Utility of Machine Learning Techniques in Classifying EEG Data of Stroke Patients

Dean Connell
18332986

CS460 Final Year Project – 2021
B.Sc. Computational Thinking

Department of Computer Science

Maynooth University

Maynooth, Co. Kildare

Ireland

A thesis submitted in partial fulfilment of the requirements for the B.Sc.

In Computational Thinking

Supervisor: Joseph Duffin.

Contents

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study as part of the B.Sc. Computational Thinking qualification, is *entirely* my own work and has not been taken from the work of others - save and to the extent that such work has been cited and acknowledged within the text of my work.

I hereby acknowledge and accept that this thesis may be distributed to future final year students, as an example of the standard expected of final year projects.

Signed: Dean Connell Date: 23/02/2021

Acknowledgements

I would like to thank my project supervisor, Joseph Duffin, for allowing me to take part in this investigation; for patiently and diligently answering any queries regarding the technical content of the project and for collecting all the data necessary to conduct this study. I would also like to thank him for his availability and dedication throughout this project, in this year of adversity.

I express my sincerest gratitude to the stroke patients who participated in the cognitive tasks required for the generation of this data. Without their cooperation, this project would not have been possible.

I would also like to thank my fellow student Jack Eaton Kilgallen, who was working in parallel with me on this project, for discussing issues and possible approaches to problems with me and for providing insight into alternative approaches to EEG analysis for the duration of this project.

I would like to sincerely thank past students Karl Crofton and Sharon John, who completed projects with the same data last year. Their attention to detail in documenting the entirety of their work was an incredibly useful resource in my research.

I would finally like to thank my own mother, whose resilience in recovering from a transient ischemic attack prompted and inspired me to undertake this project.

Thank you all.

Abstract

MRI and CT scans have been widely used to aid in the diagnosis of strokes. However, if it can be shown that we can also use EEG (electroencephalogram) data in stroke identification, it would be a faster, cheaper and less invasive alternative to the aforementioned scans. EEG data has been taken from both a group of stroke patients and a control group, while they performed a set of tasks that would elicit cognitive responses, namely Visual Paired Associates and Visual Search. These were performed by each group at two separate times: T_1, in the days immediately following the stroke and T_2, 6 months post-stroke. To attain a comparative metric, the data was imported into EEGLab (a MATLAB plugin) and manipulated using signal averaging to extract the ERPs (Event Related Potentials) for each stimulus in the task. Periodograms were also run to estimate the Power Spectral Density of each ERP signal. This data was read by a Python script and several machine learning techniques were trialled to classify stroke and control data. Support Vector Classification proved most useful in showing that control and stroke EEG data cluster separately. When used in conjunction with the Visual Paired Associates task, it admitted a mean accuracy of 83.93% in classifying EEG data into stroke and control groups. At the very least, we can conclude that Support Vector Classification is the most accurate technique studied in stroke classification with the metric attained.

List of Figures

List of Tables

Chapter one: Introduction

Summary

This is a brief description of the topic of the viability of using EEG data to determine whether a person has suffered a stroke or not. This details the motive behind the project, the technical side to the problem, the steps necessary to acquire a solution to this problem, how the work will be evaluated and finally a brief summary of the goals achieved throughout this project.

1.1 Topic addressed in this project

This project deals with the manipulation of raw electroencephalogram (EEG) data into a format, which can be classified into stroke and control groups. This data has been obtained from a group of stroke patients in a medical setting and a group of neurotypical control participants. The enhancement of an existing pipeline which takes this raw data and produces a metric, which is used to distinguish between a healthy brain and one which has suffered the effects of a stroke, is paramount in the attempt to discover an EEG biomarker separating the two groups.

This project will address the use of machine learning techniques such as clustering and classification, based on this metric extracted from the EEG data, to determine a method of accurately identifying brains which have been neuro-compromised due to stroke.

As will be detailed throughout this thesis, topics addressed in this project include signal processing, machine learning, data visualisation and data analysis.

1.2 Motivation

Currently, the use of medical imaging such as Computerised Tomography (CT) and Magnetic Resonance Imaging (MRI) are widely recognised for their performance in recording enough information for specialists to determine if a stroke has occurred. However, these scans can be invasive, expensive and slow; the results take time to be processed, needing a specialist to read the scan accurately. Time is an extremely important factor in diagnosis of a stroke, as symptoms can exacerbate quickly, and brain tissue infarction can occur leading to complications. Given that one in five people will have a stroke at some point their life (Irish Heart, 2020), it would be a significant and desirable breakthrough if a cheaper and faster method of stroke identification were to be accomplished.

A possible method that would work as described would be the use of EEG data coupled with analysis through machine learning. Research in this area has already been carried out (Wu et al., 2016, pp. 2399-2405) in which it is seen that EEG scans can provide insight into the effects of acute stroke. There must be some attainable metric, which we can find, such that we can distinguish between these cognitive responses of a stroke and healthy patient based on the power of the measured responses at particular electrodes from the two groups.

1.3 Problem statement

The technical problem addressed in this project is the production of modular and cohesive pipeline that takes raw EEG data recorded on a BioSemi 32-electrode cap and produces a metric or comparative standard that is used to differentiate between a stroke patient and a participant that has not suffered a stroke. The data we are given has been pre-processed by the BESA (Brain Electrical Source Analysis) proprietary software (*BESA*, 2021) and therefore the goal of our pipeline is to extract an explanatory metric from this cleaned data and use it to contrast control and stroke EEG data.

The comparison of the two experimental groups will be performed using machine learning in Python and will determine the preferred algorithm for comparing EEG data with this metric for each cognitive task in each experimental group. The machine learning aspect of this pipeline is a necessity, as many of the peculiarities of EEG data analysis are not visible to the naked eye and therefore cannot be categorised simply by a specialist at current.

1.4 Approach

In my analysis of this problem, extensive literary research was conducted in the technical aspects of this project. I researched different types of stroke, the neurophysiological basis of EEG (Olejniczak, 2006, pp. 186-189), event related potentials, the software I would be using, the BESA workflow and methods of classifying data in Python. Once I was acquainted with the technicalities of the project, I decided to focus on the data recorded while participants performed the Visual Paired Associates and Visual Search tasks. I familiarised myself with the way these tasks were conducted to generate the EEG data required and noted their associated trigger values.

The raw data was recorded on a BioSemi 32 electrode cap and pre-processed in BESA before I received it. At this stage, the data was filtered; with the high frequency set to 30Hz, the low pass set to 0.3Hz and the notch filter set to 50Hz to remove electrical mains interference. Ocular artefacts were removed automatically by using a spatial filter technique, and other artefacts were then identified and marked. Bad electrode channels were also identified and if possible, interpolated from other proximal electrodes. This cleaned data was then exported in EDF format.

This EDF data was loaded into MATLAB via the EEGLab plugin. However, the EDF data I received was incompatible with EEGLab, so I was required to complete some processing steps (which are discussed in section 4.2) on these files before it could be analysed. After successfully loading the data into EEGLab, the EEG signals were plotted to identify the electrode channels with the strongest responses after each trigger. I then used MATLAB scripts to generate ERPs for each trigger and estimated power spectra, exporting this information as a .mat file. Finally, I imported these .mat files into a Python script, which was used to apply machine learning techniques such as partial least squares discriminant analysis (PLS-DA) and support vector classifiers, along with many other methods. This is done to separate and distinguish between control and stroke groups according to the metric obtained and identify the most accurate classification algorithm from the set studied.

1.5 Metrics

Technically, the metric this project strives towards is whether a biomarker is found or not. As I am not responsible for the development of the MATLAB scripts, which generate the comparative metric, the evaluation of this particular project lies in the identification of the best method of classification, which enhances the existing pipeline.

As a result, in evaluating the success of this project, we consider the analysis of metrics generated by the machine learning techniques outlined above. For example, I will calculate the accuracy of each classification and generate confusion matrices. I will then compare the accuracies of each method and identify which method is the best predictor of whether a particular EEG input belongs in the stroke group or not.

1.6 Project

I have achieved several significant milestones throughout this project. I have identified the steps necessary to convert the data output by the BESA pre-processing part of the pipeline into a workable EDF format, which can be processed by EEGLab. I have added these steps to the pipeline, which would be useful for future developments with this data.

I have also successfully reproduced the signal processing results of two students who have undertaken this project previously, by modifying their MATLAB code and omitting redundant scripts. Because of this, I could perform statistical analysis on both the Visual Paired Associates and Visual Search tasks in a realistic timeframe.

Finally, I have identified Support Vector Classification as the most accurate machine learning technique based on the metric obtained from running the MATLAB scripts. This method shows a mean accuracy of 83.93% when classifying EEG signals collected from the Visual Paired Associates task. While this may not determine a distinguishing biomarker, it lays the foundation in ERP analysis that this statistical method may prove useful with advancements in EEG technologies.

Chapter two: Technical Background

Summary

The technical nature of this project requires that we discuss a wide range of professional areas, as this combines the distinguished disciplines of neurophysiology, data analysis and signal processing. I will discuss the definition of a stroke, different types of stroke, electroencephalograms and their relevance in determining strokes, event related potentials, signal processing, machine learning and a brief background on the data provided.

2.1 Topic material

2.1.1 What is a stroke?

A stroke is most simply described as a rupture or a blockage of a blood vessel, which causes an interruption of blood supply to the brain (Irish Heart, 2020). Depending on factors such as the severity of the stroke, the time before stroke symptoms are treated and the location of the rupture or blockage, brain cells can be damaged or destroyed. This can affect several cognitive and motor functions for example speech, memory and co-ordination (Al-Qazzaz et al., 2014, p. 1677).

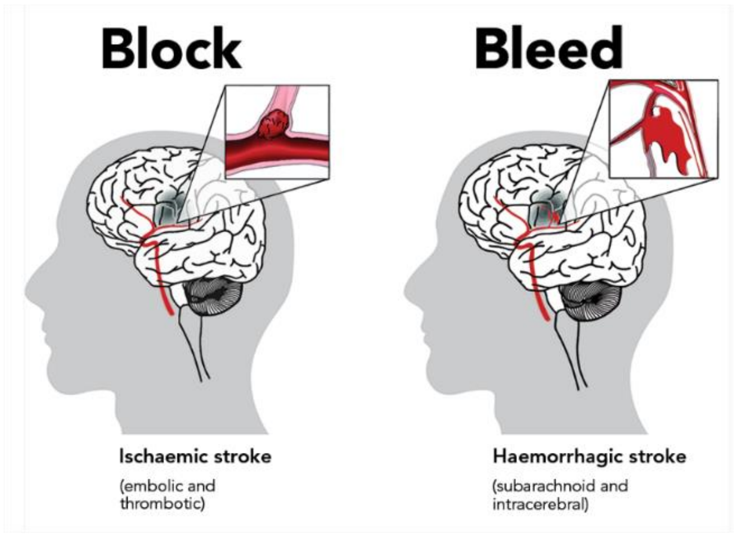


Figure 2.1: Ischaemic and haemorrhagic stroke. (https://enableme.org.au/Resources/Types-of-stroke)

Strokes can be classified into three main categories (American Stroke Association, n.d.): transient ischaemic attacks, ischaemic strokes and haemorrhagic strokes.

Transient Ischaemic Attacks (TIAs), also known as mini-strokes, are defined as stroke signs and symptoms which subside within 24 hours, but usually within minutes. These are caused by small clots, which temporarily block the blood vessel, interrupting blood supply to the brain.

Ischemic strokes are caused by in situ thrombosis (local coagulation or clotting) or distal embolisms because of cardioembolic or atherosclerotic arterial diseases, such as a build-up of plaque in blood vessels (as seen on the left in Figure 2.1).

Haemorrhagic strokes are caused by the rupture of a blood vessel (as seen on the right in Figure 2.1). These can be subdivided into two main groups: intracerebral and subarachnoid. Intracerebral haemorrhages are ruptures of small penetrating vessels in the brain, causing blood to seep into brain tissue. Subarachnoid haemorrhages are ruptures of intracranial aneurysms between the arachnoid membrane and the pia mater.

2.2 Technical material

2.2.1 Electroencephalograms (EEGs)

An electroencephalogram, more commonly known as an EEG, is a graphical representation of the difference in voltage between two cerebral locations, plotted over time. The scalp EEG signal is generated by the firing of cerebral neurons, which is amplified by the conductive properties of brain tissue between the electrical source and the electrode on the EEG apparatus (Olejniczak, 2006, pp. 186-189). EEG signals are recorded using an electrode cap, with the application of conductive gel to aid in fixing each electrode to a particular position on the scalp.

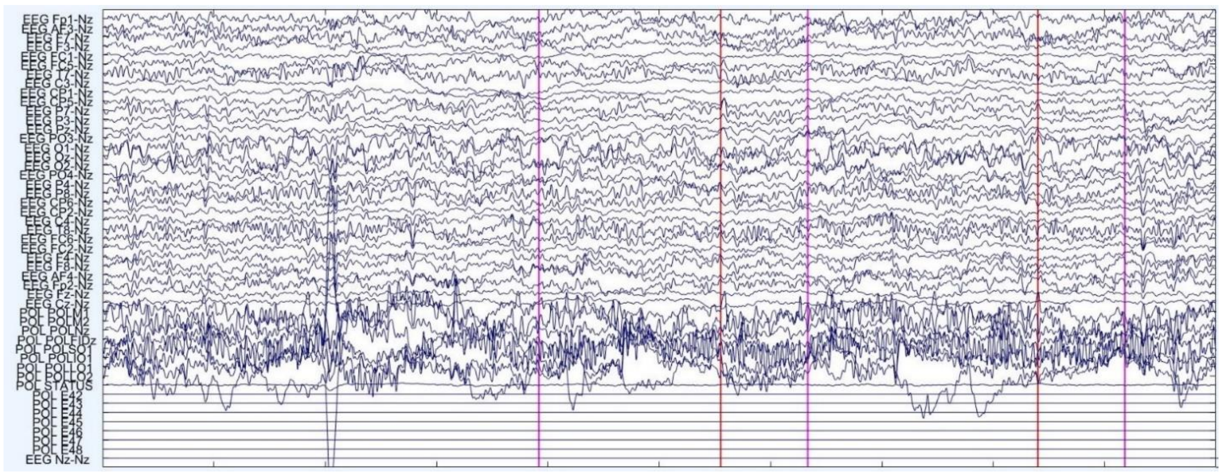


Figure 2.2: Example of an EEG signal from EEGLab – Participant 14 for T_1 of Visual Paired Associates.

According to research conducted (Wu et al., 2016, pp. 2399-2405), there is evidence to suggest that the use of EEG data alone - without CT or MRI scans - can determine whether a stroke has occurred or not. This study claims that EEG can provide insight into the effects of acute stroke particularly, establishing validity for measuring altered brain activity in this setting. With recent advancements in EEG technology, we can achieve more accurate EEG readings, which means that the possibility of EEG alone identifying all types of stroke becomes more realistic with time. It is important to note that a 256-electrode cap was used in this study, which is significantly denser than the 32-electrode cap used in gathering the data for this project. This implies that the data I will analyse has a lower spatial resolution and may not have recorded subtle changes in brain activity.

2.2.2 Event Related Potentials (ERPs)

Event related potentials are roughly defined as the changes in an ongoing EEG due to cognitive stimulation. They have low amplitude compared to background EEG activity and are therefore hardly visualised in single trials, or in our case, as single responses to a trigger. Instead, event related potentials are the average evoked responses over several trials (Quiroga & Garcia, 2003, pp. 376-390).

The use of signal averaging to realise an event related potential creates an invariant or characteristic signal for the particular response in question. While this proves useful in gathering a participant's typical response for a particular stimulus, the trouble with ERPs lies in the potential loss of vital micro-information as a result of this signal averaging.

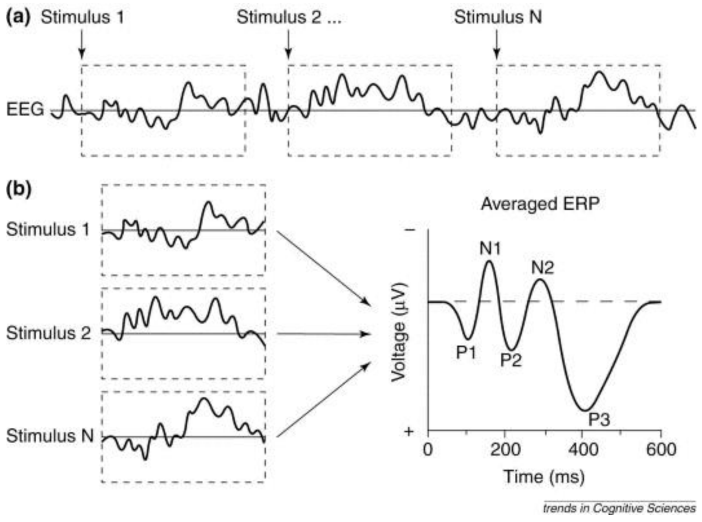


Figure 2.3: The generation of an event related potential. (Steven J. Luck, Geoffrey F. Woodman, Edward K. Vogel, Event-related potential studies of attention, Trends in Cognitive Sciences, Volume 4, Issue 11, 2000)

However, in my literary review of event related potentials, I discovered that they can be used to determine if brain infarction has occurred resulting from stroke. According to this particular study (Korpelainen et al., 2000, p. 202-208) brain infarction prolonged the P300 (also known as P3) latency but did not affect the P300 amplitude or distribution on the scalp. This indicates that the difference in latency of reaction to a stimulus may be important in distinguishing stroke and control ERPs. This study also found that the correlation between age and P300 latency was absent at a time 3 months post-stroke, however this correlation was present at 12 months post stroke. This suggests that there is a significant relationship between stroke and P300 latency, which may prove useful in determining the viability of ERPs as metrics used to classify stroke EEG data.

2.2.3 Signal Processing

To understand the complete pipeline and the data manipulation required in this project, we must be familiarised with the extensive topic of signal processing. According to Fourier analysis, any signal may be expressed as a sum of sinusoidal functions. Analysing EEG signals through Fourier transforms such as these will prove beneficial in allowing us to view the frequencies of EEG components, which make up the EEG signal. Using the signal processing toolbox for MATLAB, we can run periodograms which are similar to Fourier transforms on this EEG data. These periodograms are used to estimate the power spectral density of each event related potential.

2.2.4 Machine Learning

As mentioned previously, machine learning techniques will be used to classify the EEG data in the hopes of discovering a biomarker. I will choose to focus on five techniques: partial least squares discriminant analysis, support vector classification, linear discriminant analysis, logistic regression and multilayer perceptron neural networks. Three other techniques were considered but did not produce meaningful results. These are described in detail in Appendix 1.

Partial least squares discriminant analysis is a method of constructing predictive models based on projecting the predicted and observable variables to a new space with a lesser dimension (Ruiz-Perez et al., 2020, p. 1). According to this study (Wu et al., 2016, pp. 2399-2405), partial least squares regression proved to be beneficial when used in tandem with EEG data in stroke classification, so my use of discriminant analysis for classification purposes could prove beneficial.

Support vector classifiers (also known as support vector machines) describe computer algorithms, which learn by example to assign labels to objects in a dataset (Noble, 2006, pp. 1565-1567). They work by maximising a particular mathematical function with respect to a given collection of data. A common application of support vector machines is the automatic classification of microarray gene expression profiles, but they can be used for many types of binary data classification.

Linear discriminant analysis is a technique employed for projecting variables to a lower dimension (dimensionality reduction) and classification. LDA sketches a decision boundary between the different classes which provides a basis of separation (Mohanty et al., 2013, pp. 245-267). This works much like PLS discriminant analysis in this way, however linear regression is used to calculate the decision boundary rather than PLS regression.

Logistic regression is a method widely used in binary classification where a linear combination of explanatory variables are mapped to the interval (0,1) using the sigmoid function (Urso et al., 2019, pp. 413-430). Outputs of this function are classified according to their value. If the output is greater than 0.5, it is classified in group 1. Otherwise, the output is classified in group 0. These are often compared with support vector machines in and generally perform comparably in practice, so it is important to include this classification method in our analysis.

Multilayer perceptron (MLP) neural networks are three layered networks of nodes; called the input, output and hidden layers. The input layer receives the data to be processed and classification is performed by the output layer in this case. The hidden layers perform computation to translate the input data into the classified output, using different synaptic weights to attribute a value to each artificial neuron (Abirami & Chitra, 2020, pp. 339-368). MLPs are generally used to solve problems that are not linearly separable, therefore it is prudent to include them in the set of analysed techniques.

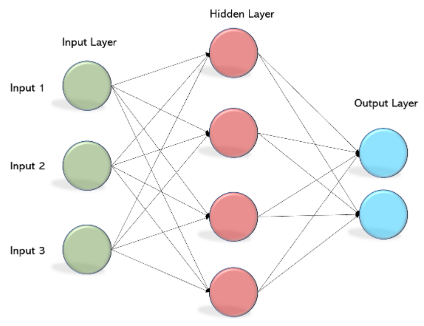


Figure 2.4: Schematic representation of a multilayer perceptron neural network. (https://becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data-f6dd3d7e998f)

2.2.5 Cognitive Task Structure

In the collection of this EEG data, two cognitive tasks were performed: Visual Paired Associates and Visual Search. These were performed by each experimental group at two separate times: T_1, in the days immediately following the stroke and T_2, 6 months post-stroke.

Visual Paired Associates is conducted in two phases: a study phase and the test phase. In the study phase, participants study pairs of stimuli presented before a background. They are asked to remember just the stimulus pairs and not to be concerned about the background.

In the test phase, participants are issued with pairs of stimuli once again and asked to identify whether the pair was observed or not. The following table details all correct response types.

Table 1 *Response types for the Visual Paired Associates task.*

Observed pair with observed background	True Congruent Correct (TCC)
Observed pair with unseen background	True Incongruent Correct (TIC)
Unseen pair with observed background	False Congruent Correct (FCC)
Unseen pair with unseen background	False Incongruent Correct (FIC)

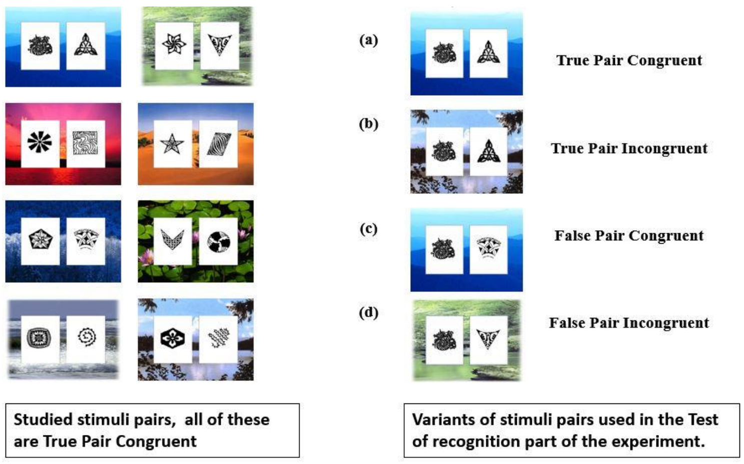

(a)
True Pair Congruent
(b)
True Pair Incongruent
(c)
False Pair Congruent
(d)
False Pair Incongruent
Studied stimuli pairs, all of these are True Pair Congruent
Variants of stimuli pairs used in the Test of recognition part of the experiment.

Figure 2.5: Visual Representation of the stimuli used in the Visual Paired Associates task.

Visual Search is conducted in just one test phase. There are two blocks of trials in which images appear on screen and the participant is asked to identify whether a red forward slash can be seen or not in each. The following table details all correct response types for this task.

Table 2 *Response types for the Visual Search task.*

Stimulus present in far left segment of image	Far Left Correct (FLC)
Stimulus present in left segment of image	Left Correct (LC)
Stimulus present in right segment of image	Right Correct (RC)
Stimulus present in far right segment of image	Far Right Correct (FRC)
Stimulus not present in image	Absent Correct (AC)

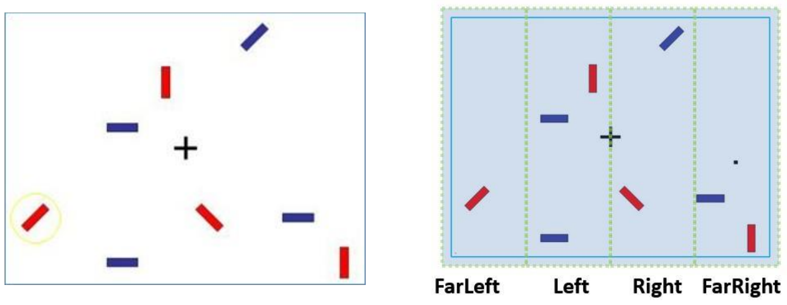


Figure 2.6: Visual Representation of the stimuli used Visual Search task. The stimulus is present in the far left segment of this example.

Chapter three: The Problem

Summary

The purpose of this chapter is to describe the problems involved in manipulating EEG data into a format that can be used as a classification metric. This chapter will also detail the issues encountered in the assessment of various statistical machine learning methods with this metric to identify the most accurate technique.

Our problem statement can be summarised as follows: given cleaned EEG data and an existing pipeline, examine and modify the pipeline such that the most accurate machine learning classification technique, given a particular metric, can be identified from methods considered. The issues faced in this project included manipulating the EDF data given into workable format for the pipeline and attempting to discover a clear biomarker separating the stroke and control data.

3.1 Problem analysis

As I have previously discussed, stroke tends to affect cognition and motor function in numerous ways depending on the severity and length of the interruption of blood supply to the brain. Because EEG is a recognised measurement of brain activity, a biomarker for stroke identification is rationally likely to exist in the form of EEG waves in some format. The problem addressed by this project lies in the discovery and extraction of such a biomarker and the viability of using this to quickly identify the presence of stroke in electroencephalograms. I believe that this may be done meticulously by using a machine learning algorithm, as identifying such a characteristic in the intricacies of an EEG wave by eye would be extremely difficult.

Electroencephalogram data was gathered using the BioSemi-32 apparatus, which comprises of a 32-electrode cap equipped with the 10/20 international electrode system. The signal is recorded by collecting electrical impulses at the scalp of the participant while performing a computer-based cognitive task.

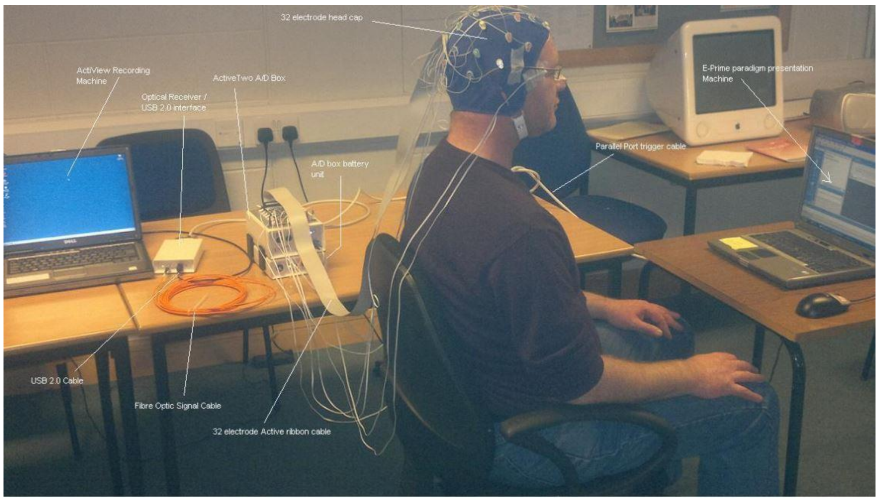

32 electrode head cap
ActiView Recording Machine
ActiveTwo A/D Box
Optical Receiver / USB 2.0 interface
E-Prime paradigm presentation Machine
Parallel Port trigger cable
A/D box battery unit
USB 2.0 Cable
Fibre Optic Signal Cable
32 electrode Active ribbon cable

Figure 3.1: BioSemi 32-electrode cap and setup of the cognitive experiments. (Courtesy: Joseph Duffin)

The data that I receive to analyse has undergone pre-processing using the BESA proprietary software by my project supervisor. Raw EEG data has been converted from BDF (binary data format) to EDF (European data format) at this stage. The largest problem encountered in this project is with this EDF data, as that it needs to undergo further file conversions to be translated into a format, which is useful in the given pipelines.

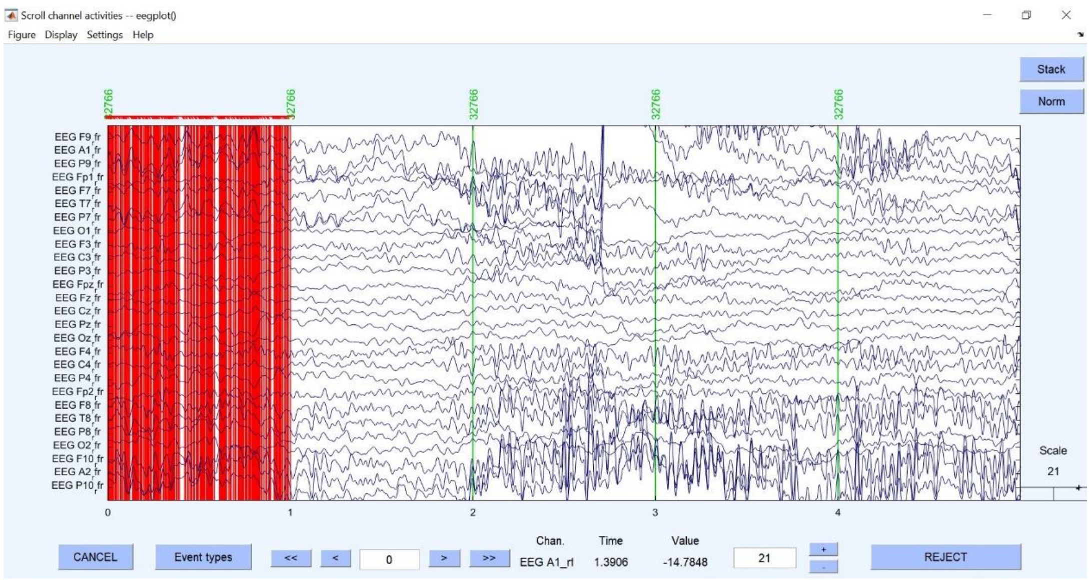


Figure 3.2: EDF data formatting error viewed in EEGLab prior to necessary file conversion.

I have also received two separate pipeline components used to attain a metric – one metric generating component for each task. These components have been constructed on MATLAB and require the use of structs obtained from using the open-source library, EEGLab (*What Is EEGLAB?*, 2021). The MATLAB code in these received pipeline components has already been developed and thus the only problems encountered at this stage are the understanding of the code, the removal of redundant scripts and the editing of parts of the code to ensure each file is generated correctly after the removal of these scripts.

The main problem addressed in this project is the classification of the data according to the metric attained. Extensive research has been conducted into choosing suitable machine learning algorithms for classifying the data given into control and stroke groups, so the overarching problem approached in this thesis is the identification of the most favourable method of classification given the metric obtained. This statistical analysis is performed by calculating the accuracy score for each algorithm and this will be used to assess the utility of the algorithm in each case.

As a result, another problem addressed in this project is the comparison of the Visual Paired Associates and Visual Search tasks regarding their benefit in evoking responses that distinguish between regular and stroke-compromised brains. This will involve using my findings from the statistical analysis to gain an overall assessment of each task and contrasting these scores.

Chapter four: The Solution

Summary

This chapter outlines the process in achieving solutions to the problems discussed in the previous chapter. It will discuss the logic determining the decisions I have made regarding elements of the project. It will also provide UML diagrams detailing the entire process from the recording of the data up to the determination of the most beneficial classification algorithm. It will explore each component of the pipeline and explain the steps taken in each component. Finally, it will detail each MATLAB and Python script used in the pipeline that is used to develop solutions to the problem statement. It is to be noted that the comparison of machine learning techniques will be detailed fully in the following chapter – evaluation, and that this chapter is concerned more with the steps taken to produce a classification metric.

4.1 Architectural Level

The steps required to reach a solution to our problems are detailed by a cohesive pipeline structure which classifies these steps into four main components: pre-processing (which has been performed by my supervisor), metric pre-processing, metric attainment and statistical analysis. The specific tasks attributed to each pipeline component are summarised in the UML diagram below.

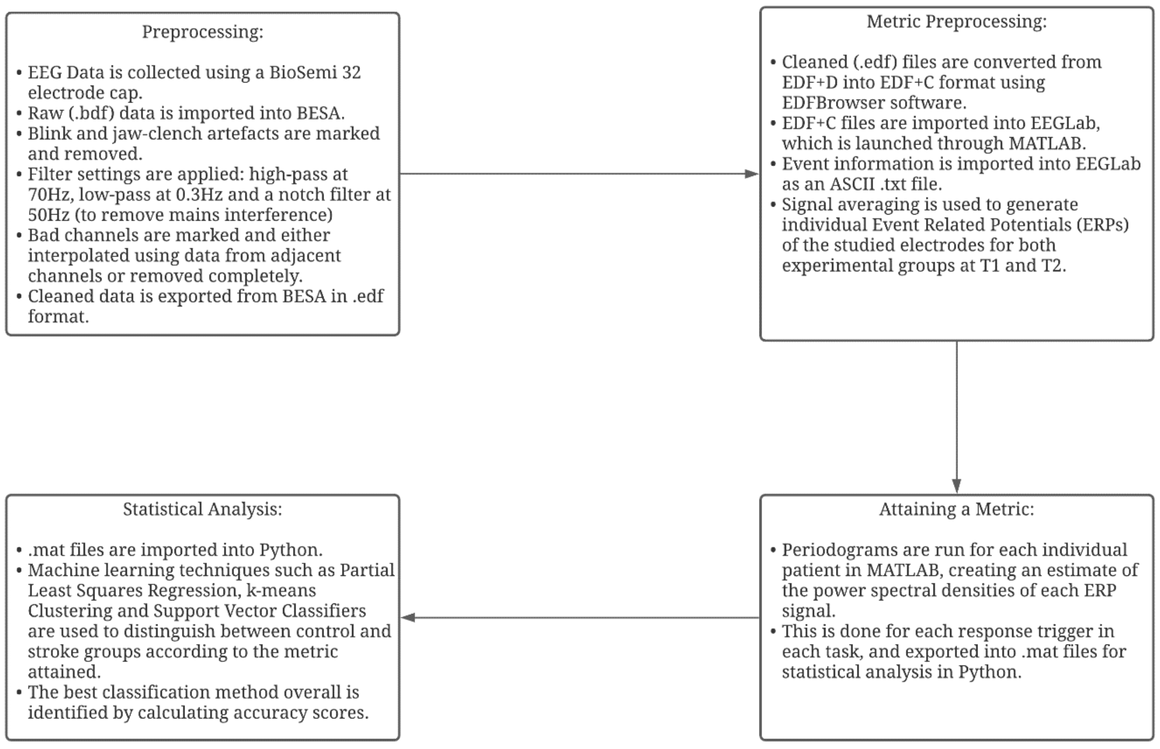

Preprocessing:
• EEG Data is collected using a BioSemi 32 electrode cap.
• Raw (.bdf) data is imported into BESA.
• Blink and jaw-clench artefacts are marked and removed.
• Filter settings are applied: high-pass at 70Hz, low-pass at 0.3Hz and a notch filter at 50Hz (to remove mains interference)
• Bad channels are marked and either interpolated using data from adjacent channels or removed completely.
• Cleaned data is exported from BESA in .edf format.
Metric Preprocessing:
• Cleaned (.edf) files are converted from EDF+D into EDF+C format using EDFBrowser software.
• EDF+C files are imported into EEGLab, which is launched through MATLAB.
• Event information is imported into EEGLab as an ASCII .txt file.
• Signal averaging is used to generate individual Event Related Potentials (ERPs) of the studied electrodes for both experimental groups at T1 and T2.
Attaining a Metric:
• Periodograms are run for each individual patient in MATLAB, creating an estimate of the power spectral densities of each ERP signal.
• This is done for each response trigger in each task, and exported into .mat files for statistical analysis in Python.
Statistical Analysis:
• .mat files are imported into Python.
• Machine learning techniques such as Partial Least Squares Regression, k-means Clustering and Support Vector Classifiers are used to distinguish between control and stroke groups according to the metric attained.
• The best classification method overall is identified by calculating accuracy scores.

Figure 4.1: UML Diagram of the entire pipeline, divided into four main components.

The idea of this pipeline is simple, raw EEG data is recorded and is processed through these four components to produce a metric. This metric is then used in conjunction with a number of machine learning algorithms to identify the most accurate classification method overall for both tasks. A key advantage in working with a pipeline as described is the ease in using it to generate data for use in the statistical analysis, which I have chosen to focus on. In a project working with intricate raw data, it is important to have established a pipeline that will manipulate the data into a form that can easily be interpreted. Therefore, the documentation of the steps taken to achieve this data formatting is paramount in future investigation of data of this type.

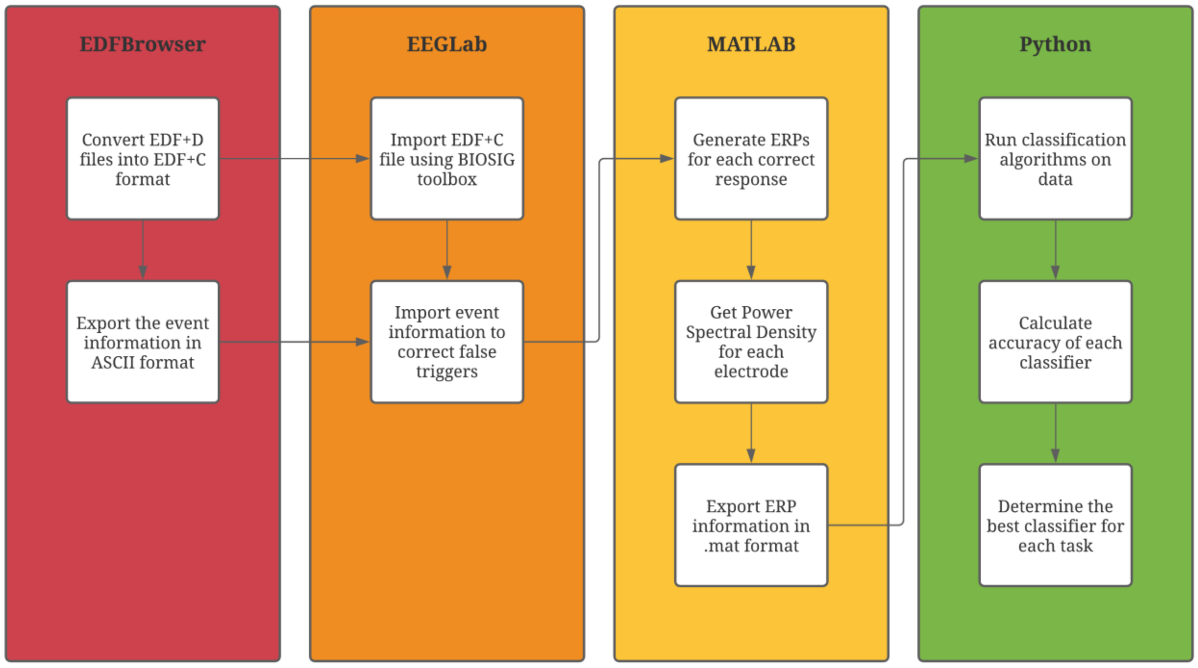

EDFBrowser
Convert EDF+D files into EDF+C format
Export the event information in ASCII format
EEGLab
Import EDF+C file using BIOSIG toolbox
Import event information to correct false triggers
MATLAB
Generate ERPs for each correct response
Get Power Spectral Density for each electrode
Export ERP information in .mat format
Python
Run classification algorithms on data
Calculate accuracy of each classifier
Determine the best classifier for each task

Figure 4.2: Technical specification of the pipeline along with the software used at each stage.

The figure above details the technical specification of the pipeline, this time divided into blocks according to the software or technology I required at each stage in the process.

EDFBrowser (*EDFBrowser*, 2021) was required to convert the corrupted EDF files as seen in Figure 3.2 into a format that could be read by EEGLab correctly and was selected due to the wide range of tools it offered in the conversion and modification of EEG data in EDF format.

EEGLab - and consequentially, MATLAB - was chosen to extract the information needed from EEG data to be used in attaining a metric for three reasons. Firstly, there is a wealth of documentation relating to EEGLab, such as the EEGLab Wiki (*EEGLab Wiki*, n.d.). This provides great support when issues are encountered using the software. Secondly, EEGLab has an intuitive graphical user interface which proves to be simple to use, saving time for statistical analysis. Finally, MATLAB scripts were made available to me from a previous implementation of a similar pipeline, so this allowed me to invest feasible time into the goal of this thesis – the comparison of classification techniques.

Python was chosen to perform the statistical analysis portion of this project, thanks to the wealth of libraries available to aid statistical machine learning such as NumPy, SciPy and Scikit-learn. I briefly considered using R for this analysis, however I believe with the use of the aforementioned packages, Python is more flexible, attractive and simple to use in the endeavours of this project.

4.2 Analytical Work – Pipeline Processes

A description of the solution implemented is best conveyed using Figure 4.1 as a guide. I will discuss each component of the pipeline and the analytical work required in order to attain the metric used in the studied classification algorithms.

4.2.1 Pre-processing

This step of the pipeline is completed by my supervisor before I receive the data. It is important in the overall conversion of raw EEG data into a clean format, which is necessary for accuracy in our analysis.

The raw BDF data which has been collected during the experiment is imported into BESA. The BDF data is exported into .foc format and surface point coordinate files are loaded into BESA on the .foc file. Ocular and jaw clench artefacts are marked and subsequently removed using spatial filtering techniques; for example, the Berg and Scherg algorithm to remove EEG interference caused by blinking (Berg & Scherg, 1994, pp. 229-241).

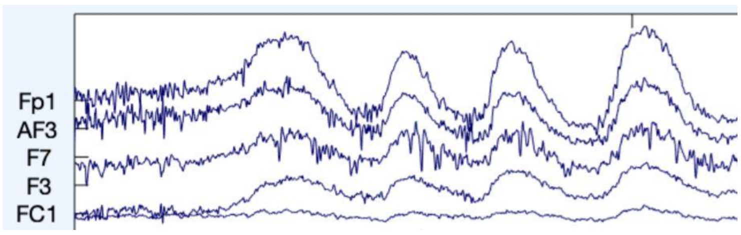


Figure 4.3: Blink artefact sample from a BDF file, characterised by large inverted waves in the signal.

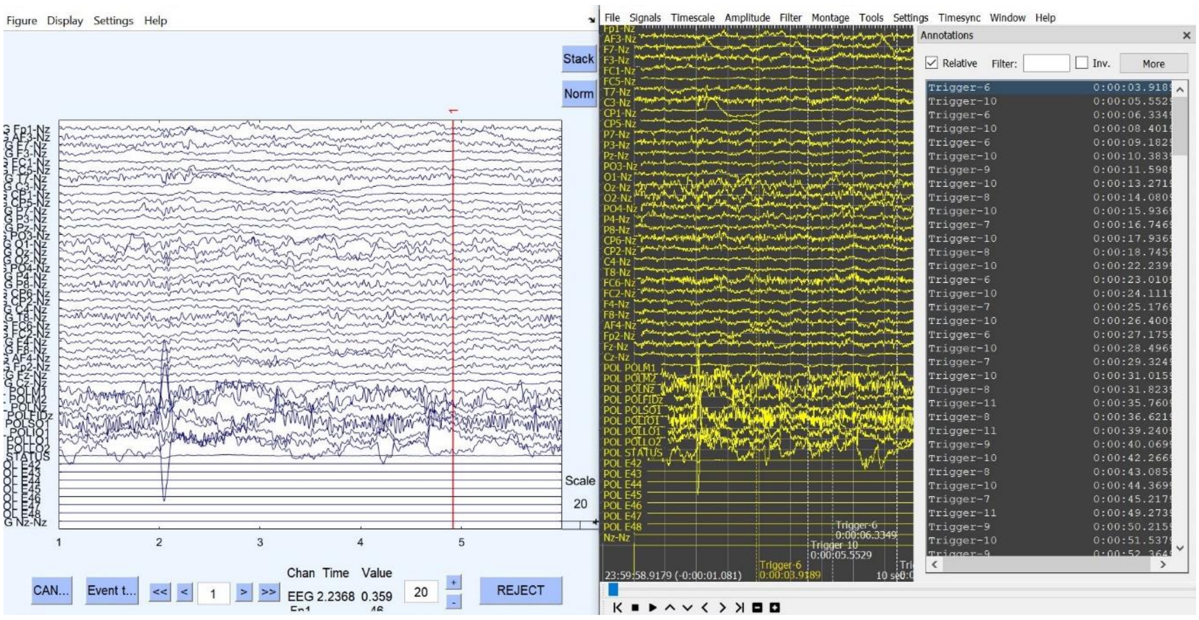
Signal filters are also applied to remove any noise that may interfere with the EEG signal. A bandpass filter which passes frequencies of 0.3Hz - 70Hz is applied, along with a notch filter of 50Hz to remove mains interference proximal to the electrode cap. Following this, extra unnecessary channels are marked and removed and bad channels are identified. These bad channels are marked and either interpolated by proximal neighbouring electrodes or else removed completely. Finally, this cleaned file is exported from BESA into an EDF format. I receive the data after the processing described, in this particular EDF format.

4.2.2 Metric Pre-processing

My particular approach to the solution begins here. The EDF files that I have received are not formatted correctly for use in conjunction with EEGLab, my chosen software. The reason for this incorrect formatting is unknown, however I theorise that it is relating to discrepancies between the formatting of event information between BESA and EEGLab. When the EEGLab files I have received are loaded into EEGLab, the formatting error described in Figure 3.2 occurs.

Therefore, it is necessary to convert these EDF files into a workable format. The action required was identified as an EDF+D (discontinuous EDF file) to EDF+C (continuous EDF file) conversion, which could be performed using a software called EDFBrowser. I discovered at this part of the process that many of the EDF files provided could not be converted. In order to continue with the project, I could only work with the subset of the files which could be converted.

This conversion allows the EDF file to be displayed with events at the correct timestamps in EEGLab. However, these event triggers are incorrectly labelled when compared to the file open in EDFBrowser as seen below.



```
Trigger-6     0:00:03.918
Trigger-10    0:00:05.552
Trigger-6     0:00:06.334
Trigger-10    0:00:08.401
Trigger-6     0:00:09.182
Trigger-10    0:00:10.383
Trigger-9     0:00:11.598
Trigger-10    0:00:13.271
Trigger-8     0:00:14.080
Trigger-10    0:00:15.936
Trigger-7     0:00:16.746
Trigger-10    0:00:17.936
Trigger-8     0:00:18.745
Trigger-10    0:00:22.239
Trigger-6     0:00:23.010
Trigger-10    0:00:24.111
Trigger-7     0:00:25.176
Trigger-10    0:00:26.400
Trigger-6     0:00:27.175
Trigger-10    0:00:28.496
Trigger-7     0:00:29.324
Trigger-10    0:00:31.015
Trigger-8     0:00:31.823
Trigger-11    0:00:35.760
Trigger-8     0:00:36.621
Trigger-11    0:00:39.240
Trigger-9     0:00:40.069
Trigger-10    0:00:42.266
Trigger-8     0:00:43.085
Trigger-10    0:00:44.369
Trigger-7     0:00:45.217
Trigger-11    0:00:49.273
Trigger-9     0:00:50.215
Trigger-10    0:00:51.537
```

Figure 4.4: Discrepancies between trigger values in EEGLab (left) and EDFBrowser (right).

This issue is rectified by the exportation of an annotated event file in text format, which can also be performed on each EDF+C file in EDFBrowser. When this ASCII text file is imported into EEGLab along with its associated EDF+C file, this incorrect labelling of triggers is rectified and the discrepancies between trigger values in EEGLab and EDFBrowser are resolved.

When this necessary conversion is complete and both elements are imported into EEGLab, we can identify the electrodes which generate the highest amplitude in response to the stimuli in each task. After consulting the previous work of my supervisor (J. Duffin, personal communication, March 18, 2021), I decided to focus on the PO3 and PO4 electrodes for the Visual Paired Associates task and alternatively, the P7 and P8 electrodes for the Visual Search task.

Once the electrodes have been chosen, we can use the trigger information to epoch around the correct responses in each file to collect the information necessary to generate ERPs. This is done using the getVPAERP.m and getVSERP.m scripts explained in section 4.3. The following table explains the classification of each trigger number in each task, which these files use to identify the correct responses to epoch (by looking at the trigger value of the event before trigger value 10 in each case).

Table 3 *Trigger values and associated meanings for each task.*

Visual Paired Associates		**Visual Search**	
4	False Congruent Pair (FC)	1	Far Left Stimulus (FL)
5	False Incongruent Pair (FI)	2	Left Stimulus (L)
6	True Congruent Pair (TC)	3	Right Stimulus (R)
7	True Incongruent Pair (TI)	4	Far Right Stimulus (FR)
10	Correct	10	Correct
11	Error	11	Error
		128	Stimulus Absent (A)

When the signals around the triggers corresponding to correct responses are epoched according to a fixed timescale (between 200ms before and 2500ms after the event), these signals are averaged for each correct response generating the Event Related Potentials for each.

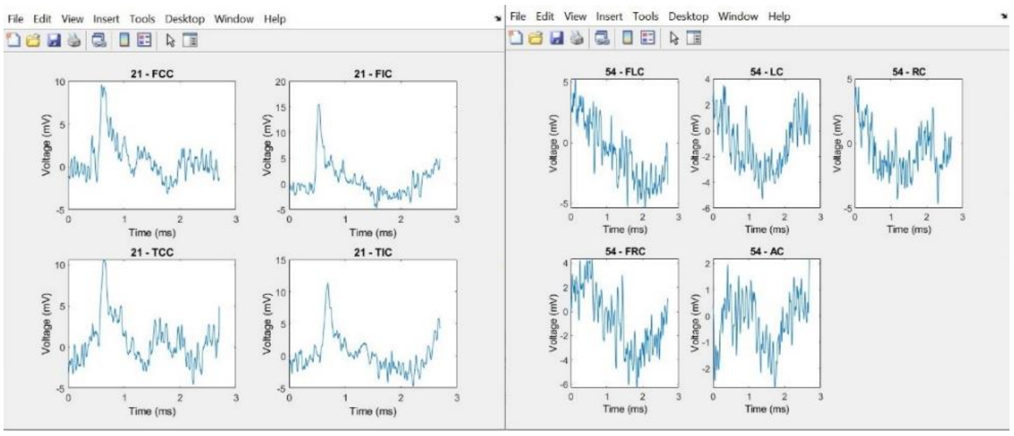


Figure 4.5: Event related potentials for PO4 in VPA task and P7 in VS task for participants 21 and 54 respectively at T_1.

4.2.3 Metric Attainment

The attainment of the metric is entirely performed in MATLAB, using the Signal Processing Toolbox. Specifically, the periodogram function is used to estimate the power spectral density (frequency vs. power of a signal) of each correct response. This periodogram function is run for all participants at the electrodes studied for each ERP generated. This data is stored in the 'patient_erp' struct, which is exported into a .mat file for statistical analysis in Python.

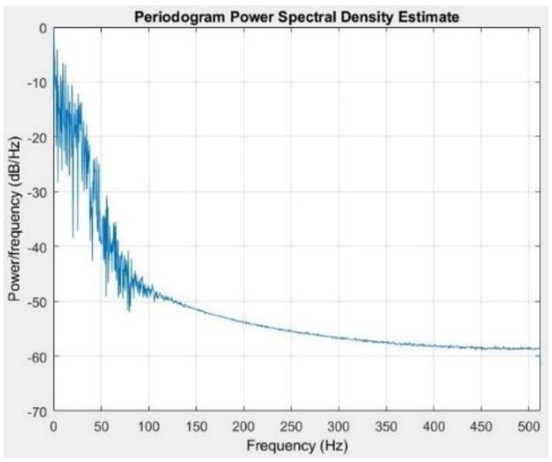


Figure 4.6: Periodogram estimating the power spectral density of the ERP generated by the FLC response in the VS task. This example is for the P7 electrode channel of participant 54 at T_2.

4.2.4 Statistical Analysis

For the statistical analysis component of this pipeline, I am running a Python script on the .mat files to generate a dataframe of the information extracted from the previous processes. Firstly, I develop a scatterplot of both groups at T_1 in each task. Here I plot the alpha band power density for each participant, based on the research conducted in this area by previous students.

I also develop similar scatterplots for both groups at T_2 in each task (which can be seen in Appendix 2). There is no clear distinction between stroke and control groups at this particular event, so I will be choosing to focus on T_1 in my analysis of classification algorithms.

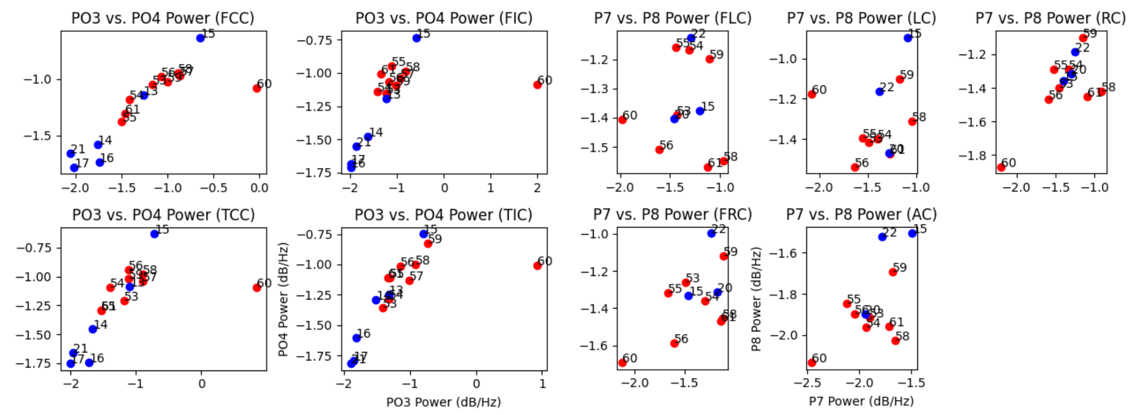


Figure 4.7: Scatterplot of stroke (red) and patient (blue) groups at T_1 for VPA (left) and VS (right).

It immediately seems that the Visual Paired Associates task provides a clearer distinction between stroke and control participants for each response, with stroke observations clustering quite tightly in each scatterplot. This distinction is not quite as clear in the Visual Search task, so further analysis will be required to determine its efficacy in this classification task.

It is also important to note the outlier observation 60 in both experiments. This is a highly influential point which may impact our results; therefore, it makes sense to investigate and determine the validity of this datapoint. I will investigate this by plotting the ERPs for participant 60.

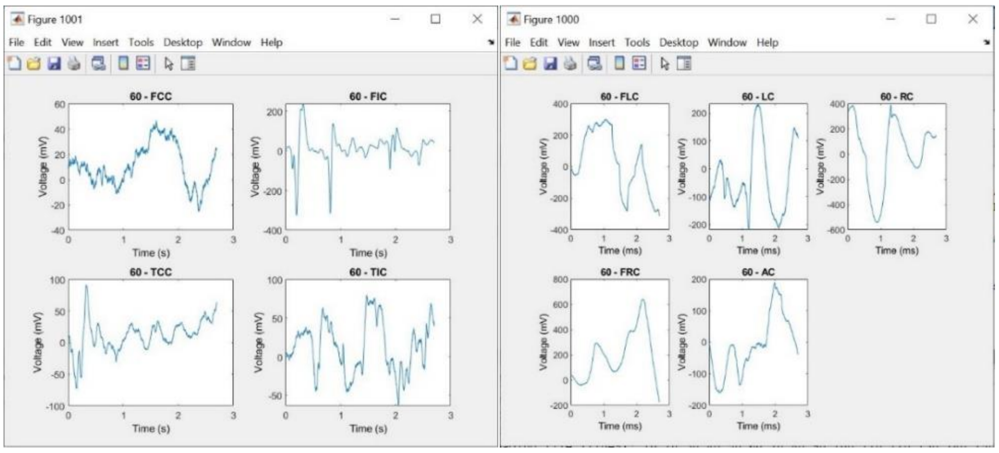


Figure 4.8: ERPs for outlying observation 60 at T_1. These plots are for PO3 in Visual Paired Associates (left) and P8 in Visual Search (right).

Evidently, the amplitude of voltage values in these ERP plots is uncharacteristic of a regular ERP. For this reason, I believe it is prudent to discard observation 60 from both datasets before we analyse each machine learning technique at T_1. Scatterplots are available (see Appendix 3) which shows the data after the removal of this observation.

Finally, a variety of machine learning algorithms are deployed on training data in an attempt to classify the stroke and control groups. PLS discriminant analysis, support vector classification, linear discriminant analysis, logistic regression and multilayer perceptron (MLP) neural networks are compared on our dataset using the leave-one-out method of determining training and testing groups. I quantify their accuracy using the accuracy_score() function in each case. This is discussed in depth in the next chapter.

It is important to note that a train-test split method using T1 data to predict T2 data was trialled, but because the T2 data was not easily separable, this method was not a success. This is discussed in detail in Appendix 4.

4.3 Implementation

The software implementation of this project is comprised of a mixture of MATLAB and Python scripts. The MATLAB scripts used are slightly altered versions of scripts used to extract a metric in a previous implementation of a similar pipeline. The original scripts have been taken from two sources, in which each source focussed on a particular cognitive task for their study. In my implementation, I have removed redundant scripts and combined similar scripts into one, resulting in fewer functional dependencies in the overall architecture of my pipeline while generating metrics for both tasks. I have developed the Python files myself as the domain I was most interested in was the statistical analysis. The resulting functions used in the pipeline are described below.

Table 4 *List of functions used in the pipeline and associated descriptions*.

Script Name	Description
VSdataSelection.m	The main file run on the imported EEG data which calls all functions specific to the VS task. Exports a .mat file containing the comparison metric used in statistical analysis.
VPAdataSelection.m	An analogous file for the Visual Paired Associates task.
getVSERP.m	Takes data, electrode channel, events, time vector and event times (in seconds) as input. Returns a struct containing ERPs of the channel for each correct VS response (FLC, LC, RC, FRC and AC)
getVPAERP.m	An analogous file for the Visual Paired Associates task.
removeVSTrigger.m	Removes the string “Trigger-” which was introduced to each trigger number by importing the ASCII event annotations into EEGLab.
removeVPATrigger.m	An analogous file for the Visual Paired Associates task, but also fixes mislabelled triggers from the VPA data collection stage.
createTimeVector.m	Takes maximum time and sampling rate as input and returns a vector of times (in seconds) for each index in the EEG data being analysed.
getEventIndex.m	Takes events and time vector as input and returns the data indices where the events occur in the EEG data vector.
VSClassifier.py	Reads in the .mat files and applies the machine learning techniques discussed earlier to the Visual Search data.
VPAClassifier.py	An analogous file for the Visual Paired Associates task.

Chapter five: Evaluation

Summary

This chapter details the process of comparing each studied classification method, calculating the accuracy of each method for each task and interpreting the resulting values. As a result, an emphasis will be placed on the validation and measurements to evaluate the overall project and to identify the favourable classification algorithm.

5.1 Solution Verification

The final step of the modified pipeline is the classification of stroke and control data according to the metric attained. There are two distinct parts to verifying our solution: the verification of the metric extracted for comparison and the verification of our technique in establishing the most accurate classification algorithm.

The verification of the utility of this metric in our classification lies in the ability to categorise new EEG data into the correct group according to this value. In order to quantify the potential of this metric, machine learning techniques suitable for binary classification are implemented. Accuracy scores are utilised to obtain a probability of correct prediction. Using the zero rule (a method of predicting the most probable classification value for all given values), a prediction accuracy of over 57.14% for the Visual Paired Associates task and over 70% for the Visual Search task would suggest that the metric we obtained is useful in classifying our data.

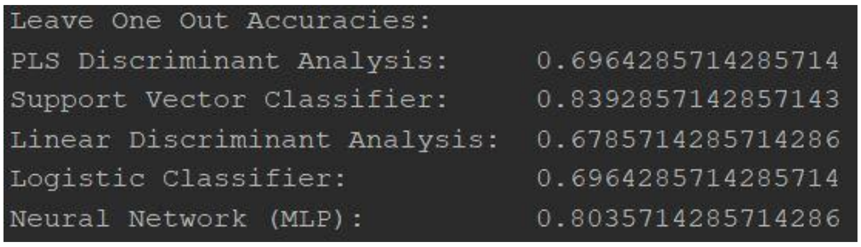
The verification of our technique in establishing the most accurate classification algorithm is easily explained. Five functions are defined which perform each classification algorithm: PLS discriminant analysis, support vector classification, linear discriminant analysis, logistic regression and multilayer perceptron (MLP) classification. Data for each task is taken as input and the leave-one-out method of separating training and testing data is applied. Each function is called, with the testing and training data as parameters and compares the predicted value with the actual value, returning the accuracy score. To assess the performance of each technique, the average accuracy score is taken over all tasks and is printed at the end. This score is used to compare the accuracies of each method concurrently and acts as an overall assessment of the accuracy of the method. The method with the highest accuracy is considered the best in this case, and because all methods are assessed concurrently on the same data, this method of comparison is valid.

5.2 Validation/Measurements

To identify the most precise machine learning method, we compare the overall accuracy of scores for each method, taking all correct responses into account for each task. To obtain this average accuracy, the leave-one-out method is used to split the data into training and test groups for each task. This is optimal given the limited number of observations in our dataset. The accuracy score is then appended to an array for each train-test split. This array is populated with all these accuracy values, and a mean accuracy is taken when all responses are considered. These mean accuracies are compared in the following section.

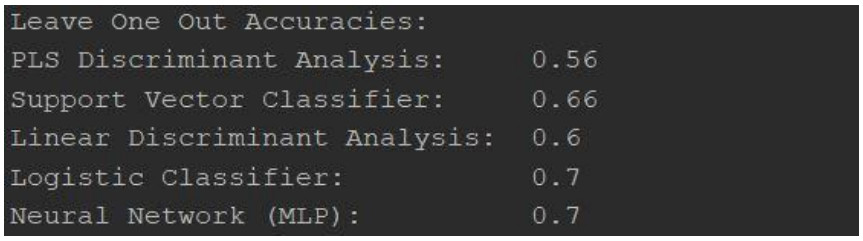
5.2.1 Results

The following figures document the mean accuracy scores for each task:



```
Leave One Out Accuracies:
PLS Discriminant Analysis:     0.6964285714285714
Support Vector Classifier:     0.8392857142857143
Linear Discriminant Analysis:  0.6785714285714286
Logistic Classifier:           0.6964285714285714
Neural Network (MLP):          0.8035714285714286
```

Figure 5.1: Mean leave-one-out accuracies for the Visual Paired Associates task.



```
Leave One Out Accuracies:
PLS Discriminant Analysis:     0.56
Support Vector Classifier:     0.66
Linear Discriminant Analysis:  0.6
Logistic Classifier:           0.7
Neural Network (MLP):          0.7
```

Figure 5.2: Mean leave-one-out accuracies for the Visual Search task.

5.2.2 Explanation of Results

Because the dataset we are analysing is small, it is difficult to determine how these accuracy scores will fluctuate on a larger dataset. The most effective way in countering this is in our use of the leave-one-out method, which takes a dataset of size n, splits it into an $n - 1: 1$ training-testing ratio and iterates through all of these data splits. This method maximises the amount of training data available, which allows us to focus specifically on the T_1 data in our analysis.

The accuracies which are printed act as an overall numerical assessment of how each method performed, given the metric attained. These are the mean accuracies over all correct responses for each task and act as a probability measure of correct classification. This allows us to infer the best classification algorithm by identifying the method which produces the highest mean accuracy in this case.

5.2.3 Analysis of Results

As previously stated in Section 5.1, a prediction accuracy of over 57.14% for the Visual Paired Associates task and over 70% for the Visual Search task suggests that the metric we obtained is useful in classifying our data. This is the case for all of the studied methods given data from the VPA task and only two of the five methods given data from the VS task (logistic regression and MLP neural networks). It follows that the metric attained generally works better to distinguish between control and stroke groups when used for the VPA task.

The individual method which obtained the highest classification accuracy score is the Support Vector Classifier. When used to classify data from the Visual Paired Associates task, an 83.93% accuracy is achieved, which is significantly above the 57.14% accuracy benchmark. This is exceptional compared to the other methods studied, most of which fall in the 65-70% accuracy range. I conducted further analysis into the mean accuracy using support vector classification given each correct response, which can be seen in the table below.

Table 5 *Mean accuracy for Support Vector Classification for each correct response in VPA task.*

Response	**Accuracy**
False Congruent Correct (FCC)	85.71%
False Incongruent Correct (FIC)	85.71%
True Congruent Correct (TCC)	85.71%
True Incongruent Correct (TIC)	78.57%

As is evident from the table, the support vector classifier has a high classification accuracy for all responses in the Visual Paired Associates task. This means that there is no concrete evidence to suggest that any particular response in this task produces more easily separable data. However, this may be explored in more detail with a larger dataset.

The multilayer perceptron classifier must not be overlooked as a useful classification tool, however. Considering both tasks in this analysis, the MLP classifier attained an accuracy of 80.36% in the Visual Paired Associates task and a 70% accuracy in the Visual Search task. It was the best classification algorithm given the Visual Search task, being one of only two algorithms to classify the data that met our baseline accuracy requirement. It performed consistently well in classifying both tasks and it is possible that it could attain a higher accuracy with a larger training dataset.

I have also provided a complete analysis of all 5 methods for the Visual Paired Associates and Visual Search tasks below, detailing the average accuracy associated with each correct response.

Table 6 *Mean accuracy for each correct response in VPA task, analysing all classification algorithms.*

Responses:	FCC	FIC	TCC	TIC
PLS Discriminant Analysis	57.14%	85.71%	71.43%	64.29%
Support Vector Classifier	85.71%	85.71%	85.71%	78.57%
Linear Discriminant Analysis	57.14%	85.71%	71.43%	57.14%
Logisitic Classifier	71.43%	78.57%	57.14%	71.43%
MLP Classifier	85.71%	78.57%	85.71%	71.43%

In this case, we can see that the false incongruent correct response tends to have a higher classification accuracy than the other correct responses. However, a larger dataset may be necessary to assess the validity of this claim.

Table 7 *Mean accuracy for each correct response in VS task, analysing all classification algorithms.*

Responses:	FLC	LC	RC	FRC	AC
PLS Discriminant Analysis	40%	50%	50%	60%	80%
Support Vector Classifier	70%	50%	70%	60%	80%
Linear Discriminant Analysis	50%	50%	60%	60%	80%
Logisitic Classifier	70%	70%	70%	70%	70%
MLP Classifier	70%	70%	70%	70%	70%

For the Visual Search task, the absent correct response seems to produce the most utile metric for classification, with accuracy scores for all methods falling in the range of 70-80%. As previously stated, more data is necessary to support the validity of this claim.

Chapter six: Conclusion

Summary

This chapter details the conclusions that arise from the analysis of results, limitations which may impact the validity of my results and possible future work that could be undertaken using the results from this project.

5.1 Results discussion

The results obtained from this study are inconclusive, since the dataset used for the comparison of the machine learning algorithms is very limited. In my analysis, there were only 14 observations for the Visual Paired Associates task (6 control and 8 stroke), and 10 observations for the Visual Search task (3 control and 7 stroke). If more data were to be made available, it would be possible to ensure more confidence in our identification of the most accurate stroke classifier and it would highlight any evidence of overfitting classification models.

One limitation of this project is the level of EEG technology available when this data was recorded. As I have mentioned in Section 2.2.1 (Electroencephalograms), a 32-electrode cap used in gathering the data for this project. It is important to note that a 256-electrode cap has been used in more recent studies, which are significantly more electrode dense, providing a higher spatial resolution for data analysis. With more recent advancements in EEG technology, it may be possible to identify more subtle elements of the data, leading to more accurate results.

Another limitation of this study is the risk associated with correcting the raw data, by removing noise from the EEG signal. The application of spatial filters to the raw EEG data could remove intricacies, which may possibly contain useful information for classification. Conversely, there may be some interference in the signal which may hinder our access to a potential biomarker.

Finally, the calculation of a classification metric may be flawed. The PO3 and PO4 electrode channels were chosen as the focus for the Visual Paired Associates task, whereas the P7 and P8 electrode channels were chosen for the Visual Search task, despite several more channels being available to choose from. Although higher power was shown by the participants in these electrode channels, there is no reason to suggest that other channels would not exhibit some form of biomarker for stroke.

Despite these risks and limitations, this project achieved the goal of investigating the utility of several machine learning algorithms in the classification of stroke and control EEG data. This modified pipeline details the full process of transforming raw EEG data into a format which can be used to successfully identify the presence of stroke in 83.93% of cases, when given the Visual Paired Associates task as described. The identification of support vector classification coupled with the VPA task as the most accurate technique in recognising stroke is the most significant result discovered in this project as it lays the foundation for further analysis of the use of EEG in stroke detection.

5.2 Future Work

The measurement of the success of this project needs to be first assessed by proving how valid the results are. In order to do this, the first major necessity is to gather more data for analysis. Although measures were introduced such as the deployment of the leave-one-out method to separate data into training and testing groups, if more data were available for analysis, there would be more confidence in the validity of the results of this thesis.

It is unlikely that support vector classification alone is objectively the most accurate method in classifying this data so further research could be conducted into other classification techniques. For the purposes of this thesis, I have only considered 8 in total, 3 which did not produce any meaningful results (see Appendix 1). However, there are many binary classification methods which could be trialled on this data, including decision trees, random forests, different types of neural networks, Bayesian networks and many others. An optimal classifier may be developed by experimenting with a larger dataset, identifying cluster criteria and experimentation with hybrid classification techniques.

This project also focused only on the alpha band of a subset of the electrodes available to analyse in each task. Although this was reasoned through clearly, future effort may be employed to analyse different power bands and different electrode combinations, to thoroughly explore the possibility of stroke biomarker identification. In addition to this, the VPA data I received was nasion referenced, whilst the VS data was referenced from 0 volts. The use of different montages to reference each channel could prove beneficial in providing more robust data to analyse and separate, which provides endless possibilities regarding the future analysis of this project.

Another direction in which this project can be taken is the use of results from pen and paper tasks along with EEG data in stroke classification. Each participant in this experiment has also completed a pen and paper cognitive task and their proficiency scores were recorded. The use of these along with the EEG metric extracted may ameliorate the accuracy scores exhibited by the classification algorithms, which may be investigated in future.

A final opportunity for future work I would like to discuss is the process of attaining a metric. More research could be conducted to identify an alternative classification metric which causes data from each group to appear in disjoint clusters. Evidence has been provided that the metric extracted from the data currently can be used the classify the data accurately, but this classification accuracy could possibly be enhanced with an alternative metric.

Overall, this project has identified an appropriate classification algorithm and has assessed its benefit in distinguishing stroke EEG data from control data on the limited dataset provided. This has provided foundations in the utility of machine learning to categorise EEG signals which would be very beneficial to society if done more accurately. At the very least, this project has shown that EEG data recorded in the days after stroke has shown some level of distinction from control data. With rapid advancements in EEG technology in recent times, future work will be beneficial in furthering the utility of statistical machine learning in classifying EEG data of stroke patients.

References

Abirami, S., & Chitra, P. (2020). Energy-efficient edge based real-time healthcare support system. *Advances in Computers*, 339–368. https://doi.org/10.1016/bs.adcom.2019.09.007

Al-Qazzaz, N., Ali, S., Ahmad, S. A., Islam, S., & Mohamad, K. (2014). Cognitive impairment and memory dysfunction after a stroke diagnosis: a post-stroke memory assessment. *Neuropsychiatric Disease and Treatment*, 1677. https://doi.org/10.2147/ndt.s67184

American Stroke Association. (n.d.). *Types of Stroke*. Www.Stroke.Org. https://www.stroke.org/en/about-stroke/types-of-stroke

Berg, P., & Scherg, M. (1994). A multiple source approach to the correction of eye artifacts. *Electroencephalography and Clinical Neurophysiology*, *90*(3), 229–241. https://doi.org/10.1016/0013-4694(94)90094-9

BESA. (2021). [Software]. https://www.besa.de

EDFBrowser. (2021). [Software]. https://www.teuniz.net/edfbrowser

EEGLab Wiki. (n.d.). EEGLAB Wiki. https://eeglab.org

Irish Heart. (2020, November 17). *What is Stroke?* https://irishheart.ie/your-health/learn-about-stroke/what-is-stroke

Korpelainen, J. T., Kauhanen, M. L., Tolonen, U., Brusin, E., Mononen, H., Hiltunen, P., Sotaniemi, K. A., Suominen, K., & Myllylä, V. V. (2000). Auditory P300 event related potential in minor ischemic stroke. *Acta Neurologica Scandinavica*, *101*(3), 202–208. https://doi.org/10.1034/j.1600-0404.2000.101003202.x

Mohanty, N., John, A. L. S., Manmatha, R., & Rath, T. (2013). Shape-Based Image Classification and Retrieval. *Handbook of Statistics - Machine Learning: Theory and Applications*, 249–267. https://doi.org/10.1016/b978-0-444-53859-8.00010-2

Noble, W. S. (2006). What is a support vector machine? *Nature Biotechnology*, *24*(12), 1565–1567. https://doi.org/10.1038/nbt1206-1565

Olejniczak, P. (2006). Neurophysiologic Basis of EEG. *Journal of Clinical Neurophysiology*, *23*(3), 186–189. https://doi.org/10.1097/01.wnp.0000220079.61973.6c

Quiroga, R., & Garcia, H. (2003). Single-trial event-related potentials with wavelet denoising. *Clinical Neurophysiology*, *114*(2), 376–390. https://doi.org/10.1016/s1388-2457(02)00365-6

Ruiz-Perez, D., Guan, H., Madhivanan, P., Mathee, K., & Narasimhan, G. (2020). So you think you can PLS-DA? *BMC Bioinformatics*, *21*(S1), 1. https://doi.org/10.1186/s12859-019-3310-7

Urso, A., Fiannaca, A., La Rosa, M., Ravì, V., & Rizzo, R. (2019). Data Mining: Prediction Methods. *Encyclopedia of Bioinformatics and Computational Biology*, 413–430. https://doi.org/10.1016/b978-0-12-809633-8.20462-7

What is EEGLAB? (2021). Swartz Center for Computational Neuroscience. https://sccn.ucsd.edu/eeglab/index.php

Wu, J., Srinivasan, R., Burke Quinlan, E., Solodkin, A., Small, S. L., & Cramer, S. C. (2016). Utility of EEG measures of brain function in patients with acute stroke. *Journal of Neurophysiology, 115*(5), 2399–2405. https://doi.org/10.1152/jn.00978.2015

Appendices

Appendix 1 Discussion of classification algorithms omitted from study

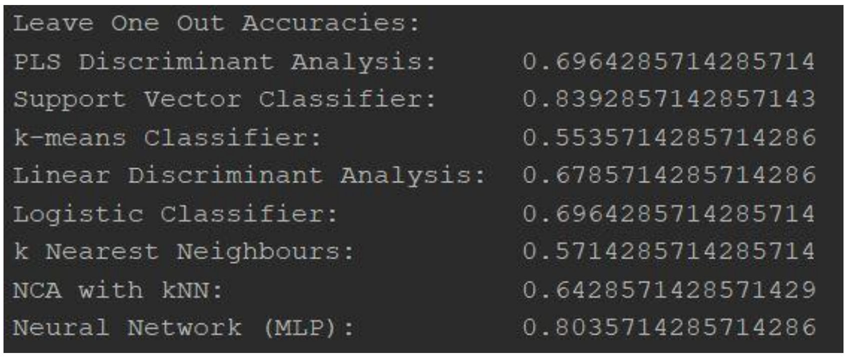
Although 5 classification algorithms were trialled for the purposes of this project, another 3 were implemented but omitted from the final study due to poor performance: k-means cluster prediction, k nearest neighbours and neighbourhood components analysis (with k nearest neighbours).

The method of k-means clustering is the use of an algorithm which partitions the dataset into k non-overlapping clusters. This involves choosing k initial centroids and partitioning objects into k clusters. This is done by assigning them to clusters with the closest centroid using the Euclidean metric. New centroids are then chosen for these clusters and the algorithm converges when no further reassignments take place. In the clustering of stroke and control data, we will use $k = 2$. We then predict the clusters in which the test data are classified.

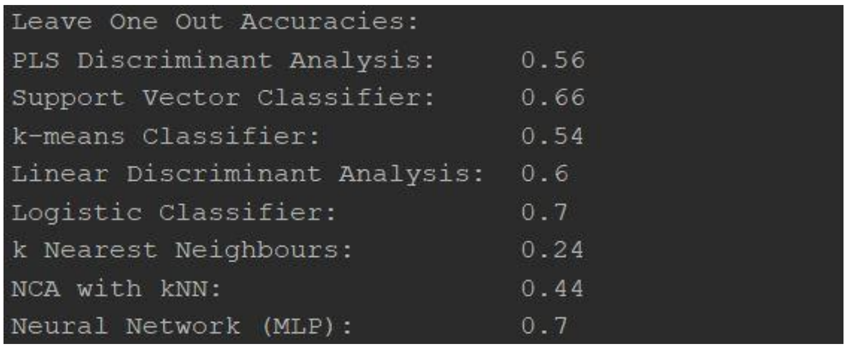
The k nearest neighbours classification technique, each object is classified according to the class which is most common among the k nearest neighbours to the point being classified, using the Euclidean metric. For example, if $k = 1$, the point being classified is placed into the same class as its closest neighbour. Small, odd values for k are typically chosen to prevent the algorithm from coming down to chance (with an equal number of proximal values from each class).

Neighbourhood components analysis is quite similar to the k nearest neighbours algorithm, classifying multivariate data into classes by using a specific distance metric. This distance metric is learned by linearly transforming input data such that the classification accuracy when using the leave-one-out method is maximised in the transformed space. When used with the k nearest neighbours technique, the classification accuracy usually increases.

The following figures indicate the performance of these algorithms given the T_1 data for the Visual Paired Associates and Visual Search tasks respectively.



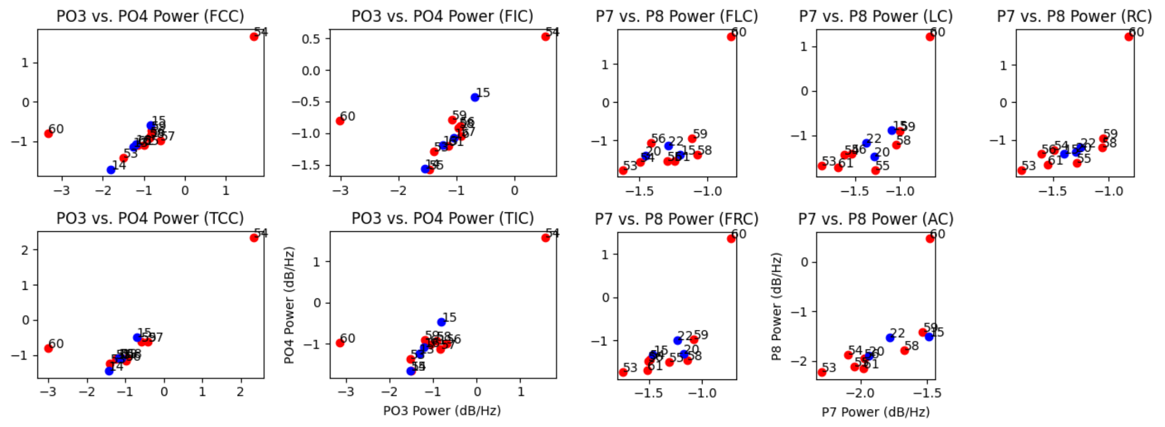
```
Leave One Out Accuracies:
PLS Discriminant Analysis:     0.6964285714285714
Support Vector Classifier:     0.8392857142857143
k-means Classifier:            0.5535714285714286
Linear Discriminant Analysis:  0.6785714285714286
Logistic Classifier:           0.6964285714285714
k Nearest Neighbours:          0.5714285714285714
NCA with kNN:                  0.6428571428571429
Neural Network (MLP):          0.8035714285714286
```



```
Leave One Out Accuracies:
PLS Discriminant Analysis:     0.56
Support Vector Classifier:     0.66
k-means Classifier:            0.54
Linear Discriminant Analysis:  0.6
Logistic Classifier:           0.7
k Nearest Neighbours:          0.24
NCA with kNN:                  0.44
Neural Network (MLP):          0.7
```

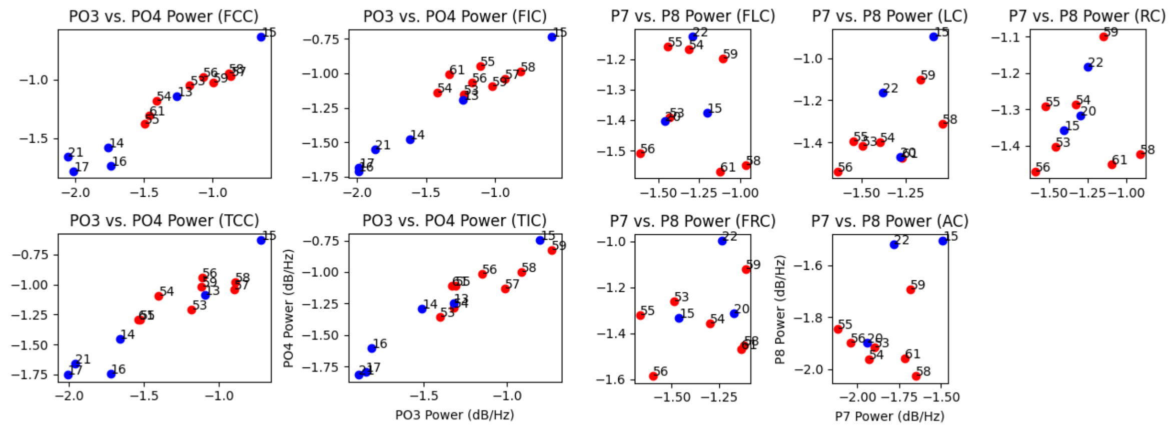
Appendix 2 Scatterplots at T_2 for both tasks

These are the scatterplots for T_2 for each task. The scatterplot associated with the Visual Paired Associates task is seen on the left, and the Visual Search scatterplot can be seen on the right.



Appendix 3 Scatterplots at T_1 for both tasks with outlier removed

These are the scatterplots that were produced as a result of the removal of the outlying observation, number 60. The scatterplot associated with the Visual Paired Associates task is seen on the left, and the Visual Search scatterplot can be seen on the right.



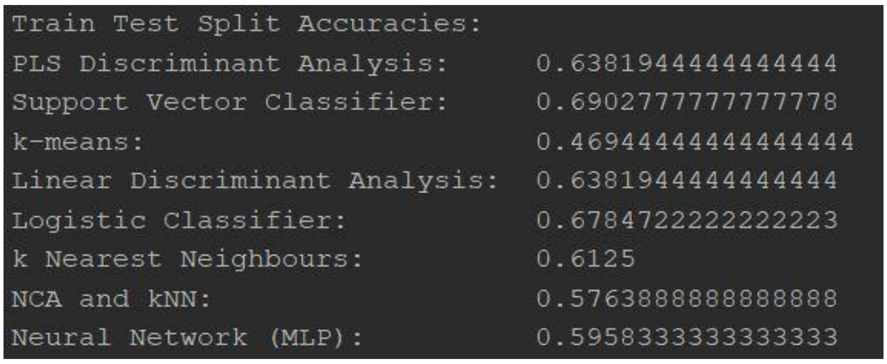
Appendix 4 Use of machine learning on T_1 to predict T_2 classifications

As discussed in the section 4.2.4 (Statistical Analysis) of the thesis, an attempt was made to use the T_1 data as training data and the T_2 data as testing data for each task. A function was defined which implemented this method, however because the T_2 data was not easily separable (see Appendix 2 for scatterplots), the accuracy scores returned show a worse performance for the Visual Paired Associates task.

Although this is seen, it is important to note that this training-testing method seems to perform slightly better than the leave-one-out method for the Visual Search task. However, this performance requires validation because there are only 10 observations altogether in each testing and training group.

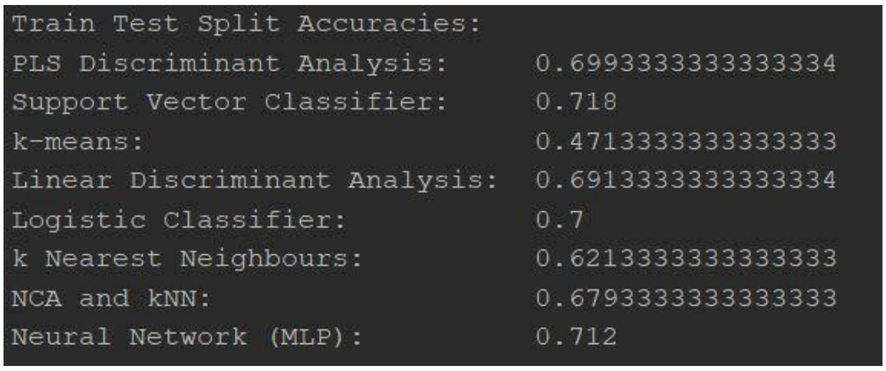
The mean accuracy scores of each classification technique using this training-testing method are summarised below for both tasks:

Visual Paired Associates mean accuracy scores:



```
Train Test Split Accuracies:
PLS Discriminant Analysis:     0.6381944444444444
Support Vector Classifier:     0.6902777777777778
k-means:                       0.46944444444444444
Linear Discriminant Analysis:  0.6381944444444444
Logistic Classifier:           0.6784722222222223
k Nearest Neighbours:          0.6125
NCA and kNN:                   0.5763888888888888
Neural Network (MLP):          0.5958333333333333
```

Visual Search mean accuracy scores:



```
Train Test Split Accuracies:
PLS Discriminant Analysis:     0.6993333333333334
Support Vector Classifier:     0.718
k-means:                       0.4713333333333333
Linear Discriminant Analysis:  0.6913333333333334
Logistic Classifier:           0.7
k Nearest Neighbours:          0.6213333333333333
NCA and kNN:                   0.6793333333333333
Neural Network (MLP):          0.712
```

Appendix 5 Python scripts developed for classification

Two very similar Python scripts were developed to trial each classification algorithm studied: one for each cognitive task. Because of the many similarities between both scripts, I will just document the code developed to determine the best classification algorithm for the Visual Paired Associates data.

Appendix 5.1 VPAClassifier.py

```
'''
Author: Dean Connell
Description: Converts data in folders into a dataframe
and tests various classifiers to determine which is most
accurate in classifying the data for VPA task.
'''

import numpy as np
import matplotlib.pyplot as plot
from scipy.io import loadmat
from sklearn.cross_decomposition import PLSRegression
from statistics import mean
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split, LeaveOneOut
```

```python
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.neighbors import (NeighborhoodComponentsAnalysis,
KNeighborsClassifier)
from sklearn.pipeline import Pipeline
from sklearn.neural_network import MLPClassifier
import warnings

warnings.filterwarnings("ignore", category=np.VisibleDeprecationWarning)


# splitting the frequency array into n segments
def segmentFrequencyBand(freqarr, n):
    average = len(freqarr) / float(n)
    output = []
    final = 0.0
    while final < len(freqarr):
        output.append([int(final), int(final + average)])
        final = final + average
    return output


# need to define our frequency bands and extract alpha band.
freqs = [0]
i = 1
while i < 280:
    freqs.append(freqs[-1] + 0.25)
    i = i + 1

freqbands = segmentFrequencyBand(freqs, 4)
alphaBand = [freqbands[0][0], freqbands[0][1]]

# defining powermap - names of struct elements
powermap = ["fccpow", "ficpow", "tccpow", "ticpow"]
respnames = ["FCC", "FIC", "TCC", "TIC"]


# store periodogram data
def storeMetricData(fileArr, path):
    mapArr = []
    temp = np.zeros(4)
    resp = 0
    for i in range(len(fileArr)):
        filein = loadmat(path + str(fileArr[i]) + ".mat").get('patient_erp')
        while resp < 4:
            response = filein[powermap[resp]]
            response = response[0][0][alphaBand[0]:alphaBand[1]]
            temp[resp] = sum(response) / len(response)
            resp = resp + 1
        mapArr.append(temp)
        temp = np.zeros(4)
        resp = 0
    return mapArr


# numbering files so we can search for them in folders specified
controlFiles = [13, 14, 15, 16, 17, 21]
strokeFiles = [53, 54, 55, 56, 57, 58, 59, 61]
controlFilesT2 = [13, 14, 15, 21]
```

```python
# importing the files necessary for analysis
PO3ConT1 = storeMetricData(controlFiles, "ERP Files\\VPA\\Controls\\T1\\PO3\\")
PO4ConT1 = storeMetricData(controlFiles, "ERP Files\\VPA\\Controls\\T1\\PO4\\")
PO3StrT1 = storeMetricData(strokeFiles, "ERP Files\\VPA\\Patients\\T1\\PO3\\")
PO4StrT1 = storeMetricData(strokeFiles, "ERP Files\\VPA\\Patients\\T1\\PO4\\")

PO3ConT2 = storeMetricData(controlFilesT2, "ERP Files\\VPA\\Controls\\T2\\PO3\\")
PO4ConT2 = storeMetricData(controlFilesT2, "ERP Files\\VPA\\Controls\\T2\\PO4\\")
PO3StrT2 = storeMetricData(strokeFiles, "ERP Files\\VPA\\Patients\\T2\\PO3\\")
PO4StrT2 = storeMetricData(strokeFiles, "ERP Files\\VPA\\Patients\\T2\\PO4\\")


# assignment to values 1 (stroke) and 0 (control):
def createDataframe(elec1, elec2, id):
    if id == 0:
        dataframe = np.array([[elec1], [elec2], [np.zeros(len(elec1))]])
    else:
        dataframe = np.array([[elec1], [elec2], [np.ones(len(elec1))]])
    return dataframe


# creating stroke and control dataframes
controlT1 = createDataframe(PO3ConT1, PO4ConT1, 0)
strokeT1 = createDataframe(PO3StrT1, PO4StrT1, 1)
controlT2 = createDataframe(PO3ConT2, PO4ConT2, 0)
strokeT2 = createDataframe(PO3StrT2, PO4StrT2, 1)

# defining training data in useful format
x1 = np.array([np.concatenate((controlT1[0, 0], strokeT1[0, 0]), axis=0),
               np.concatenate((controlT1[1, 0], strokeT1[1, 0]), axis=0)])
y1 = np.concatenate((controlT1[2, 0], strokeT1[2, 0]), axis=0)
x2 = np.array([np.concatenate((controlT2[0, 0], strokeT2[0, 0]), axis=0),
               np.concatenate((controlT2[1, 0], strokeT2[1, 0]), axis=0)])
y2 = np.concatenate((controlT2[2, 0], strokeT2[2, 0]), axis=0)
y1 = y1.astype(int)
y2 = y2.astype(int)

# print(x1) # PO3, PO4 data
# print(y1) # 0 or 1 stroke or control


# function that takes training data as array and extracts response data for
response at index n
def extractResponseData(trainingData, n):
    respData = np.zeros((len(trainingData[0]), 2))
    rowcount = 0
    colcount = 0
    for row in trainingData:
        for element in row:
            respData[rowcount][colcount] = element[n]
            rowcount = rowcount + 1
        rowcount = 0
        colcount = colcount + 1
    return respData


# response types are turned into training sets
FCCT1 = extractResponseData(x1, 0)
FICT1 = extractResponseData(x1, 1)
TCCT1 = extractResponseData(x1, 2)
TICT1 = extractResponseData(x1, 3)
```

```
FCCT2 = extractResponseData(x2, 0)
FICT2 = extractResponseData(x2, 1)
TCCT2 = extractResponseData(x2, 2)
TICT2 = extractResponseData(x2, 3)


# plots the Power Spectral Densities of one eletrode against another for both
groups.
def PSDPlots(StrPO3, StrPO4, ConPO3, ConPO4, StrFiles, ConFiles):
    n = 0
    while n < 4:
        plot.subplot(2, 2, n+1)
        i = 0
        while i < len(StrPO4):
            plot.scatter(StrPO3[i][n], StrPO4[i][n], color='red')
            plot.annotate(str(StrFiles[i]), (StrPO3[i][n], StrPO4[i][n]))
            i = i+1
        i = 0
        while i < len(ConPO4):
            plot.scatter(ConPO3[i][n], ConPO4[i][n], color='blue')
            plot.annotate(str(ConFiles[i]), (ConPO3[i][n], ConPO4[i][n]))
            i = i+1
        plot.title("PO3 vs. PO4 Power (" + respnames[n] + ")")
        n = n+1
    plot.xlabel("PO3 Power (dB/Hz)")
    plot.ylabel("PO4 Power (dB/Hz)")
    plot.show()

# Generating scatterplots for T1 and T2
# PSDPlots(PO3StrT1, PO4StrT1, PO3ConT1, PO4ConT1, strokeFiles, controlFiles)
# PSDPlots(PO3StrT2, PO4StrT2, PO3ConT2, PO4ConT2, strokeFiles, controlFiles)


# Method 1: PLS Discriminant Analysis
def PLSDisAnalysis(xTrain, yTrain, xTest, yTest):
    print("Partial Least Squares Discriminant Analysis:")
    pls = PLSRegression(n_components=2)
    pls.fit(xTrain, yTrain)
    plspred = (pls.predict(xTest)[:, 0] > 0.5).astype('uint8')
    confusion = confusion_matrix(yTest, plspred)
    accuracy = accuracy_score(yTest, plspred)
    print("Predictions: ", plspred)
    print("Actual:      ", yTest)
    # print("Confusion Matrix:\n", confusion)
    print("Accuracy:    ", accuracy)
    print()
    return accuracy


# Method 2: K-Means Clustering
def kMeansClassification(xTrain, yTrain, xTest, yTest):
    print("k-means Classifier:")
    kmeans = KMeans(n_clusters=2, random_state=0)
    kmeans.fit(xTrain, yTrain)
    kmpred = kmeans.predict(xTest)
    confusion = confusion_matrix(yTest, kmpred)
    accuracy = accuracy_score(yTest, kmpred)
    print("Predictions: ", kmpred)
    print("Actual:      ", yTest)
    # print("Confusion Matrix:\n", confusion)
    print("Accuracy:    ", accuracy)
    print()
    return accuracy
```

```python
# Method 3: SVC
def SupportVectorClassifier(xTrain, yTrain, xTest, yTest):
    print("Support Vector Classification:")
    svc = SVC(kernel='poly', gamma='auto')
    svc.fit(xTrain, yTrain)
    svcpred = svc.predict(xTest)
    confusion = confusion_matrix(yTest, svcpred)
    accuracy = accuracy_score(yTest, svcpred)
    print("Predictions: ", svcpred)
    print("Actual:      ", yTest)
    # print("Confusion Matrix:\n", confusion)
    print("Accuracy:    ", accuracy)
    print()
    return accuracy


# Method 4: Linear Discriminant Analysis
def LinearDisAnalysis(xTrain, yTrain, xTest, yTest):
    print("Linear Discriminant Analysis:")
    linreg = LinearDiscriminantAnalysis()
    linreg.fit(xTrain, yTrain)
    linpred = linreg.predict(xTest)
    confusion = confusion_matrix(yTest, linpred)
    accuracy = accuracy_score(yTest, linpred)
    print("Predictions: ", linpred)
    print("Actual:      ", yTest)
    # print("Confusion Matrix:\n", confusion)
    print("Accuracy:    ", accuracy)
    print()
    return accuracy


# Method 5: Logistic Regression
def LogisticClassifier(xTrain, yTrain, xTest, yTest):
    print("Logistic Regression:")
    logreg = LogisticRegression(random_state=0)
    logreg.fit(xTrain, yTrain)
    logpred = logreg.predict(xTest)
    confusion = confusion_matrix(yTest, logpred)
    accuracy = accuracy_score(yTest, logpred)
    print("Predictions: ", logpred)
    print("Actual:      ", yTest)
    # print("Confusion Matrix:\n", confusion)
    print("Accuracy:    ", accuracy)
    print()
    return accuracy


# Method 6: K-Nearest Neighbours
def KNN(xTrain, yTrain, xTest, yTest):
    print("k Nearest Neighbours Classification:")
    knn = KNeighborsClassifier(n_neighbors=2)
    knn.fit(xTrain, yTrain)
    knnpred = knn.predict(xTest)
    confusion = confusion_matrix(yTest, knnpred)
    accuracy = accuracy_score(yTest, knnpred)
    print("Predictions: ", knnpred)
    print("Actual:      ", yTest)
    # print("Confusion Matrix:\n", confusion)
    print("Accuracy:    ", accuracy)
    print()
    return accuracy
```

```
# Method 7: NeighbourhoodComponentsAnalysis, KNN Classifier
def NCAKNN(xTrain, yTrain, xTest, yTest):
    print("Neighbourhood Component Analysis withs kNN Classification")
    nca = NeighborhoodComponentsAnalysis(random_state=42)
    knn = KNeighborsClassifier(n_neighbors=2)
    ncapipe = Pipeline([('nca', nca), ('knn', knn)])
    ncapipe.fit(xTrain, yTrain)
    ncapipepred = ncapipe.predict(xTest)
    confusion = confusion_matrix(yTest, ncapipepred)
    accuracy = accuracy_score(yTest, ncapipepred)
    print("Predictions: ", ncapipepred)
    print("Actual:      ", yTest)
    # print("Confusion Matrix:\n", confusion)
    print("Accuracy:    ", accuracy)
    print()
    return accuracy


# Method 8: Neural Networks
def NeuralNetwork(xTrain, yTrain, xTest, yTest):
    print("Neural Network (MLP):")
    nn = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2),
random_state=1)
    nn.fit(xTrain, yTrain)
    nnpred = nn.predict(xTest)
    confusion = confusion_matrix(yTest, nnpred)
    accuracy = accuracy_score(yTest, nnpred)
    print("Predictions: ", nnpred)
    print("Actual:      ", yTest)
    # print("Confusion Matrix:\n", confusion)
    print("Accuracy:    ", accuracy)
    print()
    return accuracy


def LeaveOneOutT1(tasks):
    PLS = []
    SVC = []
    KMC = []
    LiC = []
    LoC = []
    Knn = []
    NCA = []
    NN = []
    for i in range(len(tasks)):
        task = tasks[i]
        loo = LeaveOneOut()
        loo.get_n_splits(task, y1)
        for train_index, test_index in loo.split(task, y1):
            x1Train, x1Test = task[train_index], task[test_index]
            y1Train, y1Test = y1[train_index], y1[test_index]
            # calling classification functions
            PLS.append(PLSDisAnalysis(x1Train, y1Train, x1Test, y1Test))
            SVC.append(SupportVectorClassifier(x1Train, y1Train, x1Test, y1Test))
            KMC.append(kMeansClassification(x1Train, y1Train, x1Test, y1Test))
            LiC.append(LinearDisAnalysis(x1Train, y1Train, x1Test, y1Test))
            LoC.append(LogisticClassifier(x1Train, y1Train, x1Test, y1Test))
            Knn.append(KNN(x1Train, y1Train, x1Test, y1Test))
            NCA.append(NCAKNN(x1Train, y1Train, x1Test, y1Test))
            NN.append(NeuralNetwork(x1Train, y1Train, x1Test, y1Test))
        print("Leave-One-Out Accuracies:")
        print("PLS Discriminant Analysis:    ", mean(PLS))
        print("Support Vector Classifier:    ", mean(SVC))
        print("k-means Classifier:           ", mean(KMC))
        print("Linear Discriminant Analysis: ", mean(LiC))
        print("Logistic Classifier:          ", mean(LoC))
        print("k Nearest Neighbours:         ", mean(Knn))
        print("NCA with kNN:                 ", mean(NCA))
        print("Neural Network (MLP):         ", mean(NN))


# note: if calling this function, it may be beneficial
# to print confusion matrices in each classification function.
def TrainTestSplitT2(tasks, responses):
    PLS = []
    SVC = []
    KMC = []
    LiC = []
    LoC = []
    Knn = []
    NCA = []
    NN = []
    # making test and training data
    for iterations in range(30):
        # 30 iterations for accuracies to converge
        for i in range(len(tasks)):
            x1Train, x1Test, y1Train, y1Test = train_test_split(tasks[i], y1,
test_size=0.1)
            y1Train = y1Train.astype(int)
            y1Test = y1Test.astype(int)
            # Calling classification functions:
            PLS.append(PLSDisAnalysis(x1Train, y1Train, responses[i], y2))
            SVC.append(SupportVectorClassifier(x1Train, y1Train, responses[i], y2))
            KMC.append(kMeansClassification(x1Train, y1Train, responses[i], y2))
            LiC.append(LinearDisAnalysis(x1Train, y1Train, responses[i], y2))
            LoC.append(LogisticClassifier(x1Train, y1Train, responses[i], y2))
            Knn.append(KNN(x1Train, y1Train, responses[i], y2))
            NCA.append(NCAKNN(x1Train, y1Train, responses[i], y2))
            NN.append(NeuralNetwork(x1Train, y1Train, responses[i], y2))
    print("Train Test Split Accuracies:")
    print("PLS Discriminant Analysis:    ", mean(PLS))
    print("Support Vector Classifier:    ", mean(SVC))
    print("k-means:                      ", mean(KMC))
    print("Linear Discriminant Analysis: ", mean(LiC))
    print("Logistic Classifier:          ", mean(LoC))
    print("k Nearest Neighbours:         ", mean(Knn))
    print("NCA and kNN:                  ", mean(NCA))
    print("Neural Network (MLP):         ", mean(NN))


tasks = [FCCT1, FICT1, TCCT1, TICT1]
responses = [FCCT2, FICT2, TCCT2, TICT2]

# TrainTestSplitT2(tasks, responses)
LeaveOneOutT1(tasks)
```