentioning that the subjects were never asked to compete with others but quite the contrary. However, after some trials, their respective accuracy from each trial they carried out was shown to both of them.

Being the study the one described right above, let's jump into the data we got from it for every subject. As we said, there were three different states in each of which the subject was performing twice a total of 108 trials of the activity. Finally, one more complete sequence as the one defined was repeated. This leaves us, for each subject, a total of 12 block of 108 trials each. Those 12 blocks are counted as: 2 times per state * 3 different states * 2 times each sequence.

For every trial, an encephalogram (EEG) was recorded using a total of 60 electrodes. From those, a Recursive Feature Elimination (RFE) was used to discard some of the channels that might not be generating useful information thus reducing to 42 the 60 initial ones. Also an Independent Component Analysis (ICA) was implemented in order to reduce eye movement artefacts that could bias the results of the study. Finally, from every of the remaining channels, the 800 ms previous to the movement and the 400 ms following it were kept, getting a final length of 1200 ms per channel.

Regarding the experiment explained above we can summarize our initial data as a 4-dimension tensor of size $\mathbf{R}^{42}x\mathbf{R}^{1200}x\mathbf{R}^{108}x\mathbf{R}^{12}$ where:

- 42 is the number of channels
- 1200 is the length of each channel
- 108 is the number of trials realised in each block
- 12 is the number of blocks for each subject

In Figure 3.1 we see a picture that will help us understand the data we deal with.

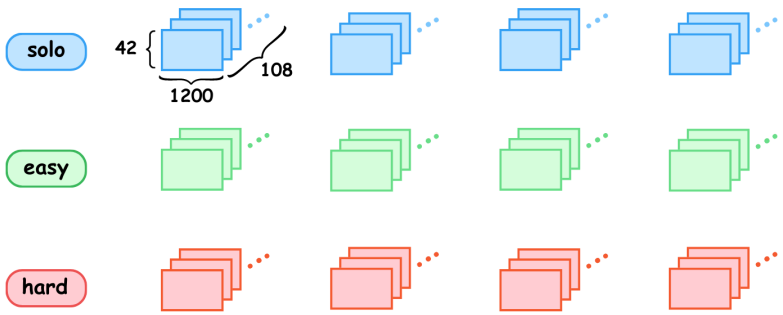


Figure 3.1: Data provided

3.2 Data preprocessing

In this chapter, we delve into the crucial step of data preprocessing and to be more specific, of data transformation. If we took the data as it is provided to us, we would need to train our network and to work with a 4D object. As you can imagine, it is not practical nor easy to work with it. Firstly, it will trigger to a much more complex code to train the network in terms of readability. Moreover, it is much challenging for us to create a mental picture of the data we deal with. Therefore, and only because our data allows it, we will try to join some data together before giving it as the input of the reservoir.

Looking back to the definition of the experiment, for every subject we have: 2 repetitions of the activity per state and this sequence repeated twice. This means that it is possible to join per state into three groups of four our initials 12 blocks since they comprise repetitions of the same activity. To make it more understandable, looking at Figure 3.1, we took, for each of the states, the blocks from columns 2,3 and 4 and concatenated them with the ones on the first column. This leaded us to have the data organised as it is shown in Figure 3.2.

However, we still had a 4D tensor, or what is the same, a total of three 3D tensors. The way we decided to proceed was to join those three 3D tensors in one, obtaining consequently, a 3D tensor of size $\mathbf{R}^{42}x\mathbf{R}^{1200}x\mathbf{R}^{1296}$. Finally, to make the computations affordable for the computer in terms of memory, we sampled the data by time, keeping one sample every 10 ms thus acquiring a 3D tensor of size $\mathbf{R}^{42}x\mathbf{R}^{120}x\mathbf{R}^{1296}$.

It is well known that EEG signals reflect the electrical activity of the brain. Moreover, every signal can be studied in different frequency bands which is quite interesting since each of them is associated with a specific mental state and cognitive processes. Therefore, some of them may provide us more accuracy than

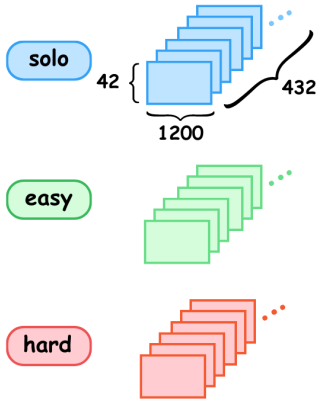


Figure 3.2: Data joined per states

others. In this project we decided to split our data in three different band frequencies and study each of them separately. The band frequencies we used are:

- alpha : 8 Hz to 12 Hz
- beta : 15 Hz to 30 Hz
- gamma : 40 Hz to 80 Hz

3.3 Leaky echo state network application

In this section we will be explaining which is the architecture we used to classify our data into the three classes: solo, easy and hard, previously defined. It consisted, along general lines, in two main steps. In the first one, we applied a LESN using as input the data we obtained from the preproccessing stage. In the second step, using the output the LESN generated readout layer), we used a Ridge Regression classifier or a Logistic Regression classifier to determine to which class each signal belonged to. We can see in the following image a visual representation of the network's architecture and how the training data was being modified through it until reaching the final output.

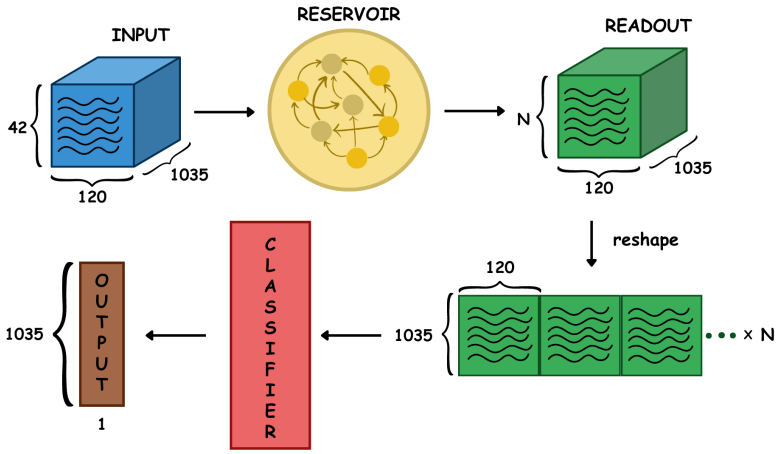


Figure 3.3: Architecture of our network

In order to train the network, we decided to split our initial dataset in two: train set and test set. The first one, which comprises the 80% of the initial dataset, was the one used to train the network. It contained a total of 1035 trials of each channel. The second one, the test set, was used to determine how good our network was doing it classifying the brainstates. It was created from the remaining 20% of the initial data set and consisted of 261 trials of each channel.

3.3.1 Reservoir layer

This layer was in charge of processing the temporal information of the input data. To do so, we used a Recurrent Neural Network called Leaky Echo State which is based on reservoir computing. Hence, in order to update the values of the reservoir weight's matrix, we used the state transition equation from definition 1.7 where the activation function is $f(x) = tanh(x)$.

Taking into consideration all the previous results and since LESN model holds the Echo State Property, we can initialize our input weights matrix W^{in} and reservoir weights matrix W with multiple values. Remember that the ESP states that the initial input values will only influence the dynamics of the nodes on the first time steps and, after some time, the trajectories will converge to the same one. In our case, we decided to initialize the sparse inputs and reservoir weights matrices with random values sampled from uniform distribution in $[-1, 1]$ with connectivity to be determined.

The only values we needed to determine were the **input probability** parameter, which defines the probability of connection between the input and reservoir layer parameters and the **reservoir probability** parameter, that defines which is

the probability of connection between the neurons inside the reservoir layer. We made some tests, trying different possible values, to find the ones which provided the best results. Finally, for Logistic Regression classifier we found that the best values were input probability = reservoir probability = 0.5. On the other hand, for Ridge Regression classifier we found that, for alpha frequency data we should use input probability = 0.2 and reservoir probability = 0.3 whereas for alpha and gamma frequency bands the values should be input probability = 0.05 and reservoir probability = 0.5.

Moreover, we also wanted to know which was the amount of neurons needed in the reservoir to obtain the best performing of the network. After some tests we obtained that the best amount of neurons when using the Logistic Regression classifier was 200 for any band frequency. However, when using the Ridge Regression classifier, the ideal amount should be of 75 for any frequency band.

Additionally, when using the Ridge Regression classifier we fixed the penalty to 1. Note that for penalty = 0, the classifier is, actually, the OLS classifier.

Lastly, we also needed to study the behaviour of the network regarding the leaking rate value used in the state transition equation. We found that the value which provided better results was when using $\alpha = 0.01$ for both classifiers.

All the tests made in order to decide which values were the best for all the last mentioned parameters are discussed in Chapter 4.

3.3.2 Classification layer

The classification layer was the one responsible for classifying the received input data into its corresponding brain state. We decided to study the behaviour of the network when using the Ridge Regression model and Logistic Regression model. For the first one, we decided to use as penalty the value $\lambda = 1$ for all frequency bands, as we saw it was the value which made the network perform better. For the second one, we decided to use as the method of computing the weights, the Newton algorithm since it gave us more accurate results than the others.

To generate the labels, when using the Logistic Regression we labeled as 0 the trials from solo state, as 1 the trials from easy state and as 2 the trials from hard state. On the other side, when using Ridge Regression classifier we applyied the hot encoding strategy to the labels thus obtaining vectors of dimension $n \times 3$ where n is the number of trials.

3.4 Hyperparameters

Hyperparameters are values that are set before the learning process begins. They must be predefined by the user and play a critical role in the training process since they have a significant impact on the performance and behavior of the model. As a consequence, in this section we will be discussing, for each of our hyperparameters, which is the value that allows our model perform the best. Particularly, we will be studying the number of neurons of the reservoir layer, the input and reservoir probability, the leaking rate, α, used in the states transition equation and the penalty, λ, used in the Ridge Regression classifier.

Firstly, in order to determine the number of neurons we should use in our reservoir, we took a set of possible values for it. We tested the accuracy of the network for all of them with each frequency band, separately. During the tests, the other hyperparameters were set to: input probability = 0.05, reservoir probability = 0.05, leaking rate = 2 and penalty = 0.01.

We can see, in Figure 3.4, the results obtained using a box plot for Rhe Ridge regression classifier. From the graphics we can observe that, for all frequencies, as expected, the accuracy of the network improves as the number of neurons increases until reaching the value of 75 neurons, where we see the maximum accuracy. For higher values of number of neurons the precision decreases, probably, due to overfitting.

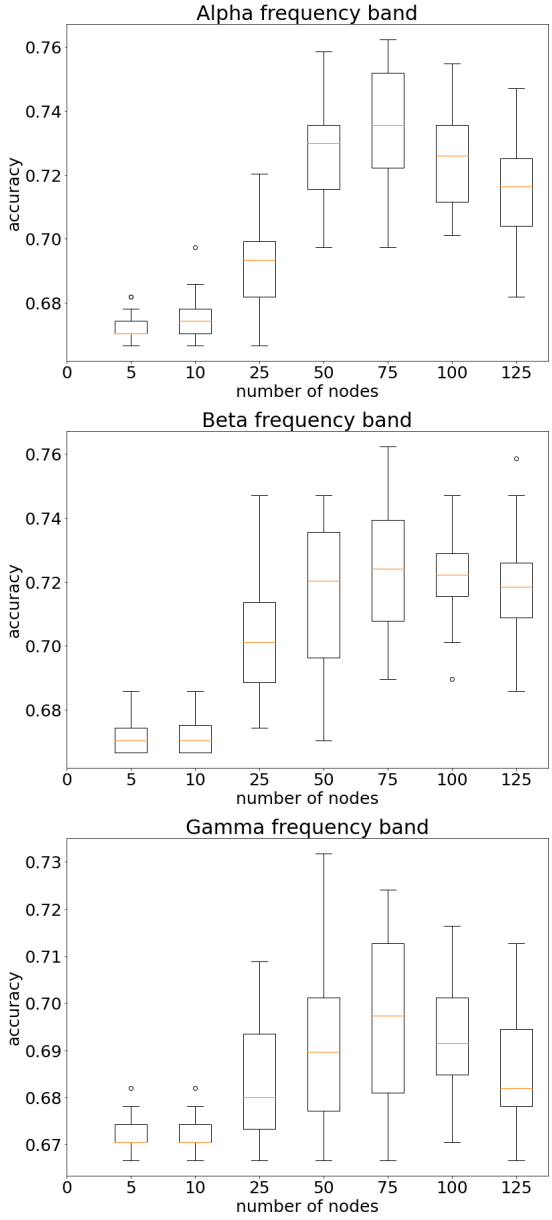


Figure 3.4: Accuracy vs number of neurons for different frequency bands using Ridge Regression classifier

Regarding the results using the Logistic Regression, we can see a very similar tendency to improving as number of nodes increases. However, the accuracy this classifier gives us is much better than the one provided by the Ridge Regression classifier. For instance, it is possible to see in Figure 3.5 that for 200 nodes a mean accuracy of 0.989 is reached when using beta frequency data while the highest pre-

cision obtained using Ridge Regression classifier was 0.734 using alpha frequency data. Another interesting fact to look at is that unlike with Ridge Regression, the model seems it does not reach overfitting at any point even using much more nodes.

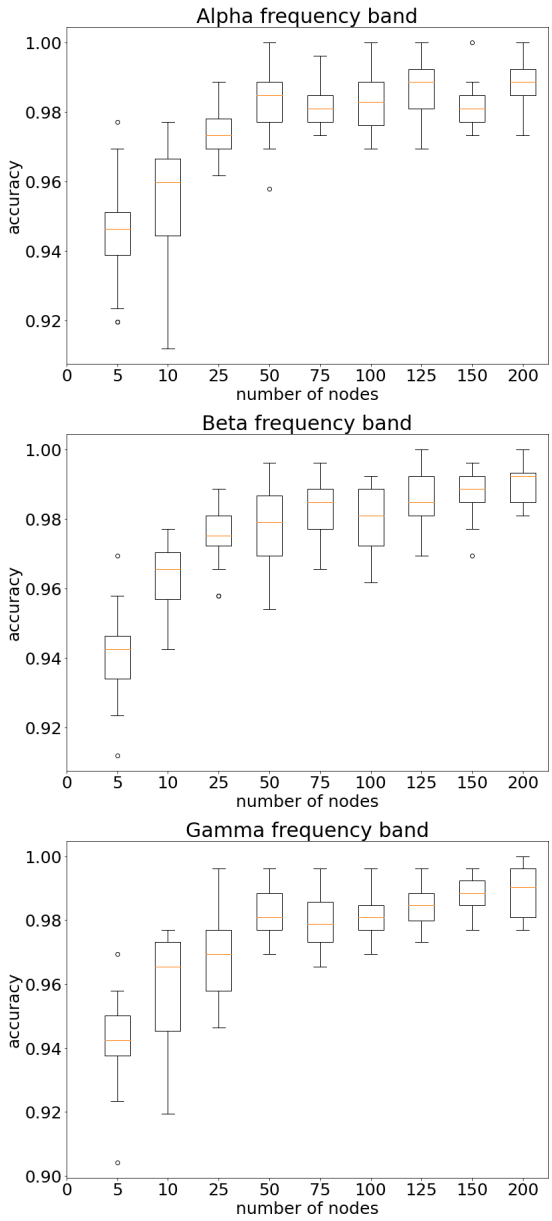


Figure 3.5: Accuracy vs number of neurons for different frequency bands using Logistic Regression classifier

From now on, when studying the other hypermarameters with Ridge Regres-

sion, we will fix the number of nodes to 75 whereas when using the Logistic classifier we will fix the number of nodes to 200. The other hypermarameters remain the same unless otherwise specified .

Next step is the study of the penalty hyperparameter $\lambda > 0$ used in the Ridge Regression classifier. Notice that, in particular, if $\lambda = 0$, the classifier is indeed the Ordinary Least Squares instead of the Ridge Regression classifier. As we already said, the penalty parameter controls the amount of regularization applied to the model. The higher the penalty, it reduces the magnitude of coefficients. Therefore, it is used to prevent multicollinearity, and it reduces the model complexity by coefficient shrinkage. In Figure 3.6, we see that the Ordinary method does not provide good results for any of the frequency bands. Furthermore, we can see from the graphs that the value given to the penalty it is not relevant for the good performance of the network as long as it is different from 0.

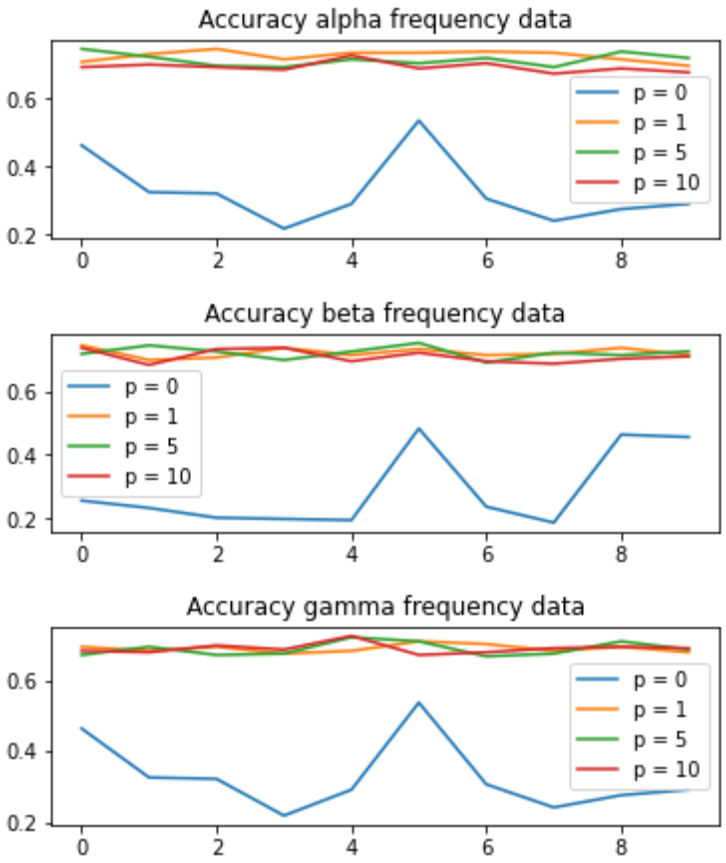


Figure 3.6: Accuracy vs penalty using data from different frequency bands

The next hyperparameter we are going to study is the leaking rate. It is used

when computing the nodes weights in the reservoir layer and typically refers to the rate at which the resting potential of a neuron decays over time. We understand the resting potential as the baseline state from which the neuron can be activated to transmit signals. For example, a leaking rate of 0.1 means that the resting potential decreases by 10% of its current value per unit of time. Therefore, by adjusting the leaking rate, you can control the temporal dynamics of your data. Higher leaking rates tend to make the neuron more responsive to inputs characterized by their brief duration or occurrence, while lower leaking rates make it more sensitive to inputs which persist for an extended period of time. By looking at Figure 3.7 we can clearly see that, for Logistic Regression classifier, in order to achieve a good performance, pretty low leaking rates are needed. For that very reason, we can state that the temporal component of our data plays an important role in the classification task, as we had already assumed at the beginning of the project.

Thanks to the results obtained and although it would seem that the best value for α should be 0.001, we found that for such a small value, the optimization method the classifier uses, Newton method, failed repeatedly to converge reaching the maximum number of iterations fixed by the algorithm. Therefore, aiming to make the model more efficient avoiding having to do so many iterations, we decided to fix the best value for α to 0.01. It must be said that the decision was also made as the improvement that $\alpha = 0.001$ provides over $\alpha = 0.01$ is not decisive.

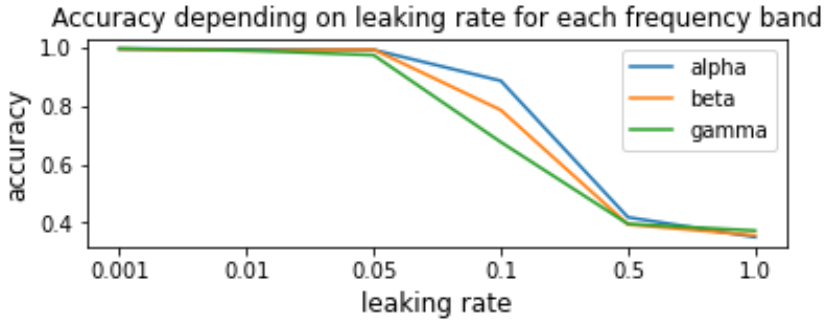


Figure 3.7: Accuracy vs leaking rate when using Logistic Regression classifier

We obtained similar results when using the Ridge Regression classifier. However, with this classifier, we can see a little improvement when using leaking rate = 0.01. For that reason and previous ones, we also decided to fix the leaking rate to 0.01 when using Ridge classifier.

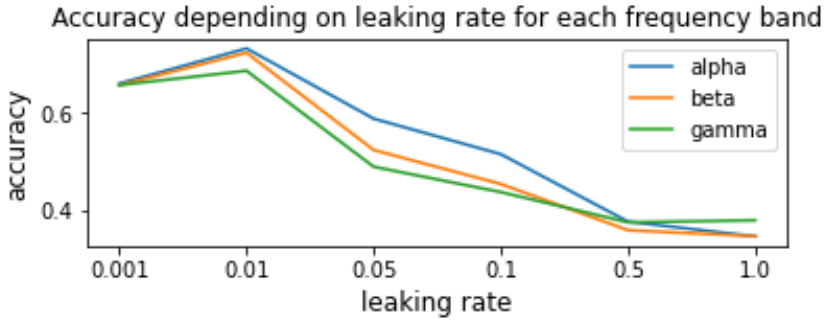
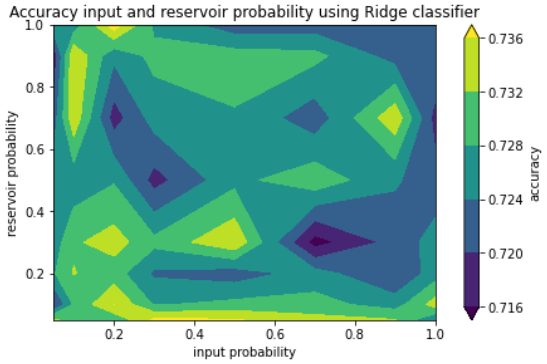


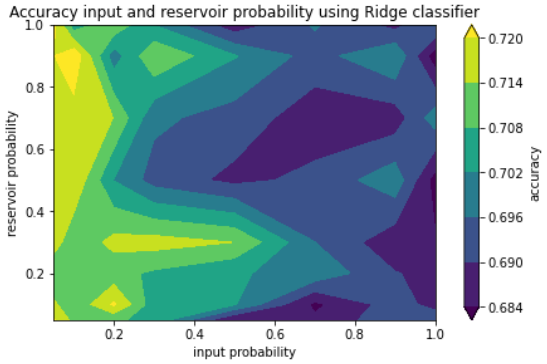
Figure 3.8: Accuracy vs leaking rate when using Logistic Regression classifier

To conclude this section, let's examine the impact of the input probability and reservoir probability on achieving the highest possible accuracy. The input probability hyperparameter controls the sparsity of connection between input neurons and reservoir neurons as it refers to the probability of each individual connection between those neurons. By varying its value, we can control the amount of information that is propagated to the reservoir layer being the more higher the value the more information spread. In a very similar way, reservoir probability hyperparameter controls the sparsity of connection but this time between reservoir neurons with themselves. Varying its value can lead to variations in the reservoir's dynamics which will have something to do with temporal information of our data. Let's also not that higher values of both hyperparameters lead to more dense weights matrices thus more memory capacity and computational capabilities.

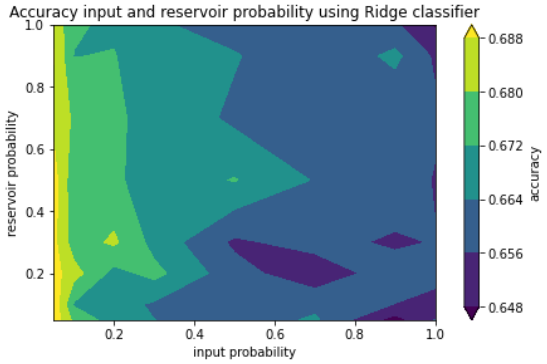
We tested the accuracy of the network for the following values of the input and reservoir probabilities: $0.05, 0.1, 0.2, 0.3, 0.5, 0.7, 0.9, 1$. Looking at the legends of Figure 3.9 and Figure 3.10, we can see that there's only a little difference between the best and the worse accuracy when using any of the classifiers. However, for Ridge classifier we see that we get better results when using alpha frequency data. Moreover, we see that the best values for the hyperparameters in this case should be quite low, at least, for one of both. For instance, the best accuracy, 0.7383, for Ridge classifier is achieved using input probability = 0.2 and reservoir probability = 1. However, for beta and gamma frequency data cases, it is pretty clear that the best results are obtained for input value 0.05 and any value of reservoir proababiliy.



(a) Alpha



(b) Beta

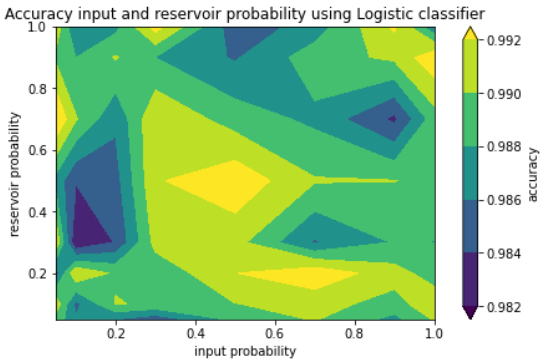


(c) Gamma

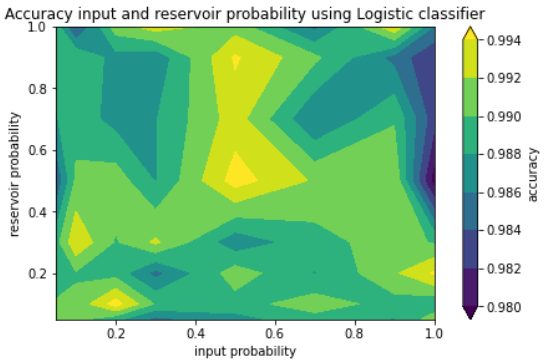
Figure 3.9: Accuracy network for different values of input and reservoir probability using Ridge classifier

For the Logistic classifier the best accuracy, 0.994, is reached when using beta frequency data. Actually, from the mean values used to plot the graphs, we see that it is reached in 4 different cases. Looking at all graphs, we see that alpha and beta frequency data seem to have similar behaviours where we can say that

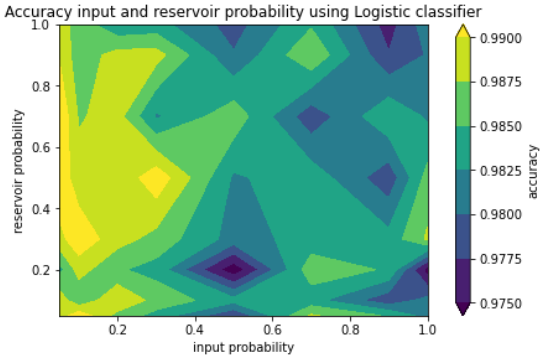
good values for the input probability and reservoir probability would be 0.5 for both. However, with gamma frequency data, we see that the best accuracy is achieved using low values of input probability, such as 0.1 and values around 0.4 for reservoir probability.



(a) Alpha



(b) Beta



(c) Gamma

Figure 3.10: Accuracy network for different values of input and reservoir probability using Logistic classifier

After seeing that the results are not very significant for different values of input and reservoir probabilities and taking into account that higher probabilities would mean more dense matrices thus computationally expensive, we decided that the best values for those parameters are input probability = 0.2 and reservoir probability = 0.1 for Ridge classifier. On the other hand, for Logistic classifier we decided to use input probability = reservoir probability = 0.5.

3.5 Code

The code used to build the network consists of a main class and three complementary classes: the data class, the network class, and the reservoir class. All files can be found in the following github repository: leaky echo state network for classifying.

- In the main class we define some of the parameters and the classifier we want to use for the network. Is the one in charge of gathering all the others and the one you need to run to get the output of the program.
- The data class is responsible for processing and dealing with the original data.
- The network class contains the code for the state transition equation and for each of the classifiers.
- The reservoir class serves as a container that brings together the functionality of the data class and the network class. It manages, as well, the computation of the accuracy of the network.

Chapter 4

Results and conclusions

In this project, we investigated the classification of EEG signals into three distinct brainstates using leaky echo state networks (ESN) in combination with two different classifier algorithms: Ridge regression and Logistic regression. Following the work made by Xènia Domènech in her final project, the primary aim of this project was to investigate the utility of temporal data in classifying the signals into their brainstates since the influence of temporal information was not tackled in her study. Secondly, we also wanted to determine if the classifier we used was decisive for the good performance of the network.

The results indicate that the Logistic regression classifier outperforms the Ridge regression classifier in accurately classifying the signals. Specifically, we obtained a maximum mean accuracy of 0.994 when using Logistic classifier while the best results obtained for Ridge classifier rounded an accuracy of 0.7383. This finding suggests that the Logistic regression algorithm is better suited for this particular classification task, showcasing its power and ability to capture the underlying patterns in the EEG data. The superior performance of the Logistic regression classifier can be attributed to its ability to model nonlinear relationships and handle multiclass classification tasks effectively.

We can also see, as mentioned when discussing Figure 3.7, that the temporal dynamics captured by the reservoir are very useful when classifying data in brainstates. It's also worth pointing out that looking at the results obtained in Xènia's work, although the difference between using distinct classifiers is really tiny, there's an improvement from using Logistic Regression to Linear Regression classifier. In our case, it is completely the opposite and this can make us realize that temporal information in EEG signals is the characteristic that brings the nonlinear relationships inside our data.

one more thing we extract from the results obtained is that there is no big difference between using alpha, beta or gamma frequency band. However, if we

needed to decide for one of them, we would choose alpha when combining it with Ridge classifier and beta when using Logistic classifier.

One last thing that surprised me is the little improvement gained with the study of the hyperparameters, especially when using Logistic classifier. This could mean that we have came across with very robust architecture. In other words, that it is capable of learning and generalizing well regardless of the specific hyperparameter values. This could indicate that the architecture is well-suited for the given task and dataset, and that it can adapt to different settings without significant changes in performance.

In conclusion, the project highlights the effectiveness of the Logistic regression classifier combined with leaky echo state networks for accurate classification of EEG signals into three brain states.

Bibliography

[1] Ignasi Cos, Gustavo Deco, Mathieu Gilson, *Behavioural and Neural Correlates of Social Pressure during Decision-Making of Precision Reaches, version 1. Available at Research Square https : //doi.org/10.21203/rs.3.rs − 1974463/v1*, (2022)

[2] Alvin C. Rencher, William F. Christensen, *Methods of Multivariate Analysis, Third Edition*, (2012).

[3] PennState Eberly College of Science, *Applied Data Mining and Statistical. Available at https : //online.stat.psu.edu/stat857/node/155/*, course 897D, Lesson 5.1.

[4] Chih-Jen Lin, Ruby Chiu-Hsing Weng, *Trust Region Newton Method for Logistic Regression. Available at Journal of Machine Learning Research 9, https : //jmlr.org/papers/v9/lin08b.html*(2008).

[5] Claudio Gallicchio, *Euler State Networks: Non-dissipative Reservoir Computing. Available at https : //doi.org/10.48550/arXiv.2203.09382*, (2023).

[6] Matteo Cucchi, *Hands-on reservoir computing: a tutorial for practical implementation, Neuromorph. Comput. Eng. 2 032002. Available at* , (2022).

[7] Claudio Gallicchio, Alessio Micheli, *Echo State Property of Deep Reservoir Computing Networks. Cogn Comput 9, 337–350. https : //doi.org/10.1007/s12559 − 017 − 9461 − 9*, (2017).

[8] Herbert Jaeger, *The "echo state" approach to analysing and training recurrent neural networks – with an Erratum note.Bonn, Germany: German National Research Center for Information Technology GMD Technical Report. 148.*, (2010).

[9] Khalid Saeed, Władysław Homenda(Eds), *Computer Information Systems and Industrial Management. Available at https : //link.springer.com/book/10.1007/978 − 3 − 319 − 99954 − 8*, (2016).

[10] Siddharth Sharma, Simone Sharma, Anidhya Athaiya, *Activation Functions in Neural Networks, International Journal of Engineering Applied Sciences and Technology, Vol. 4, Issue 12, ISSN No. 2455-2143, Pages 310-316, available at http : //www.ijeast.com/*, (2020).

[11] Xènia Domènech Gutiérrez, Ignasi Cos, Gorka Zamora, *Modularization of reservoir computing networks for the recognition of brainstates. Available at http : //hdl.handle.net/*2445/187760, (2022).

[12] Taboga, Marco, *"Ridge regression", Lectures on probability theory and mathematical statistics. Kindle Direct Publishing. Online appendix. https://www.statlect.com/fundamentals-of-statistics/ridge-regression* (2021).

[13] Taboga, Marco, *"Gauss Markov theorem", Lectures on probability theory and mathematical statistics. Kindle Direct Publishing. Online appendix. https://www.statlect.com/fundamentals-of-statistics/Gauss-Markov-theorem* (2021).