Anne Joo Yun Marthinsen
Ivar Tesdal Galtung

Investigation of mental stress detection with an 8-channel EEG system using KNN, SVM and EEGNet

Master's thesis in Cybernetics and Robotics
Supervisor: Marta Molinas
June 2023

Anne Joo Yun Marthinsen
Ivar Tesdal Galtung

Investigation of mental stress detection with an 8-channel EEG system using KNN, SVM and EEGNet

Master's thesis in Cybernetics and Robotics
Supervisor: Marta Molinas
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

It is important to preface that this work is done in continuation of the project thesis by Marthinsen [1]. Thus there will be sections, especially in chapter 2, that will be heavily inspired by or directly repurposed from the project thesis.

The aim of the project thesis was to investigate the potential use of Electroencephalography (EEG) for detecting mental stress. The study found that the most successful approach involved utilizing Hjorth features extracted from an ICA-filtered, 32-channeled dataset, which was subsequently classified via the K-Nearest-Neighbours (KNN) classifier. This method yielded accuracy, sensitivity, and specificity rates of 96.50%, 97.18%, and 95.89%, respectively. These findings suggest that certain classifiers and feature extraction methods can be effective for identifying mental stress from EEG data. However, the sample size in the project thesis was relatively small, and further research involving a larger sample size would be necessary to verify these findings.

In this study, a more restricted EEG dataset was gathered with an 8-channel EEG configuration. The new dataset underwent pre-processing and filtering procedures identical to those applied in the project thesis. Following this, the dataset was classified utilizing the same methods as before. However, the results were not as satisfactory as in the previous study, necessitating the exploration of alternative options. Subsequently, EEGNets' Convolutional Neural Networks (CNNs), which are specialized in the classification of EEG data, were employed, as well as an implementation of Deep and Shallow CNNs. Furthermore, new approaches for filtering and feature extraction were tested.

The best results using traditional classifiers were obtained using full RAW data with SVM (accuracy: 87.50%, sensitivity: 81.25%, specificity: 92.05%), time series features of RAW data with SVM (accuracy: 86.84%, sensitivity: 82.81%, specificity: 98.77%) and wavelet scattering features of RAW data with SVM (accuracy: 87.50%, sensitivity: 82.81%, specificity: 90.91%). Deep Convolutional Neural Network models also performed satisfactorily with the best-performing model being the Shallow CNN with a mean accuracy of 83.66% across all folds.

It is believed that the use of ICA for filtering did not prove as effective as in the earlier study. With that said the raw data showed promising results for capturing mental stress when using CNNs. As such, the reduction in the number of measuring points from 32 to 8 appears to have constrained the study, underscoring the need for a more comprehensive approach in future research.

Sammendrag

Det er viktig å påpeke at dette arbeidet er gjort som en fortsettelse av prosjektoppgaven av Marthinsen [1]. Derfor vil det være seksjoner, spesielt i chapter 2, som vil være sterkt inspirert av eller direkte gjenbrukt fra prosjektoppgaven.

Målet med prosjektoppgaven var å undersøke potensialet for å bruke Electroencephalography (EEG) til å oppdage mentalt stress. Studien fant at den mest vellykkede modellen involverte bruk av Hjorth- funksjonsekstraksjon fra et ICA-filtrert, 32-kanals datasett som deretter ble klassifisert via K-Nearest-Neighbours (KNN) klassifisering. Denne metoden ga nøyaktighets-, følsomhets- og spesifisitetsrater på henholdsvis 96,50%, 97,18% og 95,89%. Disse funnene antyder at visse klassifiserings- og funksjonsekstraksjonsmetoder kan være effektive for å identifisere mentalt stress fra EEG-data. Imidlertid var utvalgsstørrelsen i denne studien relativt liten, og ytterligere forskning med en større utvalgsstørrelse vil være nødvendig for å verifisere disse funnene.

I denne studien ble det samlet inn et mer begrenset EEG-datasett med en 8-kanals EEG-konfigurasjon. Det nye datasettet gjennomgikk pre-prosessering og filtreringsprosedyrer som var identiske med de som ble brukt i prosjektoppgaven. Etter dette ble datasettet klassifisert ved hjelp av de samme metodene som før. Resultatene var imidlertid ikke like tilfredsstillende som tildligere, noe som gjorde det nødvendig å utforske alternative metoder. Derfor ble EEGNets' Convolutional Neural Networks (CNNs), som er spesialisert på klassifisering av EEG-data, tatt i bruk, i tillegg til en implementasjon av Deep og Shallow CNNs. Videre ble nye tilnærminger for filtrering og funksjonsekstraksjon testet.

De beste resultatene ved bruk av tradisjonelle klassifiseringsmetoder ble oppnådd ved bruk av fullstendige RÅ-data med SVM (nøyaktighet: 87,507%, følsomhet: 81,25%, spesifisitet: 92,05%), tidsseriefunksjoner av RÅ-data med SVM (nøyaktighet: 86,84%, følsomhet: 82,81%, spesifisitet: 98,77%) og bølge-spredningstransformasjon funksjoner av RÅ-data med SVM (nøyaktighet: 87,50%, følsomhet: 82,81%, spesifisitet: 90,91%). Dype CNN-modeller presterte også tilfredsstillende, hvor den best-presterende modellen var Shallow CNN med en gjennomsnittlig nøyaktighet på 83,66% på tvers av alle folder.

Det antas at bruken av ICA for filtrering ikke var like effektiv som i den tidligere studien. Med det sagt viste rådataene lovende resultater for å fange mentalt stress ved bruk av CNNs til klassifisering. Dermed ser det ut til at reduksjonen i antall målepunkter fra 32 til 8 har begrenset studien, noe som understreker behovet for

en mer omfattende tilnærming i fremtidig forskning.

Acknowledgements

We would like to take this opportunity to express our heartfelt thanks to everyone who played a part in making our master's thesis a reality.

First and foremost, we would like to extend our appreciation to the Department of Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU), and especially to our supervisor, Ph.D Marta Molinas. Her guidance, support, and expertise within the fields of EEG signal properties and Machine Learning classifiers have been indispensable for the successful completion of our project.

In addition, we would like to show some love and appreciation to our families, friends, and significant others who have supported us through this journey. Ivar would especially like to thank Shavin for her unwavering encouragement, understanding, and insightful advice. Anne Joo wants to thank Herman for his patience, aid, and motivational pointers, and Sif and Kristine, her dear friends and study partners. Your encouragement and belief in our abilities have kept our spirits high during this time. We would also like to thank our collaboration partner, Ida, for her valuable contributions, teamwork, and enthusiastic exchange of ideas that have enhanced our learning experience and enriched the quality of our study. We are incredibly grateful for her companionship and the beneficial insights she has brought to our project.

Lastly, we would like to once again express our gratitude to the participants of our study for their participation in our data collection. Without your time and involvement, we would not have gained such a deep understanding of EEG signal collection and processing. Their contribution has been invaluable, and we are immensely thankful for their participation.

Thank you all.

Sincerely,
Ivar & Anne Joo

Contents

Acronyms

AI Artificial Intelligence. 12

BCIs Brain Computer Interfaces. 1, 41

CNN Convolutional Neural Network. iii, v, x, 17, 41, 42, 72, 73, 75

CNNs Convolutional Neural Networks. iii, v, 17, 42, 72

EEG Electroencephalography. iii, v, vii, ix, 1–4, 6–11, 18, 21–23, 26–29, 32, 33, 35, 36, 38, 41, 43, 67–69, 71–73, 75

FBCSP Filter Bank Common Spatial Pattern. 41

ICA Independent Component Analysis. iii, v, ix, x, 3, 4, 10, 21, 28, 35–38, 43, 56–58, 69, 70

KNN K-Nearest-Neighbours. iii, v, ix–xi, 2, 3, 13, 15, 21, 28, 38, 42–58, 65, 70, 72, 75, 95–105

ML Machine Learning. vii, ix, x, 1–3, 12, 13, 17, 18, 22, 38, 73, 75

MLP Multilayer Perceptron. 21, 38

PCG Phonocardiogram. ix, 3, 21, 22, 26, 27

PSD Power Spectral Density. ix–xi, 3, 4, 18, 32, 34, 39, 40, 71, 105

SGL Sparse Group Lasso. 41

SS Stress Scale. x, 2, 30–32, 48–51, 70, 71, 95–101

STAI-Y State Trait Anxiety Inventory for Adults Form Y-1 and Y-2. x, 2, 22, 23, 26, 30–32, 44–47, 52–58, 65, 71, 102–105

SVM Support Vector Machines. iii, v, ix–xi, 2, 13, 14, 16, 21, 38, 39, 42, 44–60, 65, 70, 72, 75, 95–105

TSGL Temporal constrained Sparse Group Lasso. 41

TSGL-EEGNet Temporal-constrained Sparse Group Lasso-EEGNet. x, 4, 41, 71

Chapter 1

Introduction

1.1 Background and Motivation

Mental health issues related to stress is a global problem that has been on the rise due to the pandemic and humanitarian crises. The outbreak of COVID-19 has led to the loss of millions of lives and disrupted livelihoods, forcing people to face a range of stressors such as job loss, isolation, and fear of illness. As a consequence, mental health concerns have become a global issue, and there is a growing need for research, treatment, and support.

Stress detection and associated research are important parts of mental health-related investigation, and early identification can be beneficial in managing the issues. Researchers are exploring different biomarkers that could signal the presence of mental stress, such as cortisol levels, heart rate variability, and changes in speech patterns. Early detection can enable healthcare providers to plan appropriate interventions, such as therapy or medication, which can ultimately save lives by effectively managing mental health concerns.

Machine Learning (ML) technology is constantly progressing and being applied to new fields for decision-making purposes. Extensive attempts to use Machine Learning have been done in the field of Brain Computer Interfaces (BCIs) for causes such as predicting seizures for patients with epilepsy [2] or to be able to classify emotions [3]. Electroencephalography (EEG) signals measure the electrical activity of the brain and are used as an analytical tool for sorting between normal and abnormal brain function. However, making an accurate prediction using EEG data requires a lot of time and effort when done manually. By utilizing the flexibility of Machine Learning, it is possible to automate this process, resulting in faster more efficient analysis of EEG data.

1.2 Project Description

The primary objective of the Master's thesis is to develop an automated stress detection system utilizing Machine Learning classifiers and EEG signals. The selection of EEG was based on its ability to highlight altered electrical activity within the brain, making it a suitable candidate for stress detection. Additionally, the study aims to investigate the feasibility of reducing the number of electrodes used during data collection, as this can result in faster and more cost-effective recordings. Through exploring the possibility of using a reduced number of electrodes, the study will also aim to determine the optimal placement of electrodes for accurate stress detection, as well as the effectiveness of different feature extraction methods and classification algorithms. By addressing these issues, the study aims to contribute to the development of a reliable and practical stress detection system, which can be used in various contexts, including clinical, research, and industrial settings.

The dataset collected will consist of stressed state data from a group of students experiencing mental stress symptoms during their exam period, and baseline data recorded after the winter holidays. The subjects will give a statement regarding their stress level during recording, commonly called a Stress Scale (SS). Furthermore, each participant will complete the psychological inventory questionnaire called the State Trait Anxiety Inventory for Adults Form Y-1 and Y-2 (STAI-Y). The stress levels will be compared and analyzed along with the data. The raw EEG data requires pre-processing in the form of signal decomposition and processing techniques to de-noise the signal and extract the data's features. Then, the EEG data will be fed to several Machine Learning methods in order to develop a reliable classification model for achieving the aim of a computer-aided stress detection system. The final goal is to complete a program that can assist in the early detection of stress-related mental health disorders.

1.3 Related Work

In recent years there have been done several studies on the use of EEG signals for detecting and diagnosing mental stress. A review done by Katmah et. al [4] in 2021 looked into 51 existing papers that covered the topic, and found that there are many viable methods for classifying mental stress using EEG data. Many of the methods in the different papers got an accuracy $> 90\%$, and the experiments with the best results used mental arithmetic as a stressor and linear SVM, cubic SVN, KNN, and LDA [5] as classifiers. However, despite the large number of studies done on EEG signals and mental stress, there exist no inclusive guidelines about the relevance between EEG features and its extraction methods, filtering, and artifact removal [4]. There is also little-to-no described methods for EEG channel selection for stress detection. When looking into these papers it is a common trait that experiments are conducted offline, but it would be interesting to develop an online system for recognizing stress in real-time. Another point of interest would be to look into the possibility of combining EEG with PCG data for multi-modal stress detection.

Preliminary research and work were done in the project thesis by Anne Joo Marthinsen [1] in collaboration with the rest of the Mental Health-group under the supervision of Ph.D. Marta Molinas. The objective of the project was to become familiar with Electroencephalography (EEG) and Phonocardiogram (PCG) signals. The researching group was split into two: Ida Marie Andreassen and Øystein Stavnes Sletta tackled an open source PCG dataset, while Christian Sletten and Anne Joo Marthinsen received an EEG dataset. The semester was spent learning about EEG signals, pre-processing the data, and finally implementing and testing different Machine Learning classifiers with different features on all the data. Since the task at hand was quite comprehensive, and the EEG dataset included both the raw data and pre-processed data, it was decided to use a divide-and-conquer approach. Sletten used the filtered data to implement and test different ML classifiers, while Marthinsen started implementing the filtering of the raw data. This way the group could combine the work at the end of the semester, get satisfactory results, and be prepared to collect the new dataset for the Master's thesis.

Marthinsen's part of the project ended up focusing on pre-processing of EEG data in the form of filtering and artifact removal with Independent Component Analysis (ICA). She built a `Python` script for analyzing each recording and removing artifacts. All 120 recordings were filtered twice using this technique, resulting in a solid dataset to use for classification. Furthermore, after feature testing with different classification methods was done, it was found that it was possible to get an accuracy of 96.50% using Hjorth features on K-Nearest-Neighbours (KNN). Working further on this, it was discovered that it was necessary to research channel selection. In the upcoming data collection for the Master's thesis, an electroencephalogram with 8 electrodes would be used. In comparison to the open source dataset, this is a reduction of 75% in the number of electrodes. Thus, the end of the semester was focused on implementing a channel selection algorithm inspired by The Genetic Algorithm. It was found that it was possible to keep the accuracy of the classifier as high as 90.05% while reducing the number of electrodes from 32 to 8.

1.4 Outline of the Report

The report will first go through a detailed description of the theory behind the contents of the report. This includes: Mental stress, EEG, EEG filtering and pre-processing, ML classifiers, performance measures, and the genetic algorithm. Furthermore, some theory on Power Spectral Density (PSD), is included, as this is one of the new methods that will be explored. The theory section is quite comprehensive but attempts to give sufficient knowledge for the reader to understand the methods used in the thesis. Furthermore, the next chapter gives a walk-through of the methods and materials. A brief overview of the previous project is given, followed by an extensive description of the EEG data collection that was done in December 2022/January 2023. Furthermore, an overview of data exploration, preprocessing, and artifact removal with ICA is included. Then, a brief overview of the labels and classifiers that were tested is presented. Furthermore, the new methods tested are explained: EEGNet, Temporal-constrained Sparse Group Lasso-EEGNet (TSGL-EEGNet), PSD, and wavelet scattering. Further, the results are presented with figures consisting of the confusion matrix, accuracy, specificity, and sensitivity of each classifier, feature- and label set. Lastly, the results are presented and discussed, and suggestions for future work are given.

Chapter 2

Theory

2.1 Physiology of Stress

Psychological stress refers to a state of mental or emotional strain or tension caused by adverse or demanding circumstances and is a medical and psychological term. Stress is a broad term that refers to a wide range of demanding physiological and psychological influences (stressors), and to the organism's total reaction to such (resource mobilization).

Short-term stress can be positively stimulating, such as during physical labor or exercise, or to assist in an emergency, by triggering the fight-or-flight response. On the other hand, long-term stress has been shown to have various negative consequences, including muscle tension, panic attacks, inflammation in the circulatory system, increased risk for hypertension, heart attack, and stroke [6].

Measuring stress is most often based on subjective reports from individuals, preferably in the form of questionnaires that assess the degree of anxiety, tension, re-experiencing, lack of well-being, bodily symptoms, and more. However, there are also objective methods used to measure stress levels. This can include blood tests to measure stress hormones such as cortisol, prolactin, and growth hormone, urine tests to measure metabolites of norepinephrine, or physiological methods such as measuring blood flow in small skin capillaries or conducting brain examinations using electroencephalography [7] [8][9].

2.2 Electroencephalography (EEG)

2.2.1 Introduction

The human body relies on small electrical impulses to communicate between different organs and the brain through the nervous system. These electrical signals can be detected throughout the brain, even during periods of rest.

Thanks to technological advancements, Electroencephalography (EEG) can now capture and record these electrical impulses within the cerebral cortex in real time with minimal delay. As a result, EEG has become a widely used diagnostic tool in medicine for the detection of conditions such as epilepsy and various brain dysfunctions, including tumors, strokes, and other forms of damage.

In addition to its diagnostic applications, EEG has also gained recognition as a valuable tool in research and development, including cognitive neuroscience and human-machine interfaces. The ability to measure and analyze brain activity with EEG has opened up new avenues for investigating brain function and for developing innovative approaches to human-computer interaction.

2.2.2 EEG System

To capture the electrical signals of the brain, multiple electrodes (small metal disks) are attached to the scalp, either with dry electrodes or some kind of electrical conducting fluid. Each electrode transmits a signal to one of several recording channels of the electroencephalograph. This signal consists of the difference in the voltage between the electrode and a reference electrode, often placed on one of the earlobes. The rhythmic fluctuation of this potential difference is shown as peaks and troughs on a line graph by the recording channel. There exist several different ways to place the electrodes. Figure 2.1 shows a visualization of how to set up 32 EEG electrodes, using the 10-20 setup.

The 10-20 system is a widely accepted global standard used for positioning the EEG electrodes during recording. Its purpose is first and foremost to provide compatibility, repeatability, and efficiency during experimental EEG work. The name "10-20" stems from the fact that electrodes are placed with either 10% or 20% distance of the total front-back or left-right length of the subject's skull [10].

Each electrode is assigned a unique name, with the first letter indicating its location on the corresponding area of the brain. The letters translate as follows: Fp/pre-frontal, F/frontal, T/temporal, P/parietal, O/occipital, and C/central. Then, the electrodes are numbered increasingly with the distal direction from the midline sagittal plane of the skull. Even numbers are placed on the right side of the head, while odd numbers are kept on the left. See Figure 2.1 for reference. Certain electrodes are given a "Z" (zero) instead of a number. The "Z" refers to an electrode placed on the midline sagittal plane of the skull, and these are FpZ, Fz, Cz, and Oz.

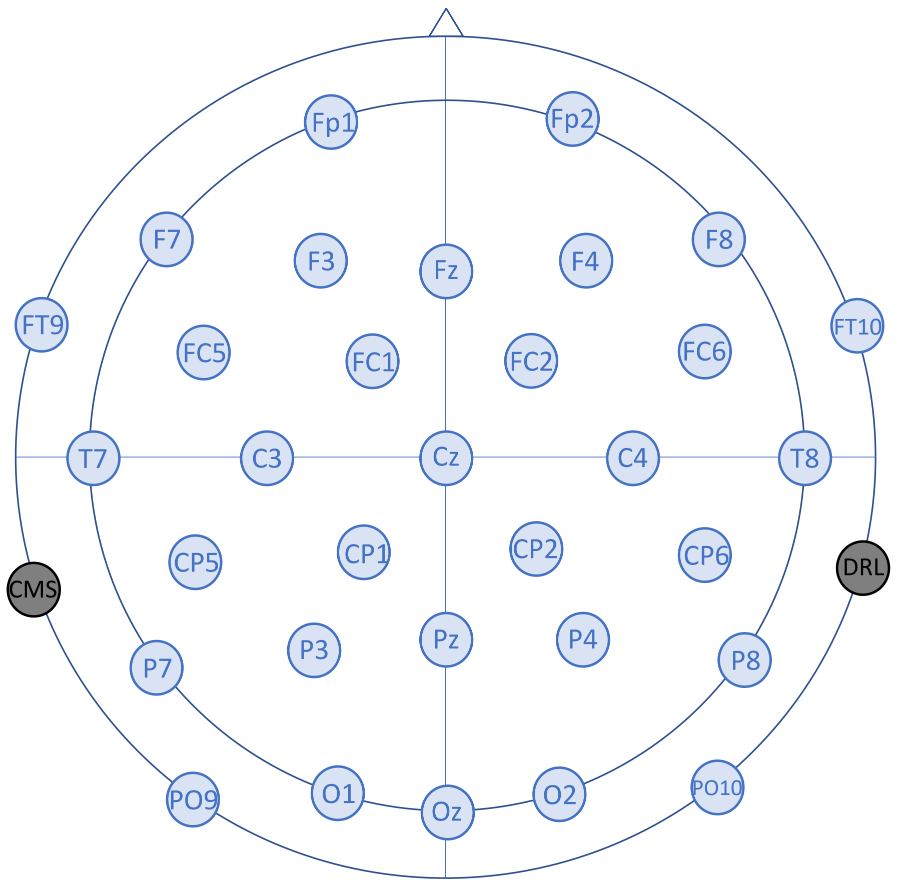


Figure 2.1: Example of setup of 32 EEG electrodes using the international standard 10-20 system. The name stands for the fact that each electrode is distanced at either 10% or 20% of the total front-back or left-right distance of the skull. Electrode lettering translates to Fp/pre-frontal, F/frontal, T/temporal, P/parietal, O/occipital, and C/central, while "Z" stands for zero, and is situated on the midline sagittal plane of the skull. The numbers increase with the distal direction from the midline, with even numbers to the right and odd numbers to the left. CMS/DRL refers to Common Mode Sense (CMS) active electrode and Driven Right Leg (DRL) passive electrode. Inspired by [11]

2.2.3 EEG Artifacts

Since EEG data is primarily gathered to diagnose a brain condition, it is usually only the signals that stem from brainwaves that are the most interesting to view. However, it is almost impossible not to record some form of noise or other signals. When talking of EEG data collection, the word *artifact* therefore often follows. Artifacts are noise originating from sources other than brain activity. Examples are eye blinks, eye twitches, muscle twitches, etc. It is crucial to remove these, as this will increase the probability that the models and results are built on actual brain activity [12]. Recognizing artifacts can be a difficult task, especially for newcomers, as it is one of those things that simply require experience. However, there are *some* recognizable traits that are easy to spot even for beginners.

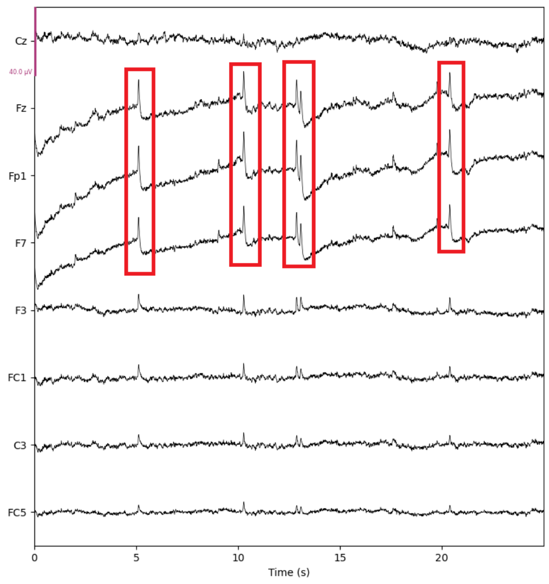
Eye blinks are commonly present in EEG raw data, unless the subjects are asked to keep their eyes closed. Eye blinks are always the most present in the frontal and pre-frontal electrodes, where they show up as irregular spikes, often with up to 10x the magnitude as the rest of the signal. For a typical blink, the electrical potential sharply increases, then decreases, over a period of about 250-300 ms [12]. Blinking also varies a lot depending on the subject: some blink often, while others blink less, some blink with much tension while others blink softer. Figure 2.2a shows an example of how blinking may show up in raw EEG data. The relatively large magnitude of the blinking often influences other channels as well, as can be seen in the other non-frontal electrode, C3, that is included in Figure 2.2a.

Eye twitches or **glances**, also known as **saccades**, exhibit similar characteristics to eye blinks in EEG data. Eye movements will also be most prominent in frontal and pre-frontal electrodes, and have no distinct pattern or rhythm. However, unlike eye blinks, they often appear to have a square-like shape. It can often be observed that one channel increases, and another decreases in the same fashion with saccades. Figure 2.2b shows an example of how eye twitches and glances may show up in raw EEG data.

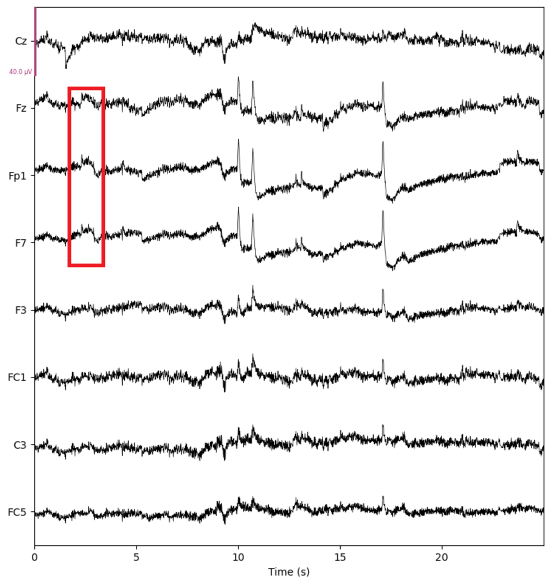
Muscle contractions, on the contrary, manifests as high-frequency activity, often over a short period of time. The contractions are typical of frequencies 20-40 Hz. Small and/or brief contractions will oftentimes not be a problem, but if the subject is tense throughout the recording, it can disturb the data. These kinds of artifacts are very hard to distinguish from actual EEG data. Thus, we will have to dig deeper in order to be able to find and remove them. This will be further discussed in section 2.3.

Heartbeats may be captured by the EEG recordings. These will show up with a rhythmic beat, with about 1-2x the magnitude of normal EEG data. It can thus be hard to distinguish heartbeat artifacts. This will also be further discussed in section 2.3.

Poor connections between the scalp and electrode(s) can happen during recording and is an unfortunate situation. This can happen with the slightest movement of the subject's head position and is hard to avoid. Bad connections result in the signal values increasing rapidly, often influencing all channels. Some researchers may choose to remove the whole time period, marking it as "bad" so that it is not included in further analysis. However, it will also be discussed how the source can be removed in 2.3.



(a) Blinking artifact



(b) Saccade artifact

Figure 2.2: Examples of how artifacts show up in raw EEG data, marked with red boxes. Only 8 electrodes are shown for simplicity. Figure a) shows an example of a typical blinking artifact: short, irregular spikes, with 10x the magnitude as the rest of the data, with the largest magnitude at the frontal electrodes. We can also see that the blinking influences the non-frontal electrode, C3, as well. Figure b) shows a typical saccade artifact: short square-like signal jumps, with no pattern or rhythm. Repurposed from [1].

2.3 Signal Filtering and Pre-processing

2.3.1 Signal Filtering

Electroencephalography (EEG) signals in humans mainly consist of frequencies ranging from 1-30 Hz, with some studies suggesting the presence of higher frequencies carrying meaningful information. However, the consensus among researchers is that the most informative range lies within 1-30 Hz[13]. Head movements and electrode shifts contribute to the presence of low-frequency noise (within the 0-1 Hz range), manifesting as slow fluctuations. On the other hand, high-frequency noise will manifest as rapid fluctuations in the EEG data and can be caused by, for example, electromagnetic interference and muscle contractions of the face and neck [13].

To preserve potentially relevant information, experts suggest using a band-pass filter on EEG data with cut-off frequencies around 1 and 40-50 Hz[14]. This ensures that no information-carrying frequencies are removed, as filters are often not ideal and will thus have an imperfect cut-off. Furthermore, filtering at 50 Hz with a band-pass or notch filter eliminates the utility frequency caused by the power grid, which is irrelevant to the analysis.

Moreover, one can apply a smoothing filter, such as the Savitzky-Golay filter, to the data. This filter smooths the data by utilizing a technique that involves local least-squares polynomial approximation, which preserves the waveform's shape and height of peaks [15]. These kinds of properties are crucial for removing noise from psychological signals without corrupting or removing valuable data.

2.3.2 Artifact Removal with ICA

The use of Independent Component Analysis (ICA) as a method for artifact removal from signals is quite popular. ICA works by separating a multivariate signal into its additive sources, which can be seen as analogous to unmixing paint. A classic example of ICA's application is *The Cocktail Party Problem* [16]: Consider a cocktail party where several groups of people have overlapping discussions. There might also be music playing from a piano, as well as noise coming from the kitchen, etc. The goal is to separate the various sources from each other using several recording devices placed around the room. Given at least as many recordings as the number of sources, ICA can separate each conversation, instrument, and background source from each other. This will, for example, allow us to extract a clear recording of the piano playing in the background.

In EEG data analysis, ICA can be used to remove uninteresting sources, such as eye twitching, head movements, or bad electrode connections. The method assumes that the recorded time series is a spatially stable mixture of the activities of independent cerebral and artifactual sources and that the summation of potentials is linear at the electrodes. These assumptions are reasonable for EEG data [17][18]. Given its capability to exclude unwanted sources and effectively remove noise, ICA is a well-suited technique for artifact removal in EEG data.

2.3.3 Features

Datasets are most often large and complex, with multiple recordings and participants. Therefore, using some type of feature extraction instead of the complete dataset is often preferable. Features are measurable properties of the data, and selecting informative, independent features is critical for classification, pattern recognition, and regression algorithms. Good features can help filter out irrelevant information and reduce dimensionality, leading to more manageable and readable datasets. Examples of features are:

- **Time series features:** Analyzing the signal's time series features can reveal patterns and help build a predictor. Examples of time series features are:
  - Peak-to-peak amplitude
  - Variance
  - Root-Mean-Squared-Error
- **Fractal features:** The fractal dimension is a ratio providing a statistical index of complexity. This analysis compares the variation in detail in a pattern with the scale at which it is measured. Examples of fractal dimensions are:
  - Higuchi Fractal Dimension
  - Katz Fractal Dimension
- **Entropy features:** As a general rule, entropy represents some measure of *information*, *surprise*, or *uncertainty* about an unknown variable. Examples are:
  - Approximate Entropy
  - Sample Entropy
  - Spectral Entropy
  - SVD entropy
- **Hjorth features:** Hjorth features are common features for feature extraction of EEG signals. The parameters are normalized slope descriptors (NSDs) called Activity, Mobility, and Complexity.
- **Frequency band features:** A frequency band is a specific range of frequencies with a defined upper and lower frequency limit. We can choose the range ourselves depending on the signal attributes, and compute their powers to use as features. When analysing EEG data, the common frequency bands delta (<4 Hz), theta (4 - 8 Hz), alpha (8 - 12 Hz), beta (13 - 30 Hz) and gamma (>30 Hz) are often utilized.

2.3.4 K-Fold Cross Validation

K-fold cross-validation is a resampling technique used to evaluate the performance of a machine-learning model, used especially on a limited dataset. It involves dividing the dataset into k subsets or folds, training the model on k-1 folds, and evaluating it on the remaining fold. This process is repeated k times, with each fold serving as the test set exactly once. The results are then averaged across the k-folds to obtain a more accurate estimate of the model's performance. K-fold validation can help prevent overfitting, as it allows for a more robust assessment of the model's generalization performance.

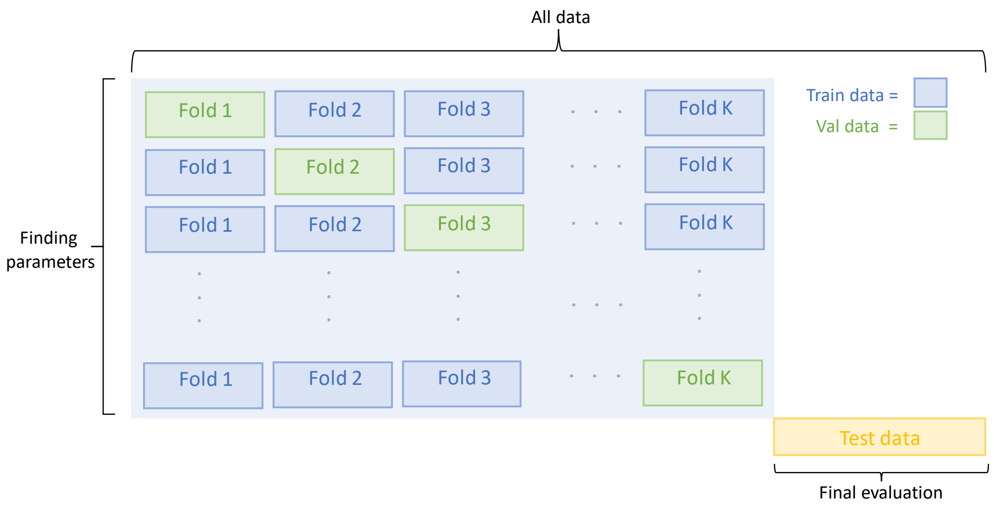


Figure 2.3: Visualization of k-fold cross validation. The training dataset is divided into k subsets or folds. Then, the model is trained and validated across all folds, while exploring and trying to obtain the optimal hyperparameters. Lastly, the final model is evaluated on the test data to assess the model's performance on brand-new data. Inspired by [19].

Stratified k-fold cross-validation is a variation of k-fold cross-validation that takes into account class imbalance in the dataset. In stratified k-fold validation, the data is split into k-folds in such a way that each fold has approximately the same proportion of samples from each class as the overall dataset. This ensures that each fold is representative of the overall dataset and helps prevent bias towards the majority class.

2.4 Machine Learning (ML) Classifiers

This section will include a short introduction to Machine Learning basics, as well as some Machine Learning methods. It should be sufficient for a newcomer in the field to familiarize themselves with the methods relevant to this project. Experienced readers may choose to skip this introductory section.

Machine Learning is a branch of Artificial Intelligence (AI) that aims to enable machines to learn like humans. This is achieved through experiential learning, where machines are rewarded for good choices and sometimes penalized for bad ones, ultimately enhancing their knowledge and accuracy over time. Applications of ML methods are many: Common examples are handwritten letter recognition, speech-to-text generators, image classification, medical diagnosis, and predictive analysis.

In many forms of research, the goal is to train a Machine Learning *classifier*. The goal of a classifier is to find some *model* or *mapping* from input data to distinct *classes*. An example of this is spam filtering. When receiving an email there are underlying classifiers that determine whether or not the incoming message is spam. The classifier examines specific *features* within the email, such as certain phrases or wordings, or the format of the sender's address. To train a spam/non-spam classifier, we would start by collecting a large dataset of labeled emails, where each email is identified as either spam or non-spam. This is what's called a *labeled dataset*. This form of classification is thus called *supervised learning*, signifying that it needs the initial data to be labeled in order to find a way to recognize spam from non-spam itself.

The dataset is then typically divided into three parts: the training set, used to fit the initial model; the validation set, used for prediction and tuning the classifier's (hyper)parameters; and the test set, which contains completely new data to assess the classifier's robustness. Now, when a new e-mail is received, the classifier can (hopefully) correctly delete the spam and keep non-spam, saving the user the trouble of doing this themselves.

2.4.1 K-Nearest-Neighbours (KNN)

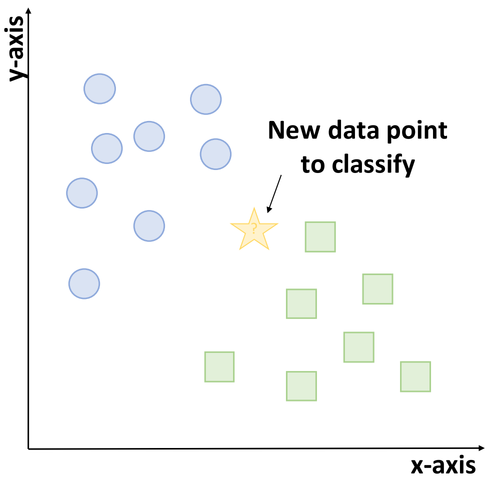
The K-Nearest-Neighbours (KNN) classifier is a non-parametric, supervised-learning classifier. That is, it is a classifier that does not assume a specific model structure for the mapping and is trained with labeled data. KNN uses the *proximity* to existing classes to classify new data points. The KNN classifier uses the k closest data points of each classifier and computes a proximity metric in order to decide which class the new data point belongs to [20]. Performance can be enhanced by tuning parameters such as the number of neighbors, denoted as k, the distance metric, denoted as d, and the leaf size. The reader is advised further reading if they find this interesting. Figure 2.4 shows a simplified visualization of how the KNN classifier works in general.

2.4.2 Support Vector Machines (SVM)

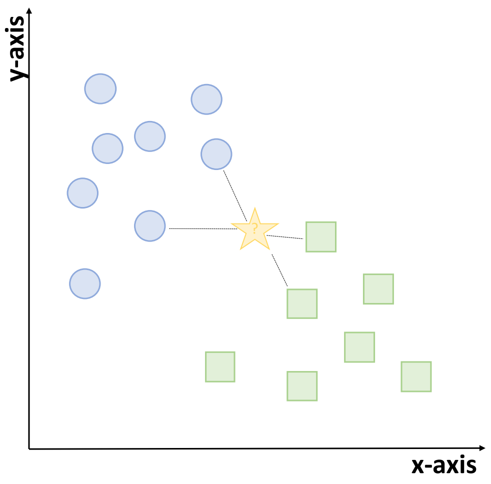
The Support Vector Machines (SVM) is a supervised learning algorithm widely employed for classification tasks. Given a labeled dataset, with each sample belonging to one of two categories, the classifier reviews the data and maps each sample as a point in an n-dimensional space, where n represents the number of input features. The objective is to separate the categories by an *optimal hyperplane*, which maximizes the distance between the categories. The peripheral data points closest to the other category are used as the *support vectors*, as they significantly influence the configuration of the hyperplane[21]. In the context of SVMs, the "margin" refers to the area between the decision boundary, which separates the different classes. The distance between the decision boundary and the training data points is called the "street width."

The regularization parameter serves as a hyperparameter that controls the complexity of the model. It determines the trade-off between the size of the street width and the accuracy of the model. A large regularization parameter signifies that the model will have a smaller street width and will try to correctly classify as many of the training data points as possible. This can lead to overfitting. A small regularization parameter, on the other hand, allows for a larger street width and is thus open for some misclassification of the training data. This can help to prevent overfitting and can improve the generalization performance of the model [22].

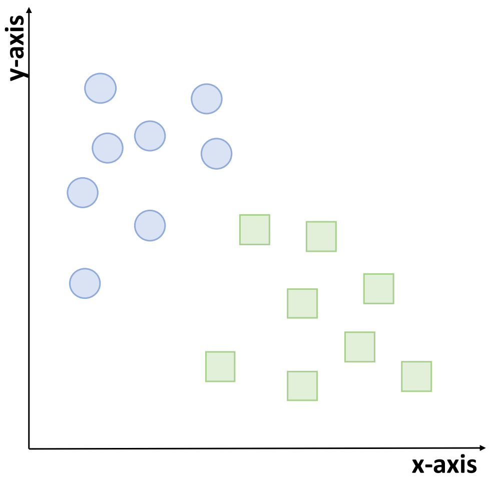
Figure 2.5 shows a simple visualization on the steps to a SVM.



(a) New datapoint to classify is introduced.

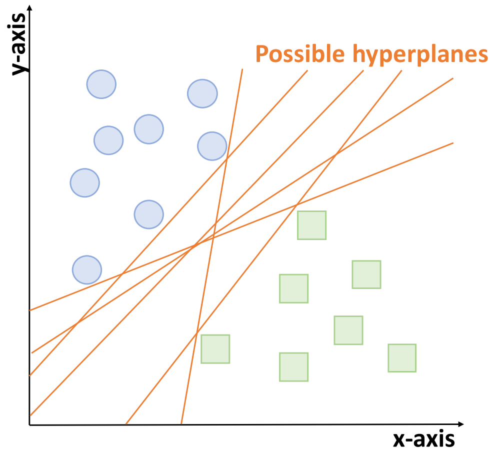


(b) The classifier finds the $k = 2$ closest neighbours.

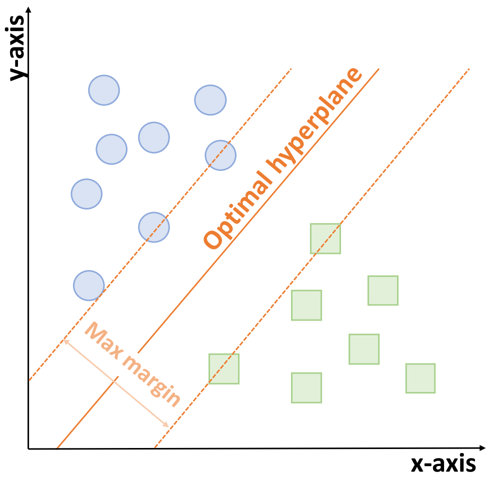


(c) The datapoint is classified with the closest group.

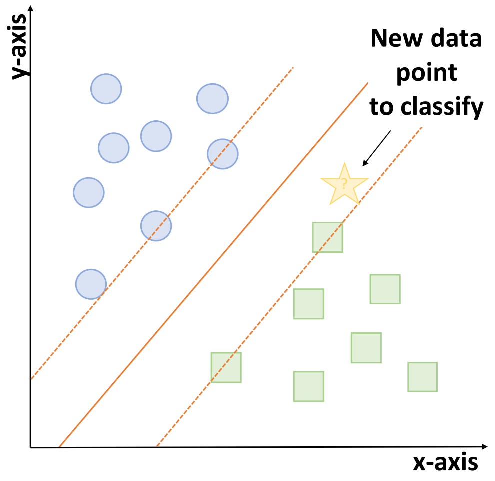
Figure 2.4: A simple visualization of how the KNN classifier works. The classifier uses the k closest data points of each classifier and computes a proximity metric (Euclidean in this case) in order to decide which class the new data point belongs to. In the example, we see that the distance to the closest 2 green squares is the smallest, and thus the new point is categorized with them. Modified from [1].



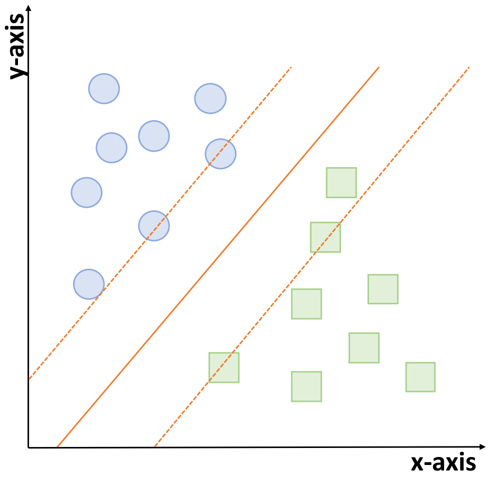
(a) The classifier looks for possible hyperplanes that divide the classes.



(b) SVM finds the optimal hyperplane and maximum margins.



(c) New data point to classify is introduced.



(d) Since the new data point is positioned closest to the green squares margin, it is classified as a green square.

Figure 2.5: A simple visualization of how the SVM classifier works. The classifier finds the optimal hyperplane, which is the best plane that divides the two classes with the most distance between them, as well as the maximum margins that describes the boundary of the classes. When a new data point is introduced, it is common to classify it with the class whose margin is the closest to the data point. Modified from [1].

2.4.3 Neural Networks

Neural networks are Machine Learning algorithms modeled after the human brain's structure and function. They comprise many interconnected nodes, called neurons, which process and transmit information. These networks are trained to recognize patterns in data by adjusting the weights of the connections between neurons, a process known as learning.

Convolutional Neural Networks (CNNs) are a type of neural network that is widely used for image and video analysis. Unlike traditional neural networks, which process the input data in a linear manner, CNN use a process called convolution to filter the data and identify patterns. The feature maps generated by the convolutional layer are then passed through a series of additional layers, including pooling layers and fully connected layers, to produce the final output. Pooling layers are used to reduce the dimensionality of the feature maps, while fully connected layers use the output of the previous layers to classify the input data.

One of the key benefits of CNNs is their ability to learn spatial invariance. This means that CNNs are able to recognize patterns in images, regardless of their position or orientation within the image. This is achieved through the use of pooling layers, which reduce the sensitivity of the network to small variations in the input data[23].

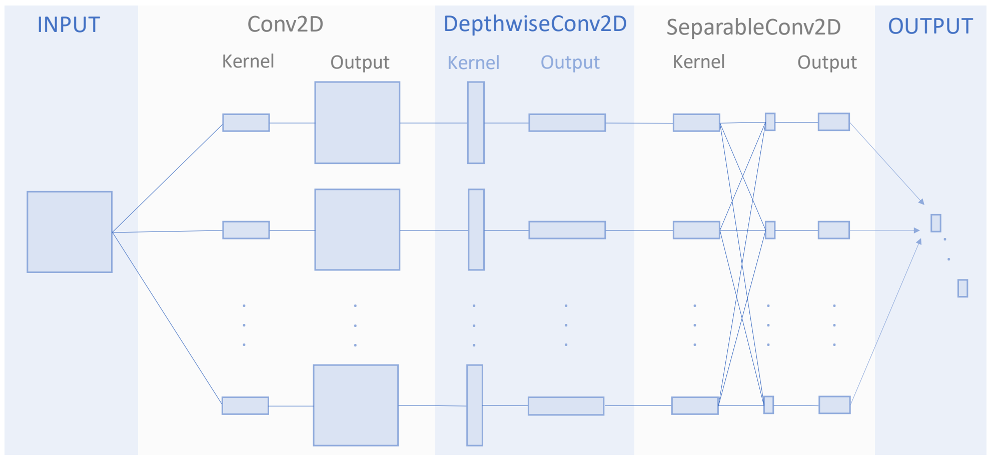


Figure 2.6: Simple visualization of the CNN, inspired from [23].

2.5 Power Spectral Density (PSD)

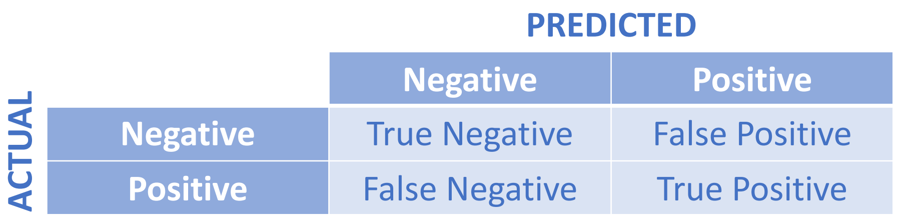
The Power Spectral Density (PSD) is a measurement that characterizes the distribution of power (or energy) content across different frequencies in a signal. It is commonly used to analyze the frequency components of signals, particularly in fields such as signal processing, communication systems, and neuroscience [24]. In signal processing and related fields, PSD is often employed to characterize broadband random signals [24]. To ensure comparability, the amplitude values in the PSD are often normalized based on the spectral resolution used to digitize the signal, representing the level of detail in the frequency analysis. The calculation of PSD frequently involves utilizing Fourier transforms, a mathematical tool that enables the examination of a signal's frequency components. By applying Fourier transforms, the power distribution across the frequency spectrum can be determined, allowing for detailed frequency analysis.

In neuroscience, PSD plays a vital role in the analysis of brain signals, such as Electroencephalography (EEG) and magnetoencephalography (MEG) signals. Researchers employ PSD to investigate the frequency content of these signals and gain insights into neural activity [25]. While the reference to Demuru et al. [25] highlights an example of PSD application, numerous studies in neuroscience utilize PSD to better understand brain dynamics and cognitive processes.

2.6 Performance Measures

In order to measure the performance of statistical, medicinal, and/or Machine Learning classification, a *confusion matrix* is often utilized. A visualization of the classic confusion matrix is given in Figure 2.7. The following gives a brief explanation of the terminology used:

- **True Positive (TP)**: We correctly predict a positive. For example when a patient *is* sick from COVID and the PCR test predicts sickness.
- **True Negative (TN)**: We correctly predict a negative. For example, when a patient *is not* sick from COVID and the PCR test predicts no sickness.
- **False Positive (FP)**: We falsely predict a positive. For example, when a patient *is not* sick from COVID and the PCR test predicts sickness.
- **False Negative (FN)**: We falsely predict a negative. For example when a patient *is* sick from COVID and the PCR test predicts no sickness.



		PREDICTED	
		Negative	Positive
ACTUAL	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

Figure 2.7: Confusion matrix. Repurposed from [1].

The more TP and TN we achieve, the better performance. The confusion matrix is often used to measure other performance indices, like Recall, Precision, Specificity, Sensitivity, Accuracy, and AUC-ROC curves [26]. In medicinal research, like screening, *accuracy, sensitivity* and *specificity* are often used.

- **Accuracy:** How many samples are classified correctly.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- **Specificity:** How many samples are correctly classified as negative. A highly specific test means few false positive results.

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

- **Sensitivity:** How many samples are correctly classified as positive. A highly sensitive test means few false negative results.

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

The screening process involves applying a medical test or procedure to a person who does not yet have symptoms of a particular disease, in order to determine their likelihood of contracting it. It is not possible to diagnose an illness from the screening procedure alone. Positive results from the screening test will require further evaluation with subsequent diagnostic tests. In most screening processes, it is therefore often favorable to prioritize sensitivity over specificity. This ensures that all at-risk patients are picked up by the process early on.

2.7 The Genetic Algorithm

The genetic algorithm is a search and optimization technique that simulates the process of natural selection. One of the advantages of the genetic algorithm is that it can handle both constrained and unconstrained problems, which makes it suitable for a wide range of applications. Additionally, the algorithm is flexible and can be customized to accommodate different types of problems and objectives.

The algorithm starts by creating an initial population of candidate solutions. The *fitness'* of the individuals are then evaluated based on a specified objective function. The fitter individuals are then selected as the elite and are kept for the next generation. From the elite, a subset is chosen as parents. The parents mate and produce *offspring*. The offspring inherit genes/features from their parents. Then, some or all of the offspring may go through *mutations* before the genes are passed on to the next generation. The hope is that some of the offspring will become fitter individuals than their parents, thus simulating natural selection. Then, the process is repeated with the new population consisting of the elite and the offspring. This process is repeated as many generations one sees fit or until another stopping criterion is met, like a fitness threshold.

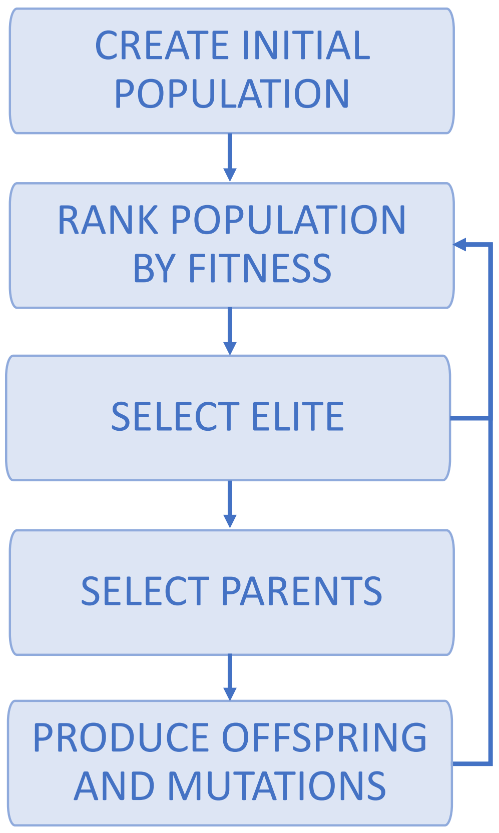

CREATE INITIAL POPULATION
RANK POPULATION BY FITNESS
SELECT ELITE
SELECT PARENTS
PRODUCE OFFSPRING AND MUTATIONS

Figure 2.8: Genetic Algorithm flowchart. Repurposed from [1].

Chapter 3

Materials and Method

Note: Only the relevant code has been included in Appendix A. The complete codebase can be found on our GitHub.

3.1 Project Thesis Work

The work done in the project thesis by Marthinsen [1] was done as preliminary research for this thesis. The project used the SAM40 dataset for analysis [11][27] and included filtering and artifact removal of the EEG data using ICA, finding useful features with the python package `mne_features` and finally building suitable classifiers with the KNN, SVM and MLP methods. The resulting classifier yielded an accuracy of 96.50%, a sensitivity of 97.18%, and a specificity of 95.89%.

Some of the methods used in the project thesis will be repeated or reused in this work. This thesis can thus be considered a continuation of the work done in the project[1], and the relevant contents of the project will be repeated throughout the thesis.

3.2 EEG and PCG Data Collection

It was established early in the process that one of the goals during the master's semester was to collect, preprocess and utilize our own stressed/non-stressed dataset. The hope was to further strengthen the hypothesis that EEG data can pick up on stress, as was explored in the brief research in the project thesis [1]. The data was to be collected using EEG and PCG signals. Two separate recordings were done, using healthy students as test subjects.

3.2.1 Design of Experiment

The goal of the experiment was to collect stressed and baseline data from young, healthy subjects. The natural stressor chosen was upcoming exams and deadlines for major projects. Thus the first recording was done between November and December of 2022, a period when many students feel elevated stress levels due to approaching exams. The second recording was carried out after the holidays during January and February of 2023 after the subjects had returned from the winter holiday.

Each subject was asked to show up at a designated time slot for recording. Before starting, all subjects were asked to read and sign a consent form. The form is attached in section C.1. After consent where given, the subjects were equipped with an EEG-cap and a digital stethoscope that both streamed to a common streaming service for synchronization. All subjects were asked to rate their stress level on a scale of 1-10 as well as fill out a State Trait Anxiety Inventory for Adults Form Y-1 and Y-2-form before and after recording. The State Trait Anxiety Inventory for Adults Form Y-1 and Y-2-form is widely used in clinical screening to measure psychological stress. The subjects were recorded for two 5-minute periods: one recording sitting still and one recording where an arithmetic stressor was introduced. Figure 3.1 shows an overview of the experiment protocol, and Figure 3.2 shows the timeline of the experiment.

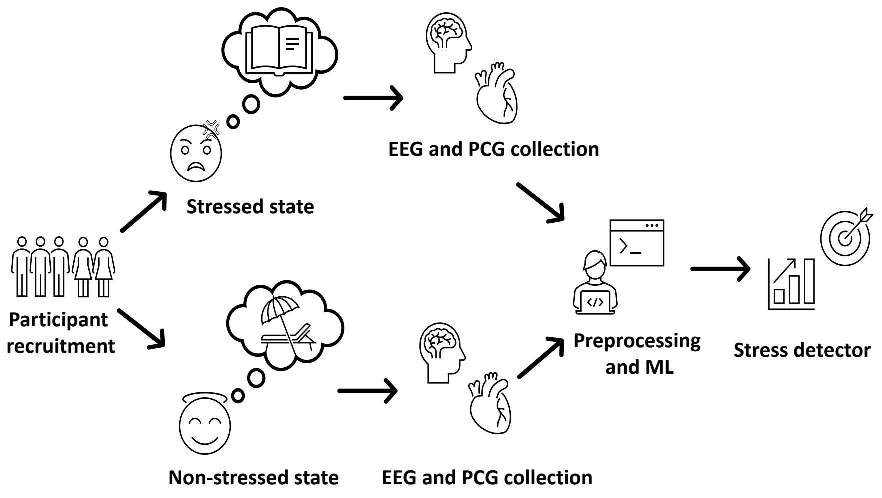


Figure 3.1: Overview of the experiment protocol. Participants are recorded during a stressed state (prior to an exam), and during a non-stressed state (after holidays). EEG and PCG data are recorded, reprocessed, and used in several Machine Learning methods. The final goal is to tune the hyperparameters of the classification in order to produce a reliable stress detector. Modified from [1].

All selected subjects received an e-mail thanking them for their interest in participating in the study. They were asked to pick a time slot from a calendar attached to the mail and to show up with clean hair and a t-shirt to wear during the session. Each recording session was originally set up as 30 minutes, but where extended to 45 min later on, as the setup took longer than anticipated. Once the subject had arrived, they were asked to fill out the consent form (attached in section C.1), as well as a State Trait Anxiety Inventory for Adults Form Y-1 and Y-2 form to measure their anxiety level. Subjects were asked to either have the stress of the exam period, or the relaxation of their recent winter holiday, in mind when answering. The information on the experiment was then repeated: Mainly how the experiment would be set up, how the recordings would be collected, and that the subjects could withdraw from the study at any point without consequences.

Once consent was given, the set-up began. Subjects were seated in a chair while the EEG-cap was placed on their head. The 8 electrodes were placed like the best-performing channels in section 4.1, visualized in Figure 3.3. Each electrode location site was cleaned with isopropyl alcohol and a Q-tip. Some electrical conducting paste was applied to the reference electrode to ensure good electrical contact. The reference electrode was then fastened to the right earlobe with some skin-friendly tape, as well as a clothespin to ensure it was secured tightly. The hair at each electrode site was pushed aside. Some electrical conducting paste was also applied to the scalp to ensure good contact with the electrodes. Mentalab's software was used to measure the electrode impedances. Impedances under 160 kΩ are essential to secure quality recordings. If the impedances were not under the desired 160 kΩ, more conducting paste was applied to the scalp, and the electrodes were re-adjusted. If all electrodes had high impedances at this point, the problem often lied in the reference electrode, which needed to be re-attached.

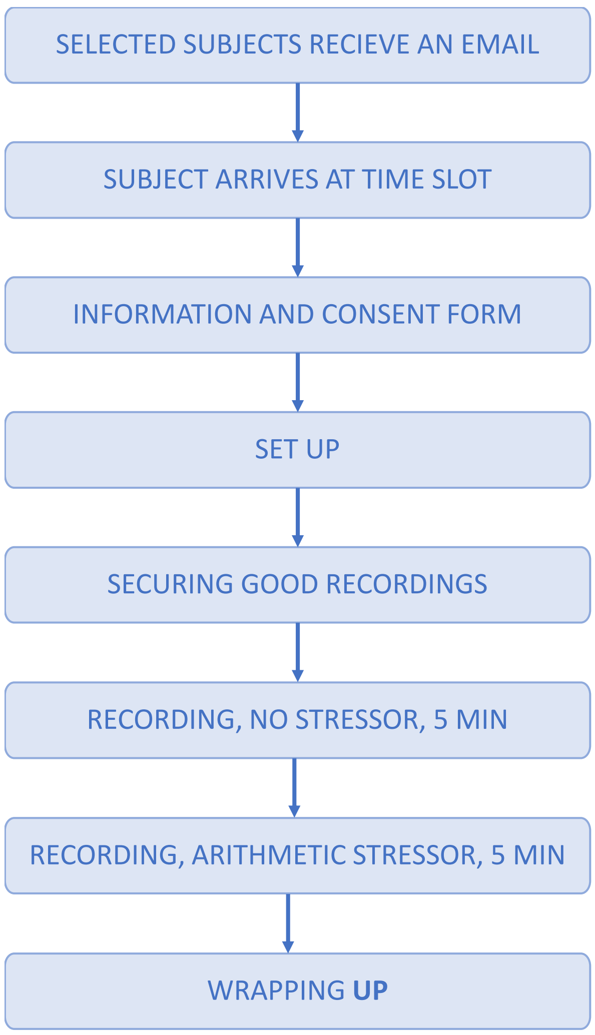

SELECTED SUBJECTS RECIEVE AN EMAIL
SUBJECT ARRIVES AT TIME SLOT
INFORMATION AND CONSENT FORM
SET UP
SECURING GOOD RECORDINGS
RECORDING, NO STRESSOR, 5 MIN
RECORDING, ARITHMETIC STRESSOR, 5 MIN
WRAPPING **UP**

Figure 3.2: Timeline of experiment execution. Repurposed from [1].

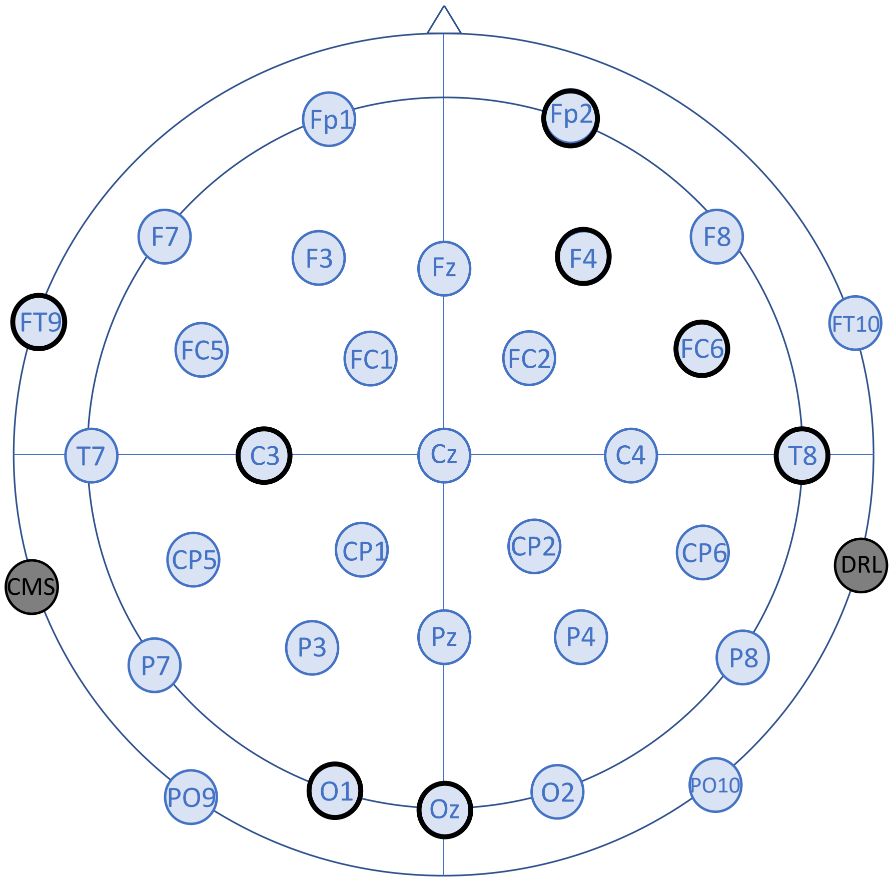


Figure 3.3: The best 8 electrode placement for capturing mental stress according to the results in section 4.1. Inspired by [11].

The PCG was attached to Erb's point and fastened with a buckle strap placed diagonally across the upper body. Erb's point is the auscultation location for heart sounds and heart murmurs and is thus one of the best points for recording PCG. Figure 3.4 shows an illustration of the common chest landmarks, where Erb's point is localized at 3). The PCG was fastened using a buckle strap, and the mic input was set to maximum. The Eko-app was used to double-check if the incoming signal had satisfactory amplitude. Figure 3.5 shows a picture taken during the recording of a subject (with their consent). The EEG cap is on and the fastening of the reference electrode is visible. The PCG buckle strap is also visible, which is used to hold the PCG in place.

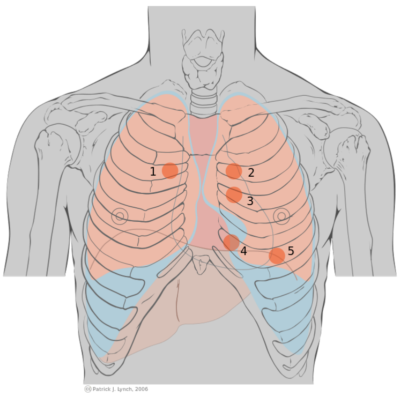


Figure 3.4: Cardiac landmarks: 1) aortic area, 2) pulmonic area, 3) Erb's point, 4) tricuspid area, 5) mitral area. From [28].

Once all the equipment was set up, two 5-minute recordings of EEG and PCG data were collected. During the first recording, the subjects were asked to sit still and to either think of their upcoming exams or their recent holiday. Afterward, they were asked to rate their anxiety level during the recording on a scale from 1-10. Then, the arithmetic stressor was introduced. The subjects were presented with arithmetic statements on a computer screen, like:

$$2 + (2/2) + (2x2x2)/2 = 8.$$

The subject was then instructed to calculate the statement in their head and press "T" if the statement was true and "F" if it was false. Subjects were asked to keep the rest of their body as still as possible and to not talk to themselves out loud as this would interfere with the recordings. After the 5 minutes were up, the subject was asked to rate their stress level from 1-10 again, as well as fill out yet another STAI-Y form, this time with the arithmetic stressor in mind.

The PCG strap was loosened and removed along with the stethoscope. The clothespin and tape were removed from the earlobe and the EEG reference electrode was cleaned. Then, the EEG cap was shimmied off the head of the subject, and all electrodes were cleaned with isopropyl alcohol and a Q-tip. The electrode sites were cleaned with a wet tissue in order to remove most of the electrical conducting paste from the subjects' hair. The subjects were thanked for their participation and given a gift card for their time.

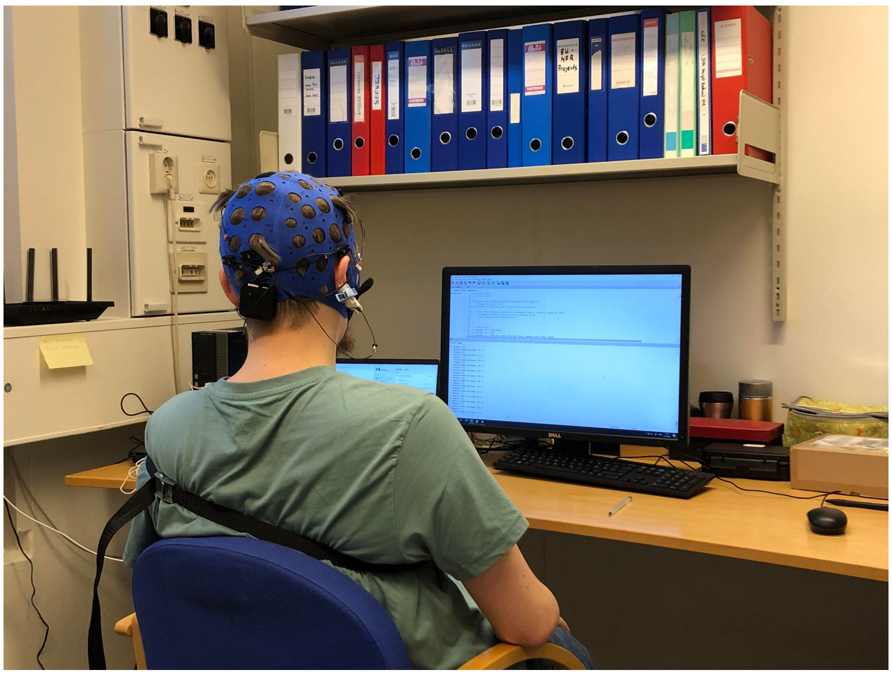


Figure 3.5: Set up example. Subject is wearing the EEG cap, and the fastening of the reference electrode on the right earlobe is visible. The PCG is placed under the t-shirt on Erb's point and fastened using the black buckle strap.

3.2.2 Participant Recruitment

Recordings were done on 28 subjects (16 male/12 female, mean age 23 with standard deviation 2). All subjects were healthy, yet stressed, students in the middle of the exam period in the winter of 2022. All subjects reported that they were in good health, without any diagnosis of cardiac-, neurological, or mental health disorder. All subjects were required to sign a consent form including a brief overview of the experiment, and a statement saying their participation was voluntary and that they could withdraw from the study at any time. Figure 3.1 shows an overview of the experiment protocol. The full consent form is attached in section C.1.

3.2.3 Equipment

The dataset was collected using an 8-electrode EEG cap from Mentalab Explore, depicted in Figure 3.6, as well as an Eko DUO ECG + Digital Stethoscope from EkoDuo. The EEG cap was set to sample at a rate of 250 Hz, while the digital stethoscope was sampling at a rate of 22 050 Hz. In order to get quality recordings from the EEG, isopropyl alcohol, electrical conducting gel, Q-tips, skin-safe tape, and a clothespin were used. The PCG was fastened and tightened using a buckle strap. Three computers were utilized for streaming EEG data (using Mentalab's software Explorepy), PCG data (from AudioCapture), and markers when the subject interacted with the arithmetic test. Markers were generated using a script in Psychopy, and the recordings were synced using Lab Streaming Layer.

Figure 3.6: The EEG cap, electrodes, and cables from Mentalab used during data collection. Photos from the Mentalab website.

3.2.4 Channel Selection

As the SAM40 dataset analyzed in the project thesis uses a 32-channel Emotiv Epoc Flex gel kit, while the Mentalab EEG cap only uses 8 adjustable electrodes, there was a necessity to research channel selection. The goal was to establish whether or not some electrodes could record a stressed state more than others, and if it is possible to reduce the number of required electrodes without (too much) loss in accuracy. An overview of previous papers researching the same issue is attached in Table 3.1. Most of the existing papers reviewed in this study have used the standardized 10-20 system for electrode placement, with an arbitrary number of electrodes, when recording and analyzing EEG data. In contrast, our approach involved employing a systematic search using the Genetic Algorithm. An implementation of the Genetic Algorithm was found and modified to match the problem description. The original code can be accessed here, and the modified code is attached in subsection A.0.1. In a separate Jupyter Notebook, different parameters for the genetic algorithm were experimented with, and the algorithm was tested multiple times.

Whenever the code runs, 15 random channel selections are initialized. Each channel selection consists of 8 channels randomly picked from the 32 possible channels. Then, the code iterates through each channel selection. For each iteration, the matching subset data is extracted from the EEG dataset. Then, the best-performing combination from the project thesis (Hjorth features of double ICA filtered EEG data run on KNN-classifier) [1] is run on the subset, which in turn returns accuracy, sensitivity, and specificity. These values are then fed into a weighted sum, which gives a measure of the performance of each subsection. Trials were run with all weights equal and with a slight increase in accuracy.

In each generation, the five best-performing channel selections were selected for crossovers. This signifies making new selections that inherit channels from the best-performing channel selections. The new selection's first four channels are picked randomly from one selection, and the last four from another. Each of the five-channel selections makes one crossover with all other selections, producing 15 new selections in total (4+3+2+1=10). These will be included in the next generation. Thus the following generation will also consist of 15 people: The five best-performing channel selections from earlier plus the 10 children. This process is repeated for 10 generations in order to find the best-performing channel subset. Several runs of the code were conducted to ensure a better solution wasn't missed if the algorithm converged to a local minimum.

The full code is included in subsection A.0.2 and subsection A.0.1, and the results will be discussed in chapter 4.

Table 3.1: EEG channels used in Stress Detection Studies and their methods for selecting them.

EEG channels selected	**Method used**	**Accuracy**	**Paper(s)**
AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4	No method described, other than following the 10-20 system	96.00%	[29]
F3, F4, T7, C3, C2, C4, T8, P3, P4	No method described, other than following the 10-20 system	90.00%	[30]
Fp1, F3, F7, Fz, Fp2, F4, F8	No method described, other than following the 10-20 system	95.00%	[31]
TP9, FP1, FP2, TP10	No method described	85.6%	[32]
F4, FP2, F8, Fz, F3, F7	Selected based on statistical analysis of EEG electrodes	91.7%	[33]
Fp1, Fp2	No method described, other than following the 10-20 system	86.66%	[34]

3.2.5 Labels

Two sets of labels were collected during recording: a State Trait Anxiety Inventory for Adults Form Y-1 and Y-2 (STAI-Y) and a Stress Scale (SS) score for each recording. The SS scores range from 1-10. As STAI-Y forms 1 and 2 measures levels of *state*- and *trait*-anxiety, respectively, only the STAI-Y1 score was calculated and used. The resulting STAI-Y scores range from 20-80. The stress scores of each subject are visualized in Figure 3.7.

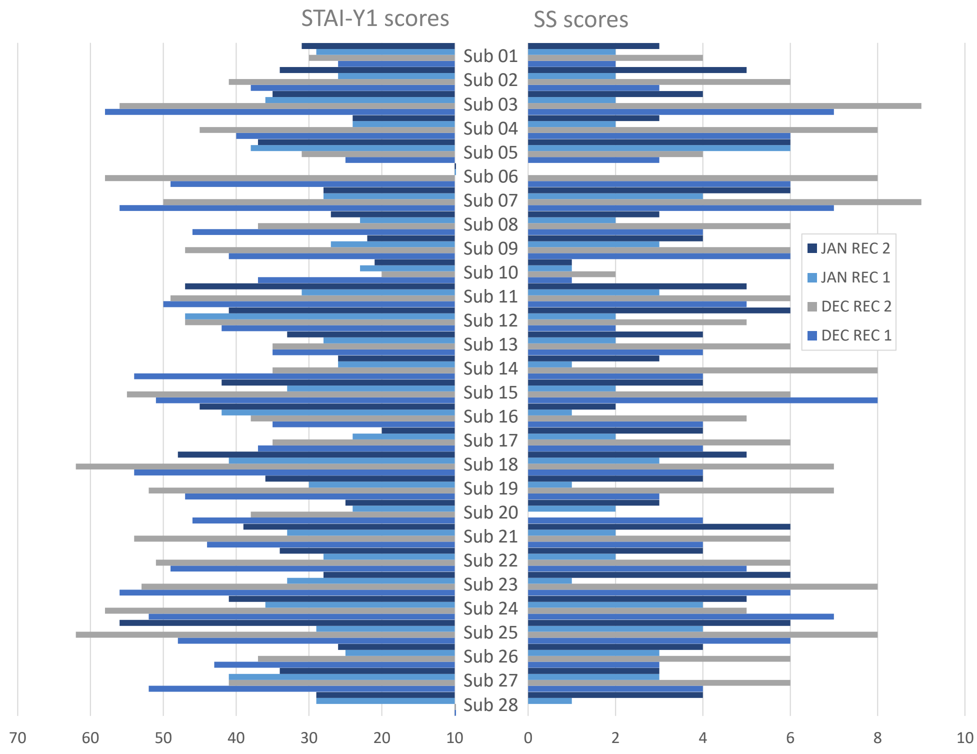


Figure 3.7: Scores for each participant, with STAI-Y scores (which range from 20-80) on the left and SS scores (which range from 1-10) on the right.

In order to convert these scores into labels, we chose specific thresholds for each set of scores. For the SS-scores, the subject's recording was labeled as *non-stressed* if they stated a stress level between 1-3, and *stressed* if they stated a stress level between 7-10. For STAI-Y scores, the stress level is defined to be none or low if the score is between 20-37, and high if the score is between 45-80 [35]. The resulting labels are shown in Figure 3.8, with STAI-Y labels on the left and SS labels on the right.

It is possible to eliminate subjects experiencing moderate stress and use a binary classification approach over a multi-classification one. This approach presents both advantages and disadvantages, particularly in terms of significantly reducing the number of subjects while enhancing the distinction between the two classes.

As illustrated in Table 3.2, the label distribution across all recordings indicates a well-balanced dataset for STAI-Y labels. In contrast, for the SS labels, almost half of the recordings are classified as moderately stressed. Adopting a binary classification approach, in this case, may not yield satisfactory results.

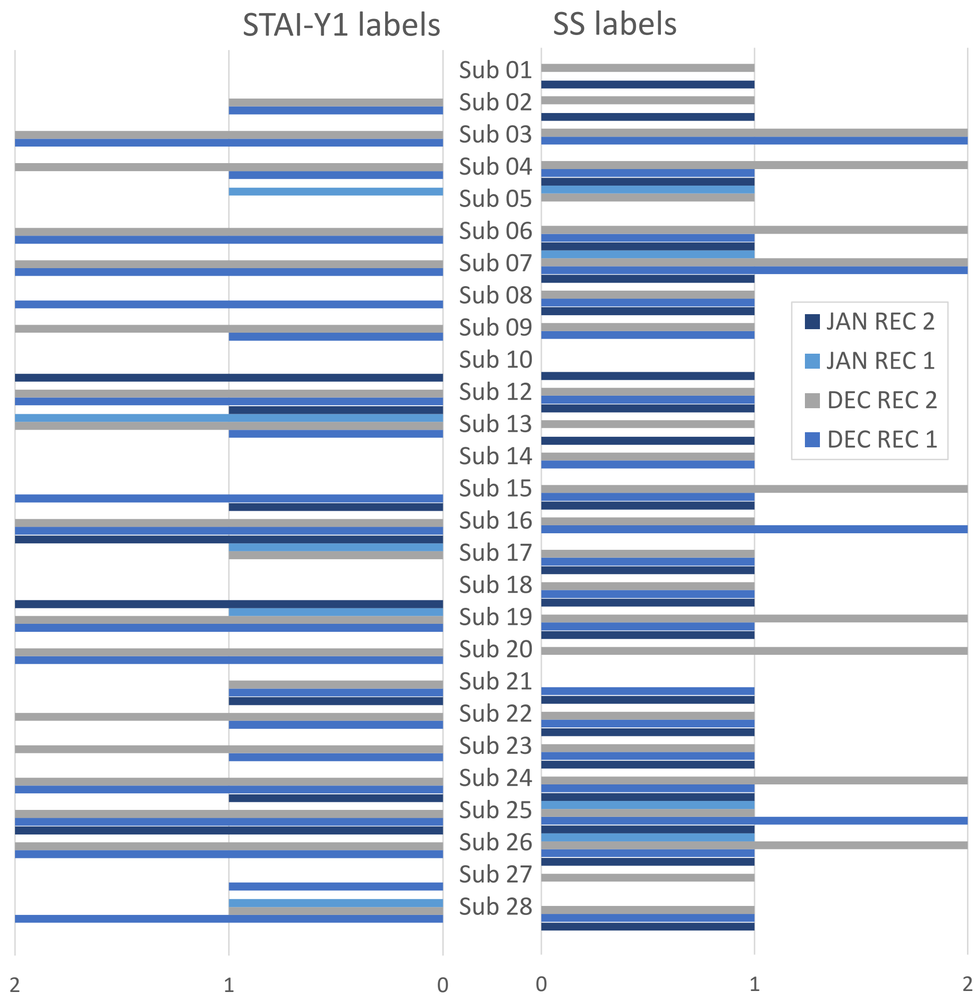


Figure 3.8: Labels for each participant, with labels equal to 0, 1, or 2 for non-stressed, moderately stressed, and stressed, respectively. STAI-Y labels are shown on the left and SS labels are shown on the right.

Table 3.2: Distribution of recordings labeled as non-stressed, moderately stressed, and stressed for both label sets. Note the large portion of moderately stressed for SS labels.

	Non-stressed	Moderately stressed	Stressed
STAI-Y	47	24	32
SS	39	51	13

3.3 Data Exploration

To gain insight into the quality and homogeneity of the collected EEG data, a data exploration was conducted. This involved evaluating the distribution of data between different classes, examining the raw data's appearance, and performing analysis using preprocessing tools. As described earlier the recordings were done over 5-minute periods and sampled at 250Hz, giving the raw signals a length of 75000 samples.

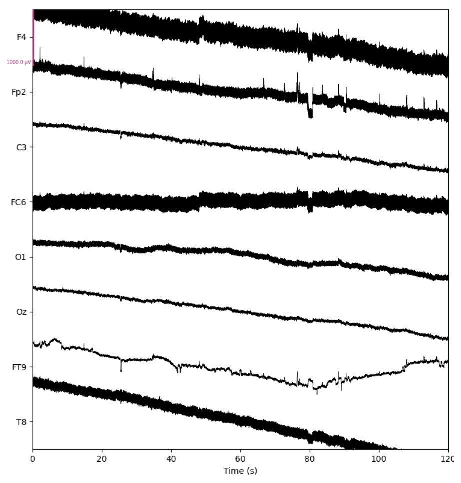
First, the distribution of labels was examined, as shown in Table 3.2. The labels derived from STAI-Y-scores revealed an imbalance in the dataset, with 45.6% non-stressed, 23.3% moderately stressed, and 31.1% stressed. However, it balances better when removing moderately stressed subjects: 59.5% non-stressed and 40.5% stressed. The dataset becomes even more imbalanced when using the SS-levels, as more participants get labeled as moderately stressed (49.5%) than stressed (12.6%) and non-stressed (37.9%). Removing moderately stressed subjects does not help balance this dataset either, as the result will be a small sample size consisting of 25% stressed and 75% non-stressed. From these observations, it was determined to change to a binary classification approach and eliminate the moderately stressed recordings. Both label sets were tested initially, and the results can be viewed in chapter 4.

The focus of our data exploration will be Participant 3's recordings. This participant reported a significant difference in stress levels between the recordings conducted in January and those conducted in December, as seen in Table 3.3. Examining the raw data from all four recordings (Figure 3.9), it becomes apparent that there is an increase in signal noise during the recordings from session 2. This is likely attributed to nearby construction work that took place during the second session.

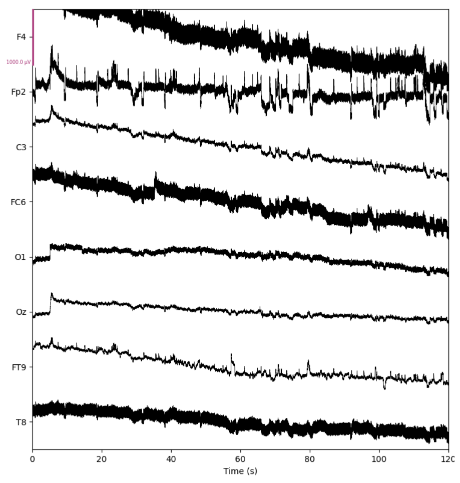
Figure 3.10 displays the Power Spectral Density (PSD) data from Participant 3, where an overall increase in signal intensity from Session 1 to Session 2 across all frequencies can be observed. However, it remains uncertain whether this increase is due to noise or the participant's reported stress levels. There are also some visible differences between the sessions, especially around the 20-40 Hz range. Given that PSD plots exhibit distinct visual disparities between stressed and non-stressed recordings, it would be worthwhile to explore the incorporation of PSD EEG in the development of the classification model.

Table 3.3: Overview of Participant 3's stress levels at all recordings given by STAI-Y-scores (min 20, max 80) and SS-levels (min 1, max 10).

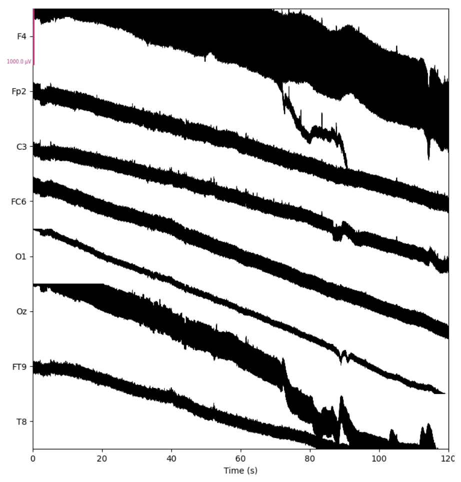
Recording	STAI-Y-score	SS-level
Session 1, Run 1	58/80	7/10
Session 1, Run 2	56/80	9/10
Session 2, Run 1	36/80	2/10
Session 2, Run 2	35/80	4/10



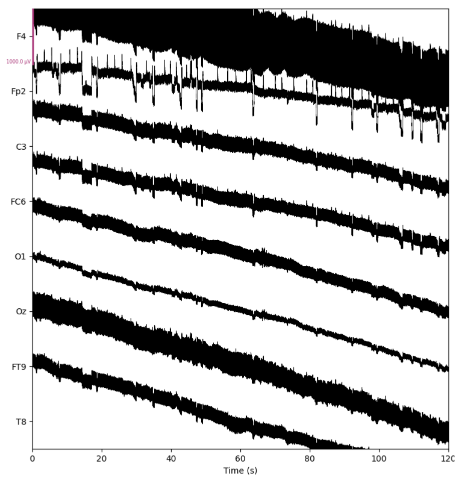
(a) RAW EEG data for Participant 3, Session 1, Run 1.



(b) RAW EEG data for Participant 3, Session 1, Run 2.

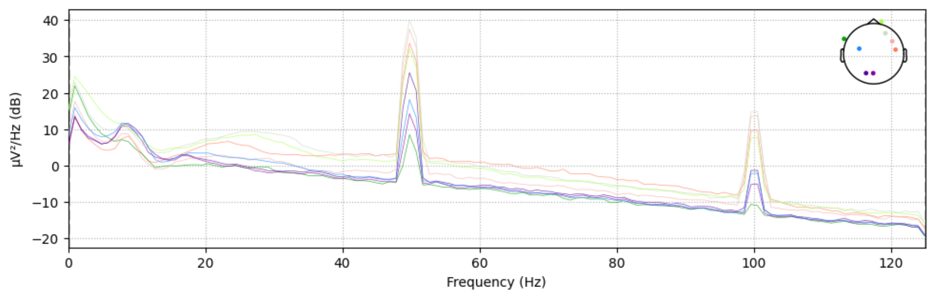


(c) RAW EEG data for Participant 3, Session 2, Run 1.

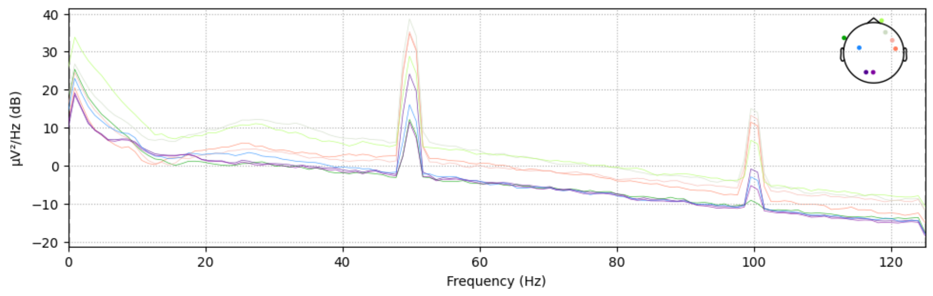


(d) RAW EEG data for Participant 3, Session 2, Run 2.

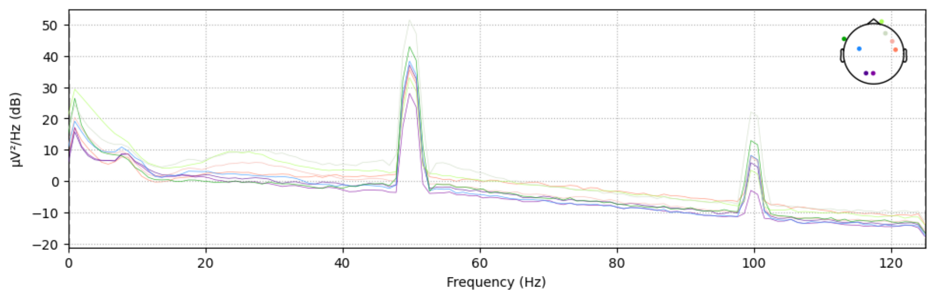
Figure 3.9: Comparison between the RAW EEG data from Participant 3. There are noticeable drifts present in all four recordings, as well as high-frequency noise, especially around the frontal electrodes. Notice the increase in signal noise during the recordings from session 2, due to nearby construction work during recording.



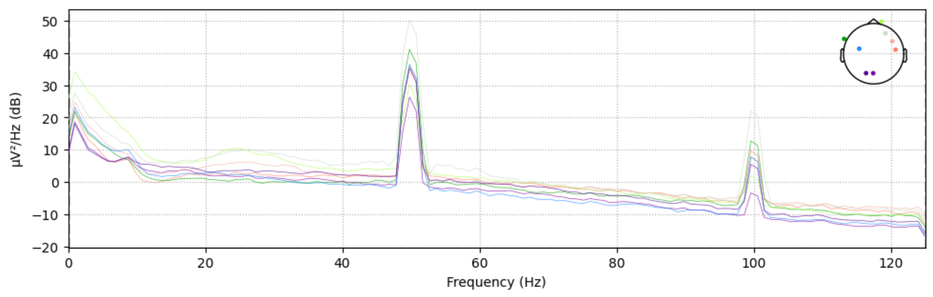
(a) PSD of RAW data for Participant 3, Session 1, Run 1. The participant's statement labels this recording as stressed.



(b) PSD of RAW data for Participant 3, Session 1, Run 2. The participant's statement labels this recording as stressed.



(c) PSD of RAW data for Participant 3, Session 2, Run 2. The participant's statement labels this recording as non-stressed.



(d) PSD of RAW data for Participant 3, Session 2, Run 2. The participant's statement labels this recording as non-stressed.

Figure 3.10: Comparison between PSD analysis of Participant 3's recordings. Notably, there is a consistent increase in signal intensity across all frequencies from Session 1 to Session 2. Additionally, visual disparities are evident between the sessions within the 20 - 40 Hz range. These variations instill optimism regarding the potential of incorporating PSD data in the classification model.

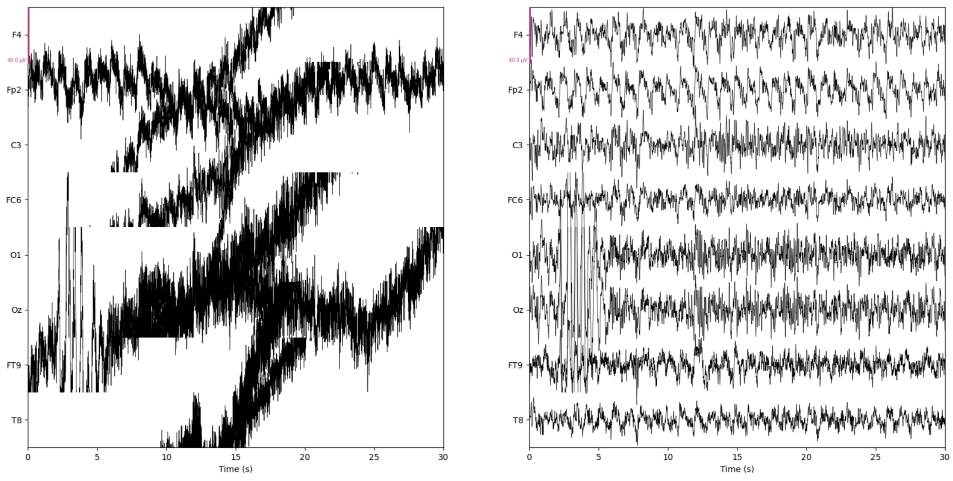
3.4 Pre-processing

In order to efficiently and reliably filter raw EEG data, a Python class named `Recording` was implemented during the project thesis. The class includes functions for loading raw data, filtering, initialization of ICA, analyzing the results, and saving the filtered data. The class has been extensively explained in the project thesis [1]. This thesis will thus not go into comprehensive details for describing the implementation of the class. Interested readers can find the full code and explanation in the project thesis [1].

A brief overview will be repeated for context's sake.

3.4.1 Filtering and Artifact Removal

This section provides an overview of the filtering process for Subject 1, Session 1, Run 1. After loading the RAW data, an initial filtering step is performed using a Savitzky-Golay filter with approximate cut-off frequency 10 Hz along with a basic band-pass filter with cut-off frequencies at 1 Hz and 50 Hz. The resulting data is referred to as "INIT" data. Figure 3.11 compares the RAW data with the INIT data, indicating successful removal of noise and drift. However, the residual peaks and jumps require an Independent Component Analysis.



(a) Raw EEG data **(b)** Filtered EEG data

Figure 3.11: Comparison between the RAW EEG data and the INIT data of Subject 1, Session 1, Run 1. The RAW data in Figure a) exhibits substantial amounts of noise, low-frequency drift, and artifacts. On the other hand, Figure b) displays the data after initial filtering, where the low-frequency drifts have been eliminated. However, some noise and artifacts remain in the data.

Working further on pre-processing the data, an ICA procedure is conducted. Figure 3.12 and Figure 3.13 shows the results after fitting an 8-component ICA model on the initially filtered data. Components 0 and 1, which are identified as artifacts due to their large spikes and/or location in Figure 3.13, are highlighted in Figure 3.12.

The effectiveness of the test removal of these components is shown in Figure 3.14: The red graph shows before cleaning and the black graph shows after. Note that some, but not all, of the large peaks have disappeared, while most of the data is still intact. Thus, the removal of components 0 and 1 yielded somewhat satisfactory results, and the components should be excluded permanently. After the exclusion, the data was reconstructed. This data will be referred to as "ICA" data. Figure 3.15 shows the difference between the EEG data before and after artifact removal. Large improvements have been done to the data: Both excessive peaks and jumps have vanished. Nonetheless, a significant amount of noise is still present in the data.

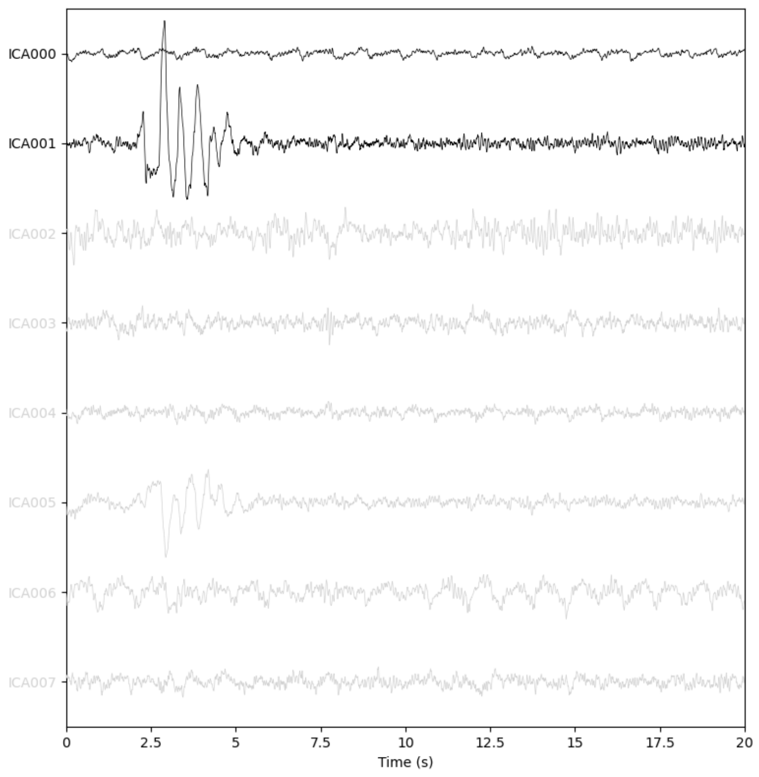


Figure 3.12: Components' voltage over time from running ICA on INIT data from Subject 1, Session 1, Run 1. Components 0 and 1 are highlighted as they stand out as artifacts. These components should be excluded.

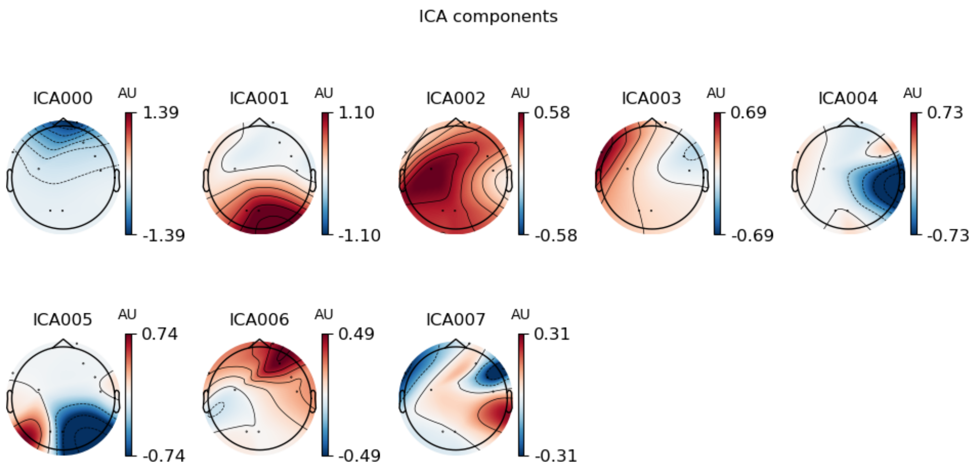


Figure 3.13: Topomap from running ICA on INIT data from Subject 1, Session 1, Run 1. The figure shows the location of the brain activity of each component. Comparing this figure with Figure 3.12, we conclude that components 0 and 1 should be removed.

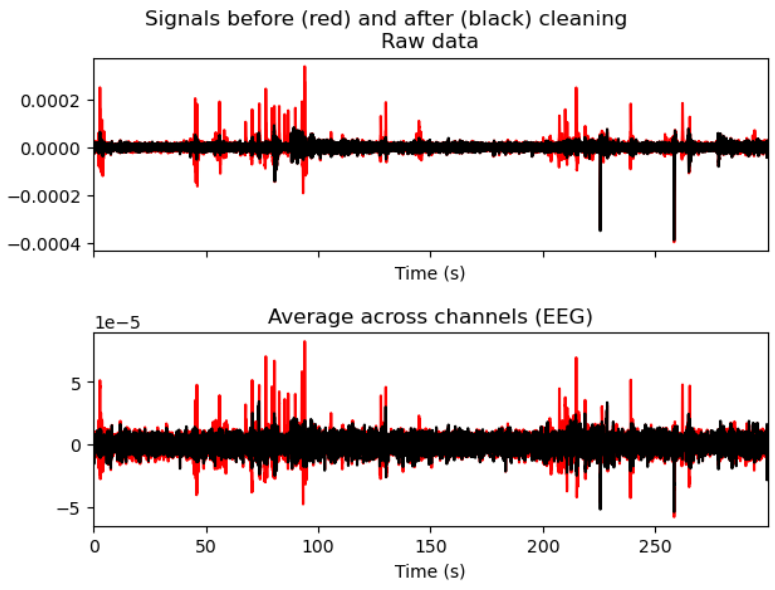
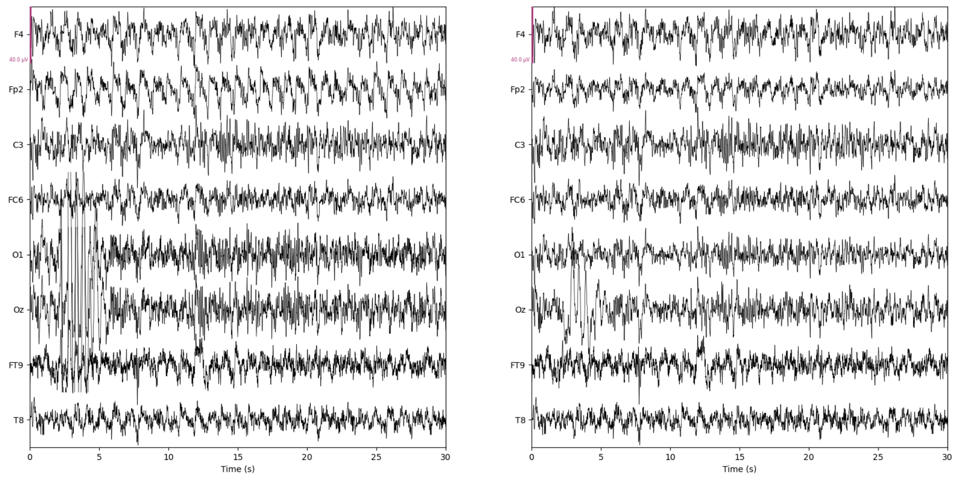


Figure 3.14: Results from removing ICA components 0 and 1 from the initially filtered data from Subject 1, Session 1, Run 1. The red graph shows before and the black graph shows after cleaning. Some peaks disappear during cleaning.



(a) Filtered EEG data **(b)** Reconstructed EEG data

Figure 3.15: INIT data vs the reconstructed data (or ICA data) after removing ICA components 0 and 1 from Subject 1, Session 1, Run 1. There is some improvement from a) to b): the large jumps and peaks have been removed, especially around channels O1 and Oz.

3.5 Machine Learning (ML)

A K-Nearest-Neighbours (KNN)-, Support Vector Machines (SVM)- and Multilayer Perceptron (MLP) was implemented by Christian Sletten in his project thesis [36], utilizing the `Python`-packages `keras`- and `tensorflow`. All classifiers were equipped with hyperparameter tuners, which help optimize the accuracy of each classifier, regardless of the dataset. The MLP classifier's significantly larger parameter grid resulted in a considerable increase in the execution time compared to the other classifiers. Furthermore, the outcomes were inconsistent and demonstrated a sub-optimal performance overall. As a result, it was determined to eliminate this classifier from any further research.
The parameter grids of the KNN and SVM-classifiers were set up as follows:

- The KNN classifier iterates through a parameter grid with leaf sizes from 1 to 10, the number of neighbors from 1 to 5, and power parameter from 1 to 2, which sets distance calculation to either Manhattan (1) or Euclidean (2).
- The SVM iterates through a parameter grid with the regularization parameter either equal to 1e-3, 1e-2, 1e-1, 1, 10, 100, 1e3, 1e4, 1e5 and 1e6 and the kernel either equal to linear, poly, rbf or sigmoid.

The code has been included in subsection A.0.3.

The subjects were divided into two sets, namely training and test sets, such that no participant was present in both sets. The training set consisted of 80% of the subjects, while the remaining 20% formed the test set. This approach allowed us to evaluate the classifier's reliability on previously unseen data from brand-new subjects. The performance of both classifiers was tested on the collected dataset using a combination of feature and label sets, with epoch length equal to 1 sec for entropy features and 0.1 sec for the rest.

3.6 Testing Robustness with SAM40

In order to test the classifier's robustness, a suggestion was to compare the performance of predicting the original test data with data from the SAM40 dataset utilized in the project thesis. The RAW data was resampled at 128 Hz and cut off at 25 seconds to match the SAM40 sampling frequency and duration. Furthermore, only the overlapping channels were separated from the SAM40 data and used. Then, a new SVM classifier was implemented to perform two predictions: one on the test set as before, and one on the SAM40 data. The code is included in subsection A.0.3.

3.7 New Methods

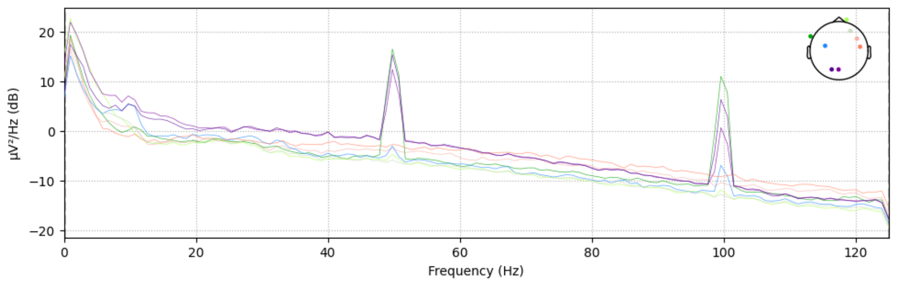
Although the previous methods developed in the project thesis performed well in some scenarios, as will be displayed and discussed in chapter 4 - chapter 5, it was preferable to explore other methods for preprocessing and classifying the data. Specifically, we aimed to investigate alternative approaches to possibly find a more robust and accurate classifier. The following section will present some of the methods that were tested.

3.7.1 New Filtering Approach

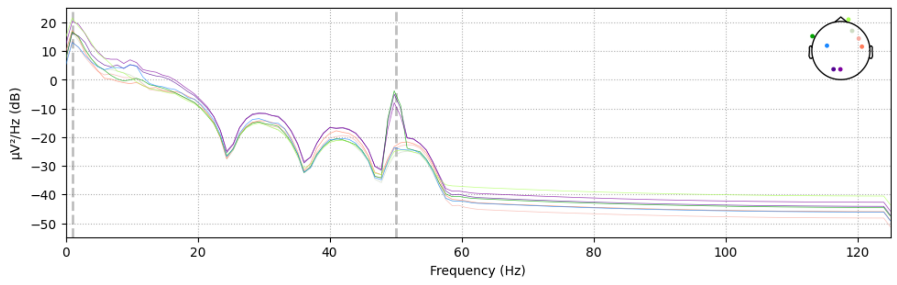
During a Power Spectral Density (PSD) analysis, it was discovered that using the exact same initial filtering technique as that employed in the project thesis was not a wise choice. The previous dataset was filtered initially with a band-pass filter having cut-off frequencies of 1 and 50 Hz, in addition to a Savitzky-Golay filter with an approximate cut-off frequency of 10 Hz. This resulted in a periodic PSD signal after filtering, as depicted in Figure 3.16b. Moreover, it was observed that the recording had picked up the utility frequency, which led to the peaks shown in Figure 3.16a.

The new RAW data underwent a more customized filtering process, which involved implementing a band-pass filter with cut-off frequencies between 1-80 Hz and a notch filter at 50 Hz and 100 Hz with a transition bandwidth of 4 Hz. This processed dataset will be referred to as the "NEW_INIT" data. As a result of this filtering, the Power Spectral Density (PSD) analysis shown in Figure 3.16c was obtained. Here, there are no periodic properties showing, and the peaks have disappeared. The notch filter was however not able to adjust perfectly and left a small notch in the data at 50 Hz.

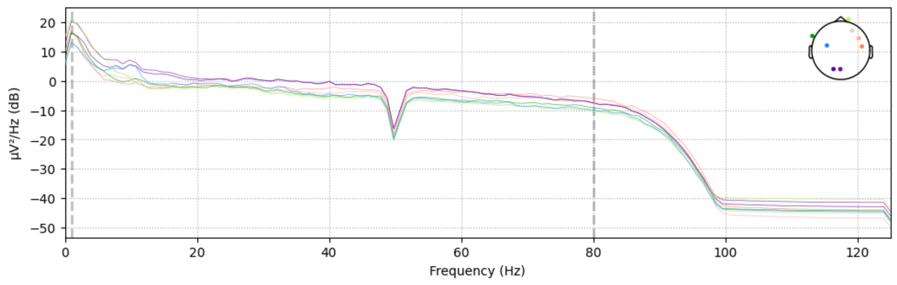
To further extend the original filtering code used in the project thesis, a PSD analysis was conducted using the `mne` package in Python. The dataset obtained from this PSD analysis of the NEW_INIT data is referred to as the "PSD"-data.



(a) PSD of RAW data



(b) PSD of INIT data



(c) PSD of NEW_INIT data

Figure 3.16: Comparison of Power Spectral Density (PSD) analysis of RAW, INIT and NEW_INIT data. For the RAW data (a), we see clear peaks at the utility frequencies (50 Hz and 100 Hz). For the previous filtering approach (b), the utility frequency gets partly cut off with a band-pass filter with cut-off frequencies at 1 and 50 Hz. However, a periodic property appears in the data due to inadequate tuning of filter properties. In the new approach (c), the resulting signal is smooth except at 50 Hz. This is due to the notch filter that has been applied.

3.7.2 EEGNet

In addition to the traditional classifiers, EEGNet described by Lawhern et al. [23] was also utilized. EEGNet is a compact CNN for classification and interpretation of EEG-based Brain Computer Interfaces (BCIs). BCIs refer to systems that allow direct communication between the brain and an external device, enabling control and manipulation of technology using brain signals. EEGNet is thus a type of deep learning architecture *specifically* designed for processing Electroencephalography (EEG) signals. EEGNet was developed to overcome some of the challenges associated with traditional EEG signal processing methods, such as the need for expert feature engineering and the limited ability to extract meaningful information from noisy or low-quality data [23]. The network encapsulates several well-known EEG feature extraction concepts, such as optimal spatial filtering and filter-bank construction, while simultaneously reducing the number of trainable parameters to fit compared to existing approaches. The full code implementation by the Army Research Laboratory (ARL) can be found on their GitHub. Along with the standard EEGNet implementation, ARL also included models for a shallow CNN and a deep CNN [37], which were also tested.

The testing got quite extensive, with different hyperparameters and labels tested. However, as each run took quite a lot of computer power and time, the search for an optimized version of this implementation was abandoned after some time. The finalized models included a class weight of 1-3 for non-stressed vs. stressed, respectively, epoch length equal to 1 s, and a sigmoid activation function as the last step. Each EEGNet model was tested with 10-fold cross-validation on the RAW data.

3.7.3 Temporal-constrained Sparse Group Lasso-EEGNet

Another CNN for classification that was utilized in this thesis is the Temporal-constrained Sparse Group Lasso-EEGNet (TSGL-EEGNet) that is based on the EEGNet. The motivation behind creating the TSGL-EEGNet method, as described by Deng et al. [38], was to better the interpretability of neural network methods within medical imaging BCIs. The main difference from EEGNet is the **Feature Selection** section born from the Filter Bank Common Spatial Pattern (FBCSP) and Temporal constrained Sparse Group Lasso (TSGL) regularization. Using FBCSP and TSGL the researchers sought to minimize overfitting and increase the interpretability of the feature selection of the EEGNet. This method uses FBCSP to autonomously select key temporal-spatial discriminative EEG characteristics from the EEG data before the feature selection [39]. The Sparse Group Lasso (SGL) can inhibit some of the active parameters in the neural network of the TSGL-EEGNet structure. While the role of the Temporal Lasso regularization is to keep time domain features smooth. As the name states, TSGL-EEGNet makes use of both regularization methods during feature selection. The implementation of TSGL-EEGNet that was used in this paper can be found on JohnBoxAnn's GitHub.

3.7.4 Wavelet Scattering

Wavelet scattering is a knowledge-based feature extraction method that is structurally similar to a CNN. However, it is based on the wavelet transform. The wavelet transform allows for the analysis of signals in both the time and frequency domains. Wavelet scattering is often preferred due to its stability to deformations of the data, unlike the wavelet transform alone. The method yields a deep representation of the wavelet transform that is both translation and rotation invariant and stable against deformations of the data. It also resembles a Deep Convolutional Neural Network, as it involves cascades of convolutions, activation functions (e.g. RELU), and pooling. However, wavelet scattering replaces these steps with wavelet convolution, complex modulus, and local averaging [40].

The main difference between wavelet scattering and CNNs is that wavelet scattering uses predefined filters, while CNNs require filter training. Consequently, wavelet scattering can be both efficient and accurate in limited dataset classification scenarios, whereas CNNs may struggle when the amount of training data is insufficient. The wavelet scattering transform used in this paper utilizes the Kymatio implementation [41], which provides the Scattering1D function for 1D signals. The Kymatio website offers both implementation details and examples.

For this study, the Kymatio implementation of Scattering1D was used. This function takes in the hyperparameters **J**, **Q**, and **T**, where:

- **Q** is the number of wavelets per octave,
- **J** is a scale parameter that controls the number of octaves, and
- **T** is the length of the full signal: 75000 samples in our case.

Various values for **J** and **Q** were tested and it was determined that the values $\mathbf{J = 6}$ and $\mathbf{Q = 16}$ yielded satisfactory results. It should be noted that the decision to use the wavelet scattering transform instead of other data types was based on its equivalence to a CNN.

When applying the Scattering1D function to extract features from the RAW data and using the SVM classifier, the runtime was approximately one hour. Using the KNN classifier, the runtime was reduced to around 2 minutes.

Chapter 4

Results

4.1 Best Performing Channel Selections

In the project [1], it was determined that KNN on two-times ICA-filtered EEG data with Hjorth-features performed the best, with an accuracy of 96.50%. This is thus the case that was studied further during channel selection using The Genetic Algorithm. The code is attached in subsection A.0.2 and subsection A.0.1.

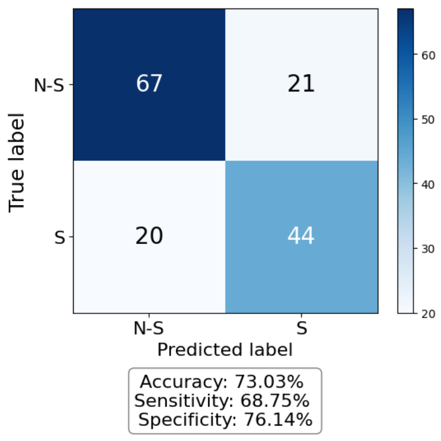
The best channel selection yielded from running the genetic algorithm in subsection A.0.2 several times yielded the results given in Table 4.1

Table 4.1: Best possible channel selection, according to accuracy and the limitations of the Mentalab EEG cap, achieved by running the Genetic Algorithm on ICA-filtered EEG data from the SAM40 dataset. Repurposed from [1].

Channels selected	Confusion Matrix	Accuracy	Specificity	Sensitivity
Fp1, C3, F4, T8, FT9, F3, Fp2, O1	$\begin{bmatrix} \mathbf{262} & \mathbf{30} \\ \mathbf{27} & \mathbf{281} \end{bmatrix}$	**90.50%**	**91.23%**	**89.73%**

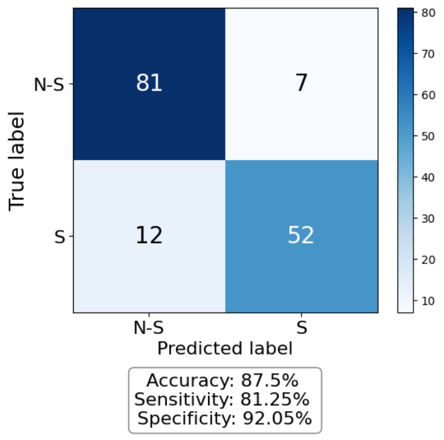
4.2 Performance of RAW Data

4.2.1 STAI-Y-labels



(a) Performance of KNN with:

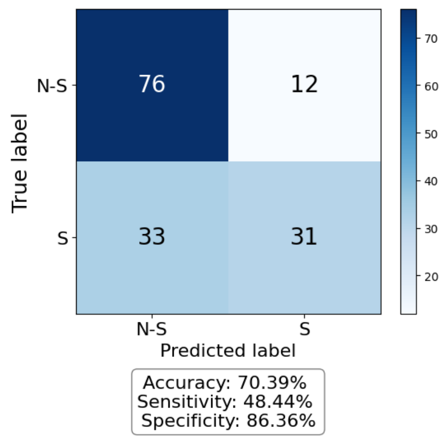
- `leaf_size = 1`
- `n_neighbors = 3`
- `p = 2`



(b) Performance of SVM with:

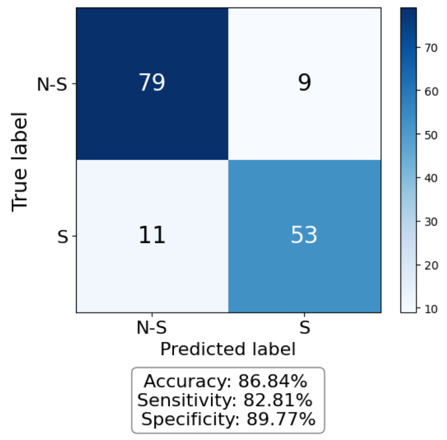
- `C = 100`
- `kernel = rbf`

Figure 4.1: The performance for the best fold in 10-fold cross-validation of classifying the **full** RAW data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 67.34% for KNN and 65.33% for SVM.



(a) Performance of KNN with:

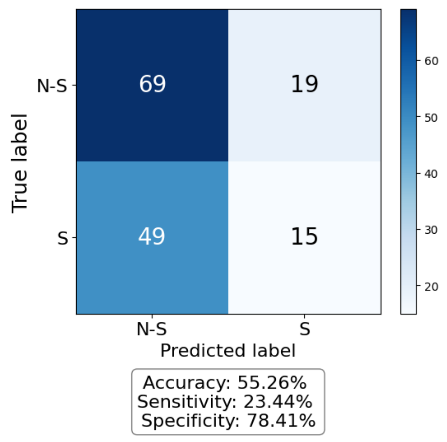
- `leaf_size = 1`
- `n_neighbors = 3`
- `p = 2`



(b) Performance of SVM with:

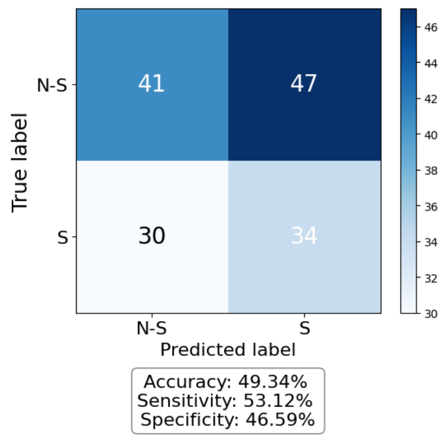
- `C = 0.01`
- `kernel = linear`

Figure 4.2: The performance for the best fold in 10-fold cross-validation of classifying the **time series features** of RAW data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 68.61% for KNN and 63.20% for SVM.



(a) Performance of KNN with:

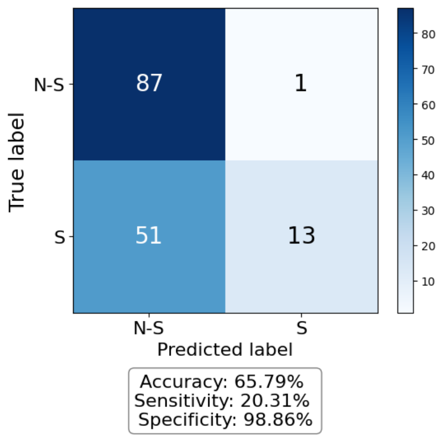
- `leaf_size = 1`
- `n_neighbors = 2`
- `p = 2`



(b) Performance of SVM with:

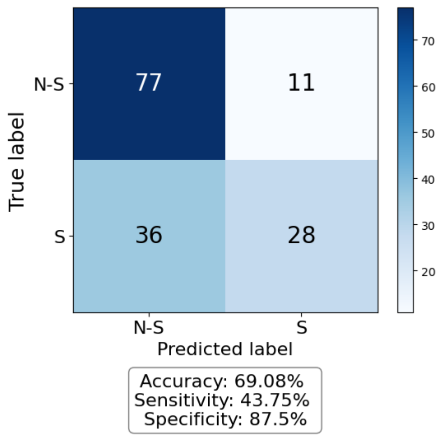
- `C = 0.01`
- `kernel = linear`

Figure 4.3: The performance for the best fold in 10-fold cross-validation of classifying the **fractal features** of RAW data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 56.51% for KNN and 58.68% for SVM.



(a) Performance of KNN with:

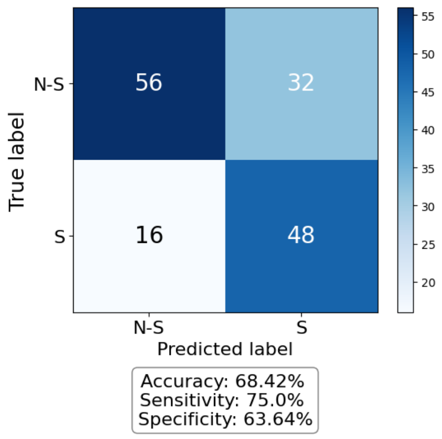
- `leaf_size = 1`
- `n_neighbors = 4`
- `p = 1`



(b) Performance of SVM with:

- `C = 100`
- `kernel = rbf`

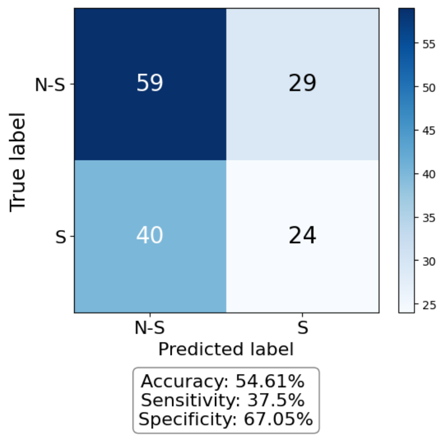
Figure 4.4: The performance for the best fold in 10-fold cross-validation of classifying the **entropy features** of RAW data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 66.69% for KNN and 62.52% for SVM.



(a) Performance of KNN with:

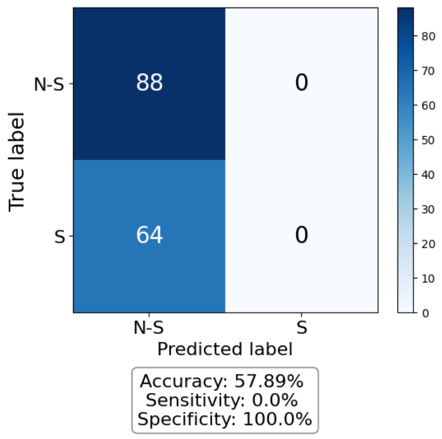
- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 2`

Figure 4.5: The performance for the best fold in 10-fold cross-validation of classifying the **Hjorth features** of RAW data with KNN and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 69.30% for KNN. Runtime for Hjorth features of RAW data was >4 hours, thus it is not included in the results.



(a) Performance of KNN with:

- `leaf_size = 1`
- `n_neighbors = 2`
- `p = 2`

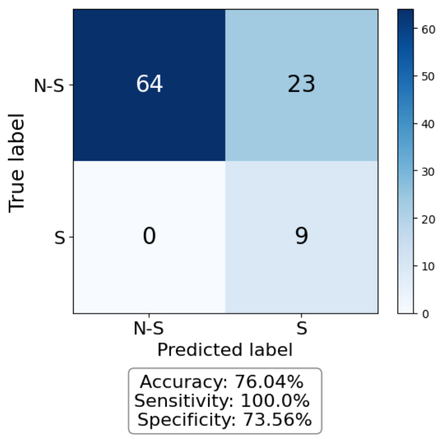


(b) Performance of SVM with:

- `C = 0.001`
- `kernel = poly`

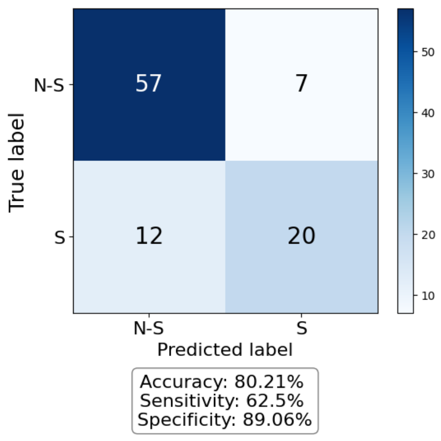
Figure 4.6: The performance for the best fold in 10-fold cross-validation of classifying the **frequency band features** of RAW data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 45.76% for KNN and 55.91% for SVM

4.2.2 SS-labels



(a) Performance of KNN with:

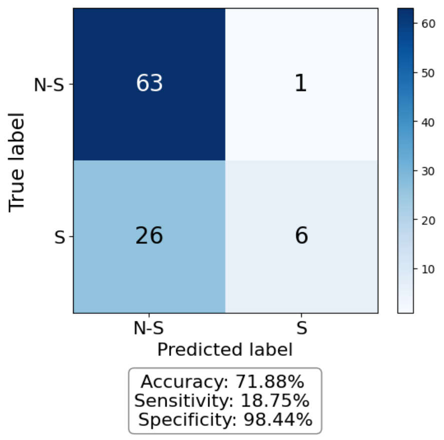
- `leaf_size = 1`
- `n_neighbors = 3`
- `p = 2`



(b) Performance of SVM with:

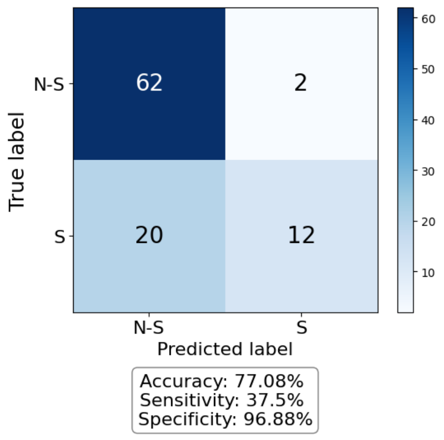
- `C = 100`
- `kernel = rbf`

Figure 4.7: The performance for the best fold in 10-fold cross-validation of classifying the **full** RAW data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 78.53% for KNN and 75.94% for SVM.



(a) Performance of KNN with:

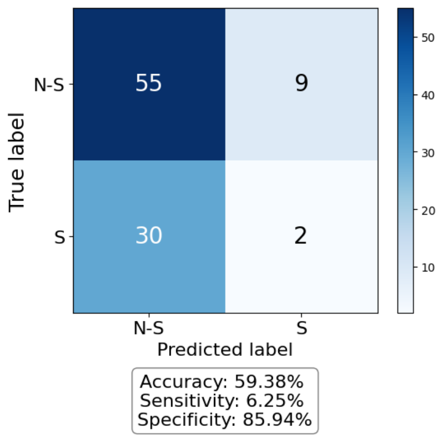
- `leaf_size = 1`
- `n_neighbors = 2`
- `p = 2`



(b) Performance of SVM with:

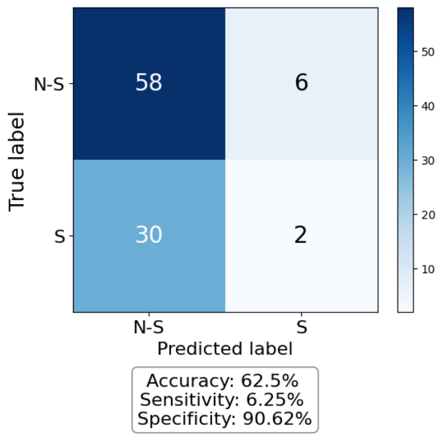
- `C = 1`
- `kernel = linear`

Figure 4.8: The performance for the best fold in 10-fold cross-validation of classifying the **time series features** of RAW data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 73.44% for KNN and 75.80% for SVM.



(a) Performance of KNN with:

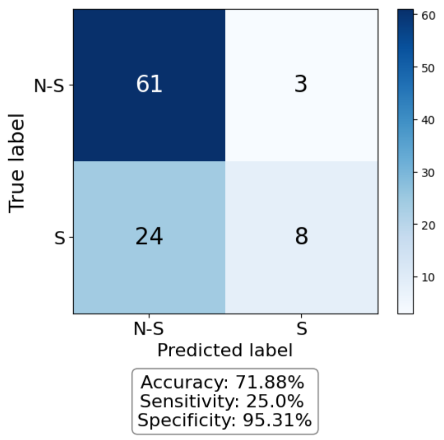
- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 2`



(b) Performance of SVM with:

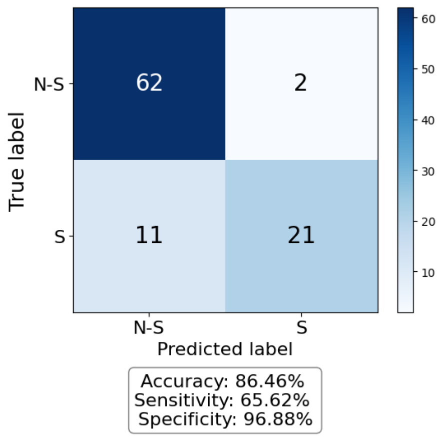
- `C = 0.01`
- `kernel = linear`

Figure 4.9: The performance for the best fold in 10-fold cross-validation of classifying the **fractal features** of RAW data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 78.91% for KNN and 80.00% for SVM.



(a) Performance of KNN with:

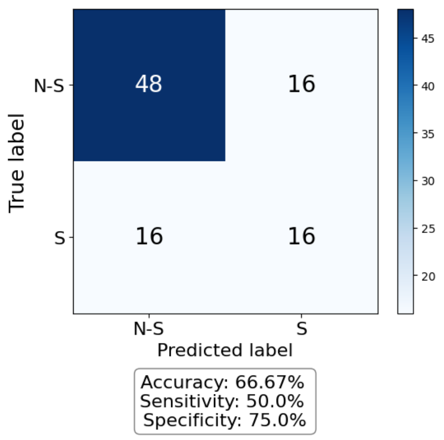
- `leaf_size = 1`
- `n_neighbors = 4`
- `p = 1`



(b) Performance of SVM with:

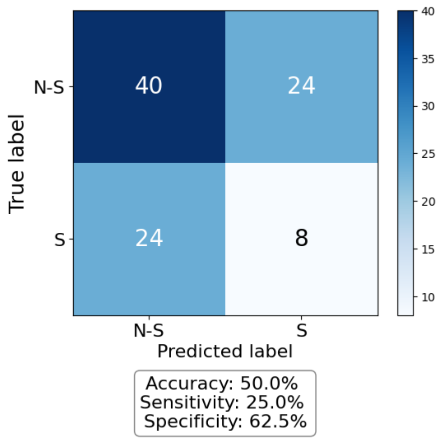
- `C = 1000`
- `kernel = rbf`

Figure 4.10: The performance for the best fold in 10-fold cross-validation of classifying the **entropy features** of RAW data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 76.95% for KNN and 78.61% for SVM.



(a) Performance of KNN with:

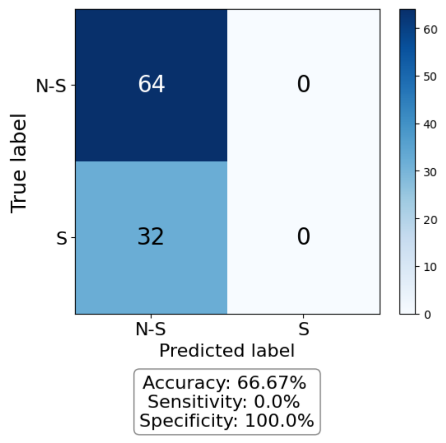
- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 2`



(b) Performance of SVM with:

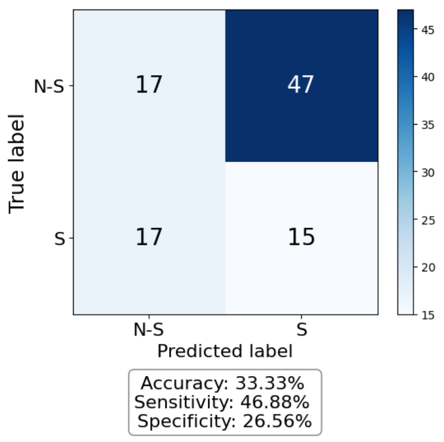
- `C = 100000`
- `kernel = rbf`

Figure 4.11: The performance for the best fold in 10-fold cross-validation of classifying the **Hjorth features** of RAW data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 81.02% for KNN and 73.27% for SVM.



(a) Performance of KNN with:

- `leaf_size = 1`
- `n_neighbors = 3`
- `p = 2`



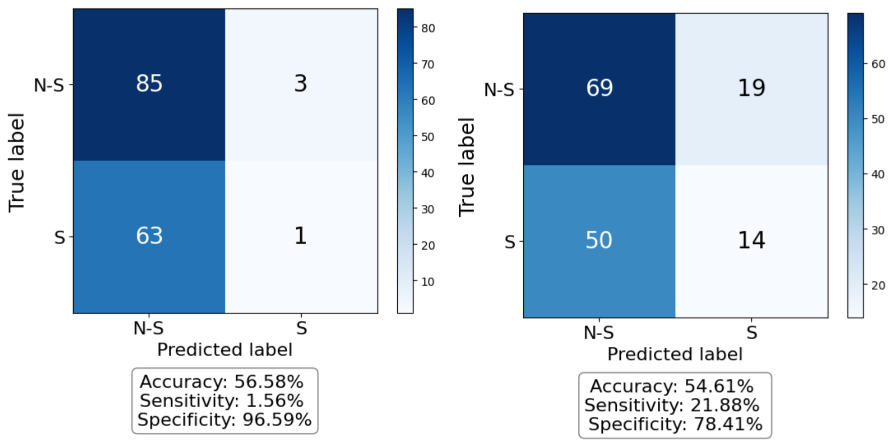
(b) Performance of SVM with:

- `C = 1000`
- `kernel = rbf`

Figure 4.12: The performance for the best fold in 10-fold cross-validation of classifying the **frequency band features** of RAW data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 84.26% for KNN and 77.38% for SVM.

4.3 Performance of INIT data

4.3.1 STAI-Y-labels



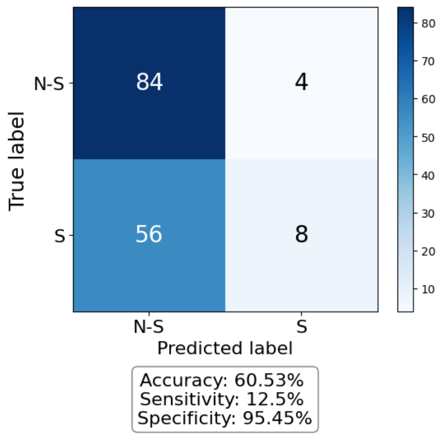
(a) Performance of KNN with:

- `leaf_size = 1`
- `n_neighbors = 2`
- `p = 1`

(b) Performance of SVM with:

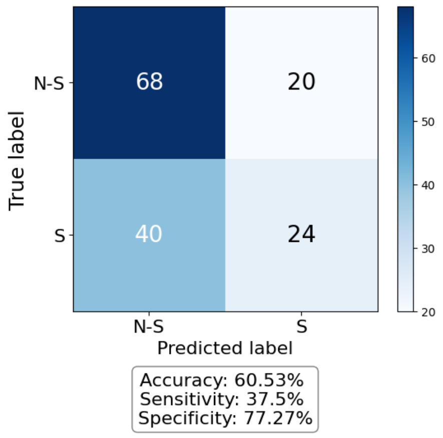
- `C = 0.001`
- `kernel = linear`

Figure 4.13: The performance for the best fold in 10-fold cross-validation of classifying the **full** INIT data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 64.69% for KNN and 65.01% for SVM.



(a) Performance of KNN with:

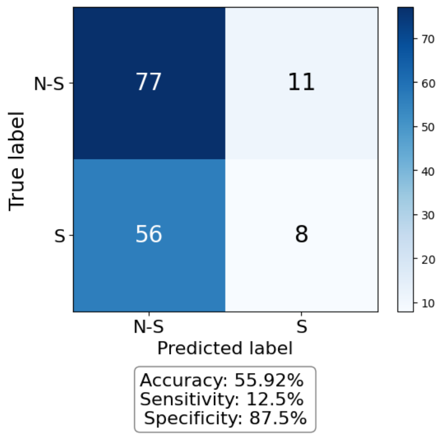
- `leaf_size = 1`
- `n_neighbors = 4`
- `p = 1`



(b) Performance of SVM with:

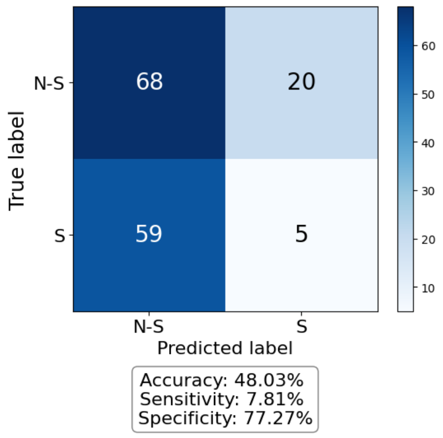
- `C = 10000`
- `kernel = rbf`

Figure 4.14: The performance for the best fold in 10-fold cross-validation of classifying the **time series features** of INIT data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 61.30% for KNN and 58.69% for SVM.



(a) Performance of KNN with:

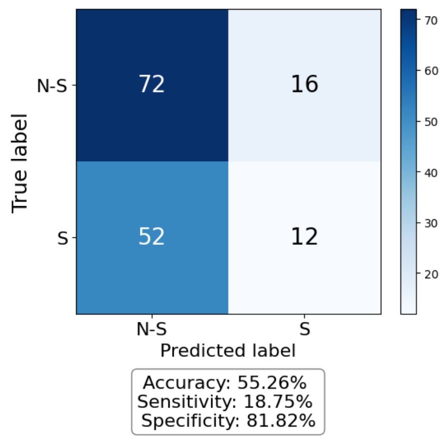
- `leaf_size = 1`
- `n_neighbors = 2`
- `p = 1`



(b) Performance of SVM with:

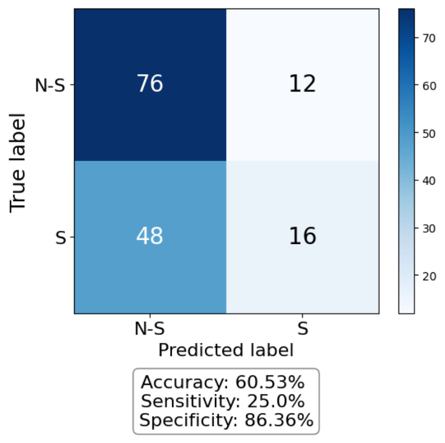
- `C = 0.01`
- `kernel = linear`

Figure 4.15: The performance for the best fold in 10-fold cross-validation of classifying the **fractal features** of INIT data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 63.18% for KNN and 61.91% for SVM.



(a) Performance of KNN with:

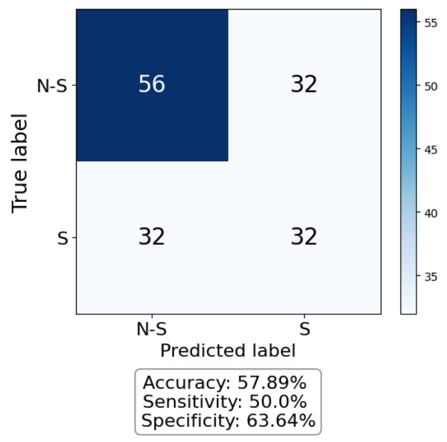
- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 1`



(b) Performance of SVM with:

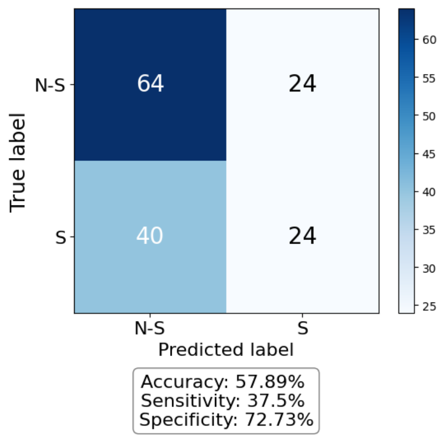
- `C = 100`
- `kernel = poly`

Figure 4.16: The performance for the best fold in 10-fold cross-validation of classifying the **entropy features** of INIT data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 58.57% for KNN and 58.23% for SVM.



(a) Performance of KNN with:

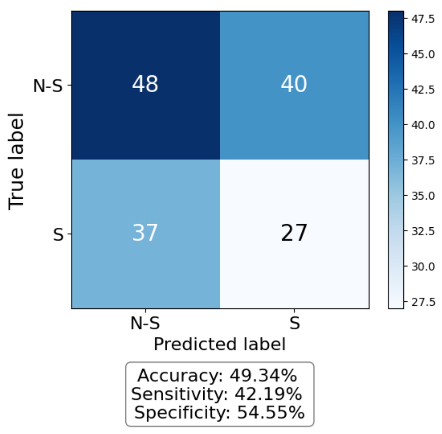
- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 1`



(b) Performance of SVM with:

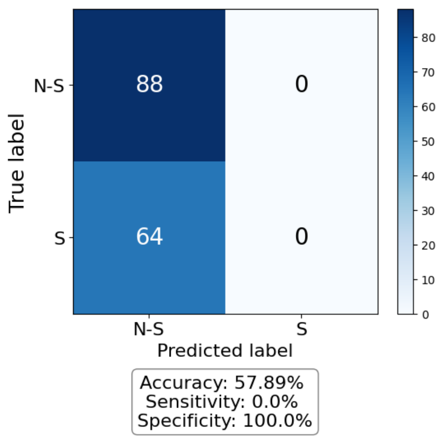
- `C = 100000`
- `kernel = linear`

Figure 4.17: The performance for the best fold in 10-fold cross-validation of classifying the **Hjorth features** of INIT data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 67.01% for KNN and 60.34% for SVM.



(a) Performance of KNN with:

- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 1`



(b) Performance of SVM with:

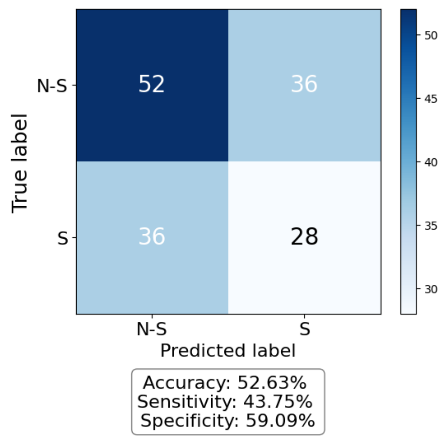
- `C = 10`
- `kernel = poly`

Figure 4.18: The performance for the best fold in 10-fold cross-validation of classifying the **frequency band features** of INIT data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 51.56% for KNN and 55.97% for SVM.

4.4 Performance of ICA data

4.4.1 STAI-Y-labels

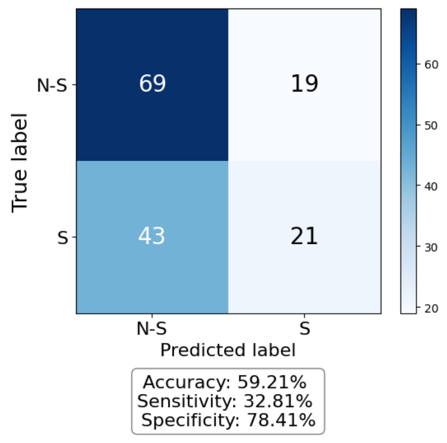
Note: Runtime for full ICA data was >4 hours, thus it is not included in the results.



Accuracy: 52.63%
Sensitivity: 43.75%
Specificity: 59.09%

(a) Performance of KNN with:

- `leaf_size = 1`
- `n_neighbors = 2`
- `p = 1`

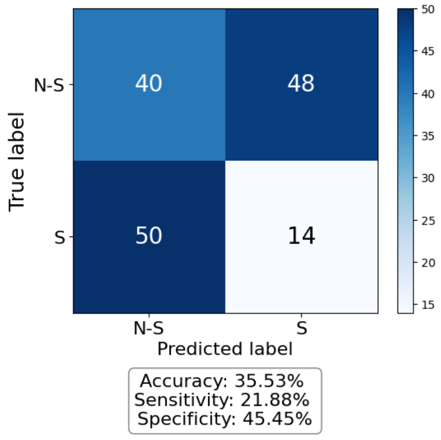


Accuracy: 59.21%
Sensitivity: 32.81%
Specificity: 78.41%

(b) Performance of SVM with:

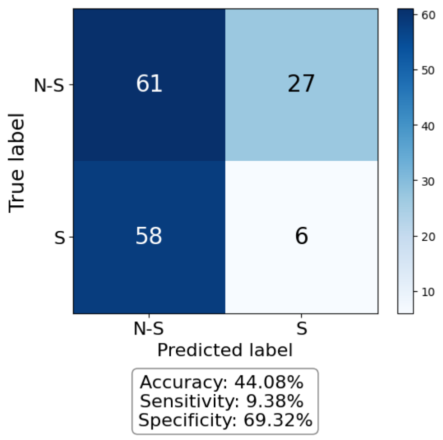
- `C = 100`
- `kernel = rbf`

Figure 4.19: The performance for the best fold in 10-fold cross-validation of classifying the **time series features** of ICA data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 52.27% for KNN and 59.11% for SVM.



(a) Performance of KNN with:

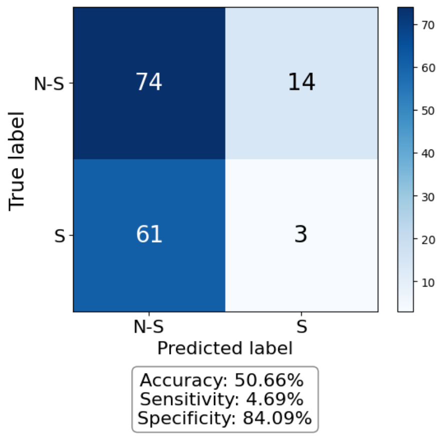
- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 1`



(b) Performance of SVM with:

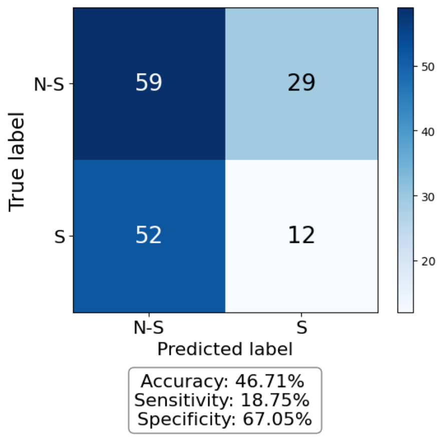
- `C = 0.01`
- `kernel = linear`

Figure 4.20: The performance for the best fold in 10-fold cross-validation of classifying the **fractal features** of ICA data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 64.92% for KNN and 63.48% for SVM.



(a) Performance of KNN with:

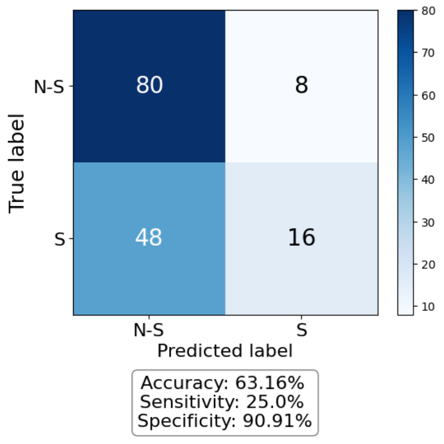
- `leaf_size = 1`
- `n_neighbors = 4`
- `p = 2`



(b) Performance of SVM with:

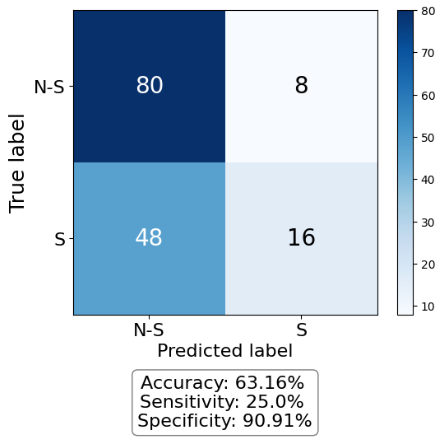
- `C = 1`
- `kernel = rbf`

Figure 4.21: The performance for the best fold in 10-fold cross-validation of classifying the **entropy features** of ICA data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 63.28% for KNN and 61.68% for SVM.



(a) Performance of KNN with:

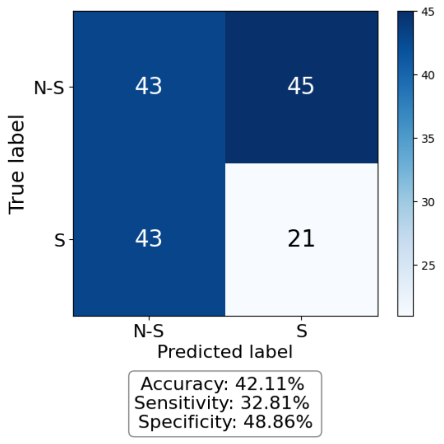
- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 2`



(b) Performance of SVM with:

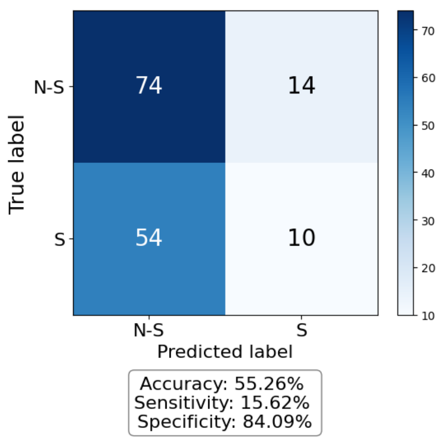
- `C = 10`
- `kernel = rbf`

Figure 4.22: The performance for the best fold in 10-fold cross-validation of classifying the **Hjorth features** of ICA data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 66.20% for KNN and 60.88% for SVM.



(a) Performance of KNN with:

- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 2`

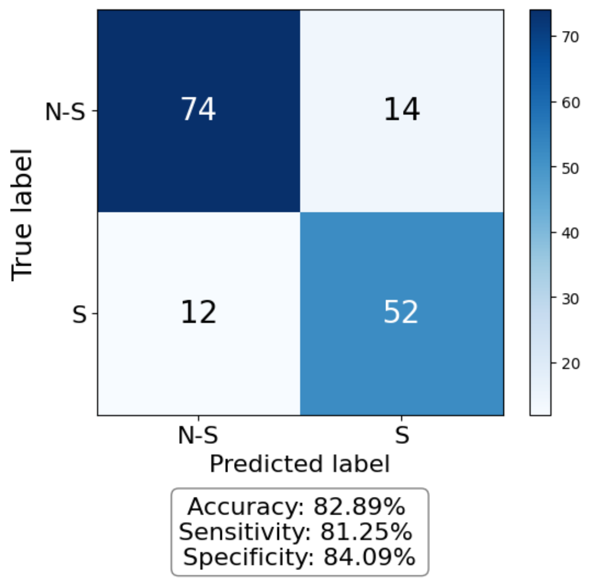


(b) Performance of SVM with:

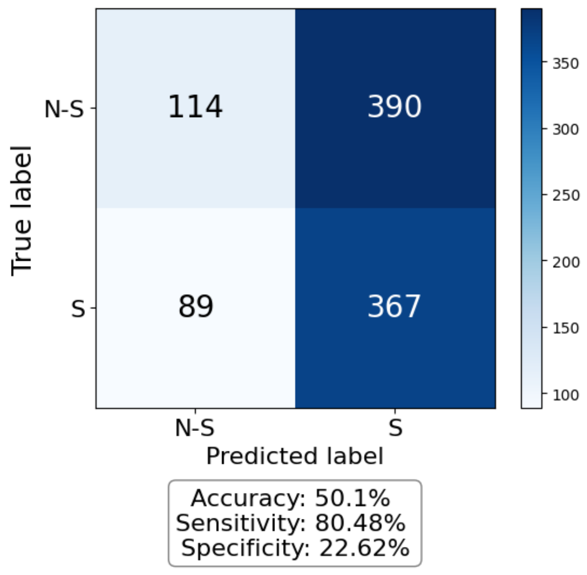
- `C = 100`
- `kernel = sigmoid`

Figure 4.23: The performance for the best fold in 10-fold cross-validation of classifying the **frequency band features** of ICA data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 55.78% for KNN and 56.54% for SVM.

4.5 Performance of SAM40

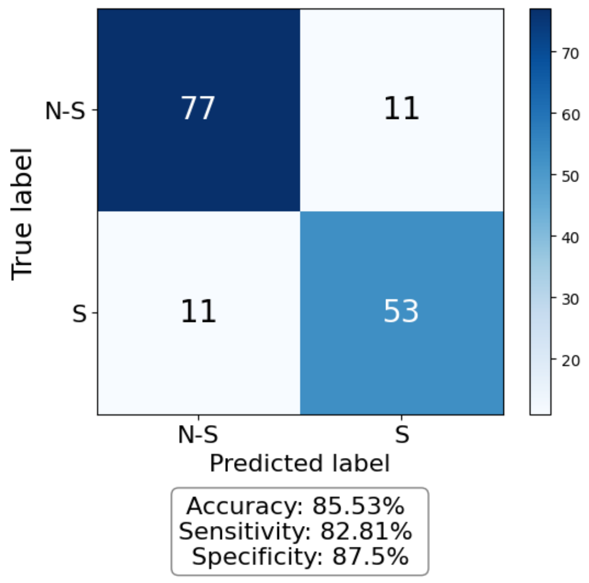


(a) Performance of SVM on downsampled RAW data

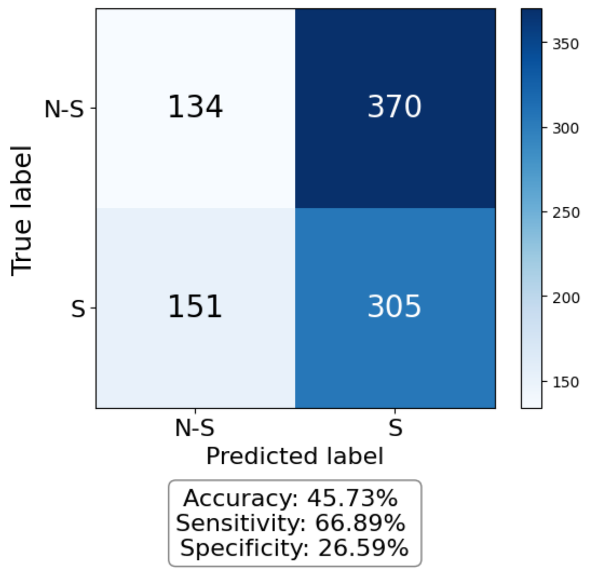


(b) Performance of SVM on SAM40 data

Figure 4.24: The performance for the best fold in 10-fold cross-validation of classifying the **full** RAW- and SAM40 data with SVM. Note the drastic decrease in accuracy.



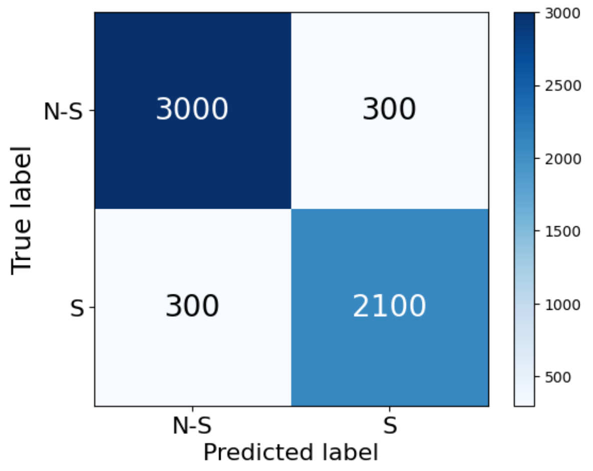
(a) Performance of SVM on downsampled RAW data



(b) Performance of SVM on SAM40 data

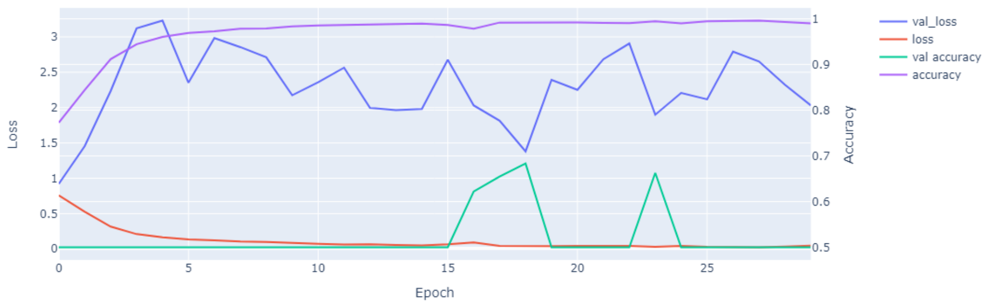
Figure 4.25: The performance for the best fold in 10-fold cross-validation of classifying the **time series features** of RAW- and SAM40 data with SVM. Note the drastic decrease in accuracy.

4.6 Performance of RAW data with EEGNet models



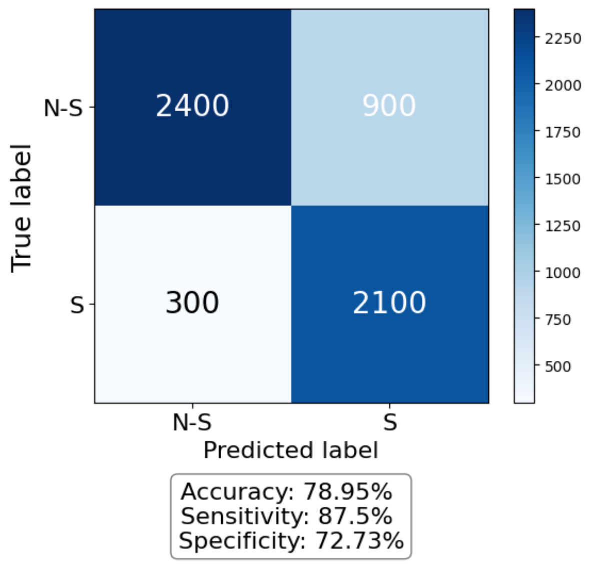
Accuracy: 89.47%
Sensitivity: 87.5%
Specificity: 90.91%

(a) Best performance of EEGNet presented as the confusion matrix, accuracy, sensitivity, and specificity.

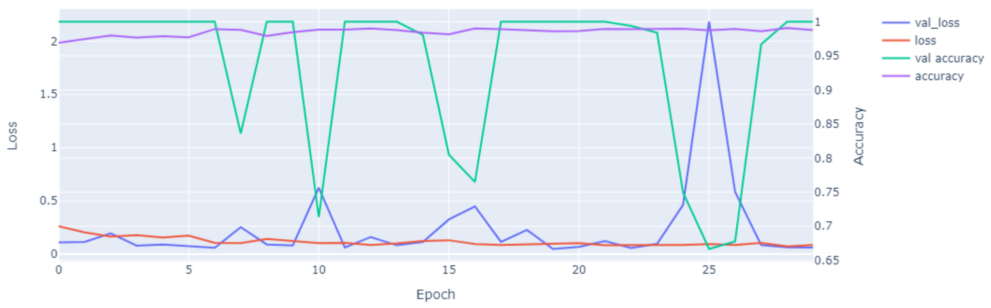


(b) Accuracy/Loss curves for training and validation of EEGNet.

Figure 4.26: Performance of the best fold in 10-fold cross-validation of EEGNet. The mean accuracy for all 10 folds was calculated to be 73.68%.

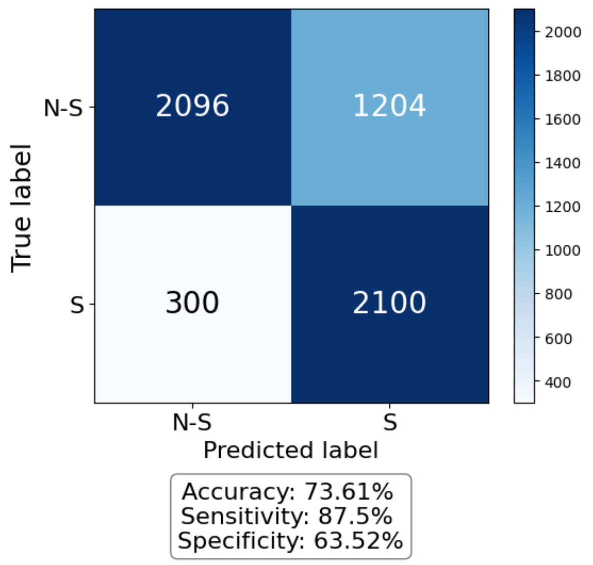


(a) Best performance of TSGL presented as the confusion matrix, accuracy, sensitivity, and specificity.

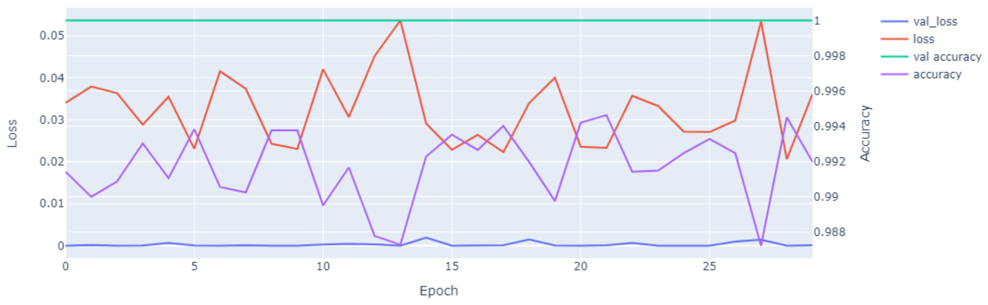


(b) Accuracy/Loss curves for training and validation of TSGL.

Figure 4.27: Performance of the best fold in 10-fold cross-validation of TSGL.The mean accuracy for all 10 folds was calculated to be 69.39%.

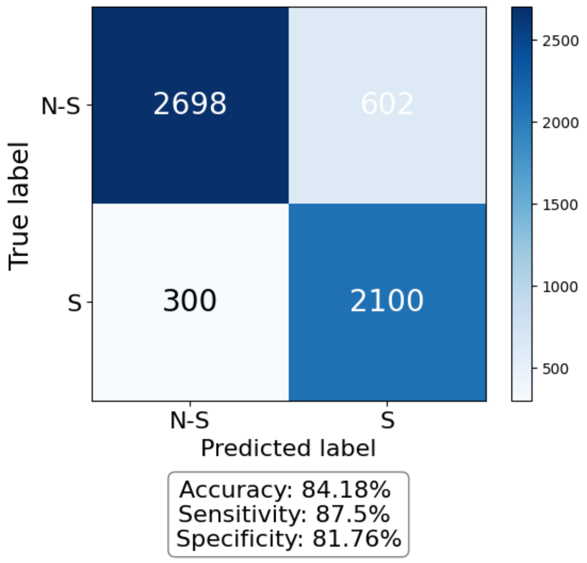


(a) Best performance of DeepConvNet presented as the confusion matrix, accuracy, sensitivity, and specificity.

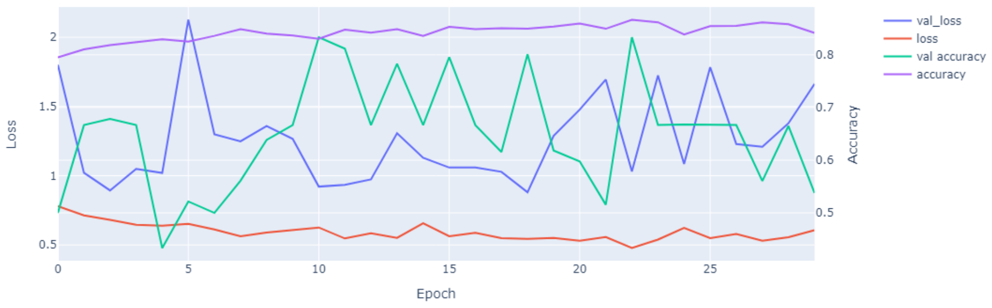


(b) Accuracy/Loss curves for training and validation of DeepConvNet.

Figure 4.28: Performance of the best fold in 10-fold cross-validation of DeepConvNet. The mean accuracy for all 10 folds was calculated to be 69.35%.



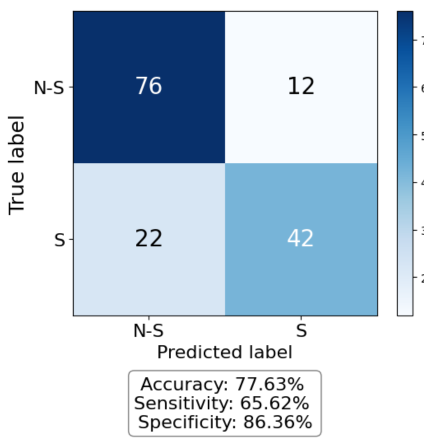
(a) Best performance of ShallowConvNet presented as the confusion matrix, accuracy, sensitivity, and specificity.



(b) Accuracy/Loss curves for training and validation of ShallowConvNet.

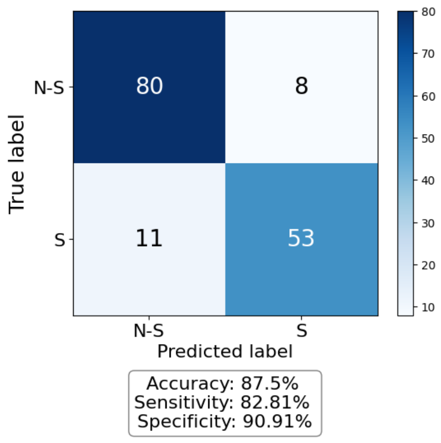
Figure 4.29: Performance of the best fold in 10-fold cross-validation of ShallowConvNet. The mean accuracy for all 10 folds was calculated to be 83.66%. Thus the ShallowConvNet outperforms the other versions of EEGNet.

4.7 Performance of Wavelet Scattering



(a) Performance of KNN with:

- `T = 75000`
- `Q = 16`
- `J = 6`



(b) Performance of SVM with:

- `T = 75000`
- `Q = 16`
- `J = 6`

Figure 4.30: The performance for the best fold in 10-fold cross-validation of classifying the **wavelet scattering features** of RAW data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 66.85% for KNN and 66.01% for SVM

Chapter 5

Discussion

5.1 EEG Channels Selected

After conducting multiple runs of the Genetic Algorithm, it was determined that the most suitable electrodes for stress detection in the project thesis were Fp1, Fp2, F3, F4, FT9, T8, C3, and O1. Consequently, these electrodes were included in the Stress Detection Studies table as our contribution to the field. The revised version of the table, Table 5.1, is presented below.

5.2 Data Collection

The sensitivity of EEG, given its small-scale nature, necessitates establishing a reliable connection between the scalp and electrodes to obtain high-quality data. Since individuals possess varying hair types and thicknesses, the level of noise and signal characteristics can vary among subjects. Some subjects were more challenging in terms of electrode fitting and calibration and had trouble maintaining electrode resistance below the ideal threshold of 160 kΩ. In contrast, other subjects were more easily set up, effortlessly achieving electrode resistances below 20 kΩ. Consequently, it remains uncertain how these factors have influenced the machine-learning models and overall results. This situation raises the crucial question of whether the models trained on the RAW dataset genuinely utilize EEG data or if the prevalent noise overwhelms the signals when building the models.

Table 5.1: EEG channels used in Stress Detection Studies and their methods for selecting them, now including our results.

EEG channels selected	**Method used**	**Accuracy**	**Paper(s)**
AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4	No method described, other than following the 10-20 system	96.00%	[29]
F3, F4, T7, C3, C2, C4, T8, P3, P4	No method described, other than following the 10-20 system	90.00%	[30]
Fp1, F3, F7, Fz, Fp2, F4, F8	No method described, other than following the 10-20 system	95.00%	[31]
TP9, FP1, FP2, TP10	No method described	85.6%	[32]
F4, FP2, F8, Fz, F3, F7	Selected based on statistical analysis of EEG electrodes	91.7%	[33]
Fp1, Fp2	No method described, other than following the 10-20 system	86.66%	[34]
Fp1, Fp2, F3, F4, FT9, T8, C3, O1	Chosen based on systematic search with the Genetic Algorithm	90.50%	Ours

5.3 Issues with Preprocessing the Dataset

The open-source dataset, SAM40, which was used in the project thesis, was collected using a 32-channel EEG set. The large number of channels enabled us to select up to 32 source components when applying ICA. Through experimentation, it was determined that 15 components were optimal for analysis. Running ICA twice resulted in improved classifier accuracy by generating cleaner EEG data. The resulting accuracy of 96.50% was very promising.

By applying the Genetic Algorithm to the ICA-filtered data, we were able to reduce the number of channels from 32 to 8 (75% reduction) while maintaining an accuracy of 90.50% (6.22% loss). These results were very promising for the work that would be done in this thesis. However, reducing electrodes in a filtered dataset yields different results from raw data with the same electrodes, especially when using ICA for preprocessing.

The same analysis was tried in this thesis. However, due to the limited number of channels that were recorded, the ICA model was built using 8 channels instead of 15. With such a small number of components for the analysis, there were fewer instances of pure noise components that were separated out by the analysis, as depicted in Figure 3.12. This happens because having fewer channels reduces the amount of information available to separate the independent sources, making it more difficult to accurately identify the underlying sources. Additionally, the presence of noise can further complicate the separation of independent sources, leading to less accurate results. The first round of ICA did not remove as much noise and artifacts as hoped, but did clean up the data to a certain extent. See Figure 3.15 for an example.

However, the second application of the Independent Component Analysis (ICA) algorithm did not produce any meaningful outcomes. During the initial round of ICA, 2-3 components were primarily removed before reconstructing the data. For the second round, a new ICA model was constructed with 8 components, but after removing 1-3 additional components, the model was unable to reconstruct the data due to the inadequate amount of data remaining in the signal. Consequently, the resulting signal was flattened, as depicted in Figure 5.1. As a result, the decision was made to abandon the double ICA technique and explore other methods for preprocessing and classifying the data. Thus, it is becoming evident that reducing the number of electrodes from 32 to 8 constraints the study too much, should one wish to keep the original artifact-removal technique as presented previously.

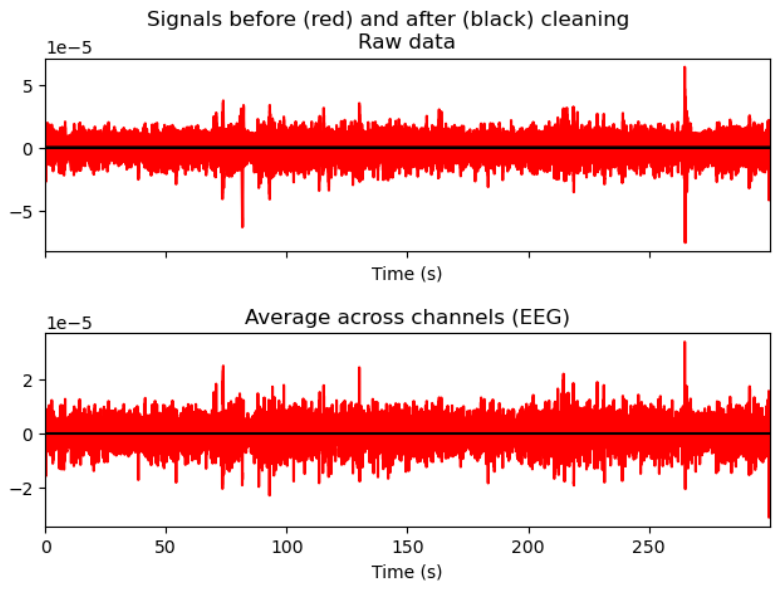


Figure 5.1: Results from removing ICA components during the second round of ICA. The reconstructed signal has flattened due to the inadequate amount of data remaining in the signal.

5.4 Results with KNN and SVM

Due to our limited dataset of only 8 recording channels, we had doubts about achieving satisfactory accuracy with our classification approach. To our surprise, both KNN and SVM classifiers yielded promising results with the raw data and various feature extractions. The best-performing combinations were:

- Full RAW data with SVM (87.50% accuracy) (Figure 4.1)
- Time series features of RAW data with SVM (86.84% accuracy) (Figure 4.2)

It is worth highlighting the performance of the full RAW dataset. However, despite its high accuracy levels, this approach had a runtime of approximately 1 hour, which is considerably longer compared to the feature dataset that only took about 5 minutes to complete. Consequently, utilizing feature extraction techniques, such as time series features, offers a more efficient solution without compromising the satisfactory accuracy levels achieved.

Additionally, when it was attempted to remove mildly stressed from the Stress Scale (SS) labeled dataset, the resulting dataset became significantly reduced in size, posing challenges for effective training. Consequently, the classifier's performance exhibited inconsistency in this scenario, and the results for the INIT and ICA datasets are thus added in Appendix B.

To test the reliability of the best-performing models, we attempted to predict on the SAM40 dataset as well. Since the SAM40 dataset was sampled at a different rate and duration, our dataset was adjusted accordingly. After running the classification method, two predictions were obtained: one for the original test set and one for the SAM40 data. Unfortunately, the SAM40 prediction only achieved a certain percentage of accuracy, indicating that the model lacks robustness and struggles to adapt to new data. This was quite disappointing and lowered our confidence in our previous findings.

The data exploration presented in section 3.3 reveals indications that the various recordings contain different amplitudes of noise. Consequently, considering the high accuracy achieved with the RAW data, and low accuracy elsewhere, it is plausible that the classifiers rely on the noise levels present in each recording to build their models. As previously stated, there was noticeable construction work nearby the recording room during the second recording, which is clearly picked up in the data. Given that none of the more processed datasets perform as effectively as the RAW dataset, this observation further supports our hypothesis.

Given the noise levels of our collected datasets, and the failed attempts to recreate the same artifact-removal technique as previously, it is difficult to say if the results are at all comparable to the ones obtained in the project thesis [1]. Ultimately, we believe that all results presented in chapter 4 have been negatively affected by the constrained number of electrodes used to record the data.

5.5 New Methods

5.5.1 New filtering approach and PSD

Extensive testing of hyperparameters, epoch length, feature extractions, and filtering methods was tried, but most did not outperform our initial testing. Especially the new dataset NEW_INIT and the PSD analysis of this data had disappointing results. Thus the matching results have been attached in Appendix B. Since the SS-label set proved to be too small after being converted to a binary label set, only STAI-Y-labels were tested with these methods.

5.5.2 EEGNet and TSGL-EEGNet

Both EEGNet and TSGL-EEGNet exhibited reasonably satisfactory performance, achieving mean accuracies of 73.68% (Figure 4.26) and 69.39% (Figure 4.27), respectively, even without extensive hyperparameter tuning. The findings suggest that our collected dataset contains valuable EEG information that to some extent captures stress signals. This is an encouraging outcome for future studies further investigating the potential correlation between EEG signals and mental stress. This also suggests that there is a potential for further improvement in classification accuracy through a more focused tuning of these models. However, it should be noted that performing such tuning would require a significant amount of time, which was not prioritized in the scope of this study. Nevertheless, exploring the possibilities of enhancing the performance of these models through hyperparameter optimization could be a valuable task for future research, potentially yielding higher classification accuracy in stress detection tasks.

5.5.3 Shallow CNN and Deep CNN

Despite EEGNet being a specialized Convolutional Neural Network for classifying EEG data, it remains intriguing to explore the performance of more conventional models, such as Deep Convolutional Neural Networks and Shallow Convolutional Neural Networks. Comparing the results in Figure 4.28 and Figure 4.29, it becomes apparent that the simpler Shallow CNN outperforms both the more complex Deep CNN and the models from EEGNet, with an impressive mean accuracy of 83.66% across the 10-fold cross-validation. Normally one would assume the more complex model to exhibit better performance in classifying complex signals, but this seems not the case here. A possible explanation for this observation might be attributed to the fact that both Convolutional Neural Network models use the RAW data as input. Since the RAW EEG signals contain a significant amount of noise, the Shallow CNN might have an advantage when summarizing the signal as it relies on simpler statistical measures.

5.5.4 Wavelet Scattering

As presented in Figure 4.30, wavelet scattering transform with SVM classifier performs (87.50% accuracy, 82.81% sensitivity, 90.91% specificity) approximately the same as the full RAW data with SVM (87.50% accuracy, 81.25% sensitivity, 92.05% specificity). Further, it also outperforms time series features of RAW data with SVM (86.84% accuracy, 82.81% sensitivity, 89.77% specificity). However, as previously mentioned in subsection 3.7.4 the runtime for wavelet scattering with SVM was over one hour. The effectiveness of this falls below the desired level. Yet, the high accuracy achieved indicates that the method could be viable for classifying stressed/non-stressed EEG signals.

There are also possibilities for further enhancement of the method, such as employing the wavelet scattering transform as an initial step in a CNN instead of utilizing the KNN and SVM classifiers. This could be especially interesting to further investigate as the wavelet scattering transform is comparable to a CNN. Regarding the high accuracy achieved with wavelet scattering, it should be mentioned that, just as with the other methods that also performed well with the RAW data, it is uncertain if the model is learning the characteristics of the noise levels rather than any meaningful features connected to the participant's stress levels. Further investigation will have to be done in order to possibly strengthen our results.

5.6 Future Work

For future research within Machine Learning-based stress detection from EEG signals, our thoughts are as follows:

- Our primary concern is that the low number of channels has constrained our study too much. We would recommend that future researchers collect a new dataset with 16 EEG channels instead of 8. As we found 8 channels to be too restricting for our study to reliably do artifact removal, we do have hope that 16 channels should be sufficient. If new data should be collected, we highly recommend that recordings be done in a controlled and noise-free area, and with more than enough time to adjust and fit the electrodes to each subject.
- The relatively short time span of the Master's thesis put a constraint on the amount of hyperparameter tuning we were able to explore. Thus, it did not allow for a thorough exploration of hyperparameter fine-tuning, especially when using CNN and EEGNet. Given the satisfactory results of both EEGNet models and the CNN models that were tested, we would recommend further investigation of these models.

From our results both in this thesis and the previous project thesis, we have a viable reason to suspect that EEG data *can* capture mental stress.

Chapter 6

Conclusion

In conclusion, the study found that Machine Learning classifiers (KNN, SVM, and EEGNet) with various feature extractions (especially time series features and wavelet scattering features) can effectively detect mental stress using Electroencephalography (EEG) data.

The best results using traditional classifiers were obtained using full RAW data with SVM (accuracy: 87.50%, sensitivity: 81.25%, specificity: 92.05%), time series features of RAW data with SVM (accuracy: 86.84%, sensitivity: 82.81%, specificity: 98.77%) and wavelet scattering features of RAW data with SVM (accuracy: 87.50%, sensitivity: 82.81%, specificity: 90.91%). Deep Convolutional Neural Network models also performed satisfactorily with the best-performing model being the Shallow CNN with a mean accuracy of 83.66% across all folds.

The unsatisfactory outcomes observed in the filtered versions of the dataset raise concerns and indicate underlying issues. Our hypothesis suggests that the second session recordings may have been affected by additional noise stemming from nearby construction work, and it is likely that the classifiers are relying on this noise to construct their models. These problems may root in the low number of recording electrodes, which in turn negatively influenced the filtering process, ultimately leading to poor accuracy of the filtered datasets.

However, the encouraging accuracy achieved by the CNN models instilled a renewed sense of optimism, as these models possess greater complexity and can identify deep-seated characteristics within the EEG data. The noteworthy mean accuracy attained by the Shallow CNN further reinforces the notion of a connection between mental stress and the raw EEG data we collected. These findings suggest that the raw EEG data holds significant potential in capturing relevant information related to mental stress.

Bibliography

[1] A. J. Y. Marthinsen, *Detection of mental stress from eeg data using artificial intelligence*, 2022. DOI: https://doi.org/10.13140/RG.2.2.27754.39360.

[2] K. Rasheed, A. Qayyum, J. Qadir, S. Sivathamboo, P. Kwan, L. Kuhlmann, T. O'Brien, and A. Razi, "Machine learning for predicting epileptic seizures using eeg signals: A review," *IEEE Reviews in Biomedical Engineering*, vol. 14, pp. 139–155, 2020.

[3] X.-W. Wang, D. Nie, and B.-L. Lu, "Emotional state classification from eeg data using machine learning approach," *Neurocomputing*, vol. 129, pp. 94–106, 2014.

[4] R. Katmah, F. Al-Shargie, U. Tariq, F. Babiloni, F. Al-Mughairbi, and H. Al-Nashash, "A review on mental stress assessment methods using eeg signal," *Sensors*, vol. 21, no. 15, 2021, ISSN: 1424-8220. DOI: https://doi.org/10.3390/s21155043. [Online]. Available: https://www.mdpi.com/1424-8220/21/15/5043.

[5] O. Attallah, *An effective mental stress state detection and evaluation system using minimum number of frontal brain electrodes*. DOI: https://10.3390/diagnostics10050292. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/32397517/.

[6] A. P. Assosiation, *Stress effects on the body*, 2018. [Online]. Available: https://www.apa.org/topics/stress/body.

[7] U. Malt and F. Svartdal, *Stress*, Store Norske Leksikon. [Online]. Available: https://snl.no/stress.

[8] A. Crosswell and K. Lockwood, *Best practices for stress measurement: How to measure psychological stress in health research*. DOI: https://doi.org/10.1177/2055102920933072. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7359652/.

[9] V. Aspiotis and et. al., *Assessing electroencephalography as a stress indicator: A vr high-altitude scenario monitored through eeg and ecg*, Sensors, 2022. DOI: https://doi.org/10.3390/s22155792. [Online]. Available: https://www.mdpi.com/1424-8220/22/15/5792.

[10] D. F. Sharbrough, D. G.-E. Chatrian, D. R. P. Lesser, D. H. Lüders, D. M. Nuwer, and D. T. W. Picton, *American electricalencephalographic society guidelines for standard electrode position nomenclature*. [Online]. Available: https://journals.lww.com/clinicalneurophys/Citation/1991/04000/American_Electroencephalographic_Society.7.aspx.

[11] N. Choudhury, P. Das, N. Deb, P. Dutta, R. Ghosh, S. Kashyap, S. Phadikar, A. Phukan, R. Saha, K. Sengupta, and N. Sinha, *Sam 40: Dataset of 40 subject eeg recordings to monitor the induced-stress while performing stroop color-word test, arithmetic task, and mirror image recognition task*. DOI: https://doi.org/10.1016/j.dib.2021.107772. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352340921010465?via%5C%3Dihub.

[12] A. J. Newman, *Artifacts in eeg data*, Data Science for Psychology and Neuroscience - in Python. [Online]. Available: https://neuraldatascience.io/7-eeg/erp_artifacts.html.

[13] A. J. Newman, *Filtering eeg data*, Data Science for Psychology and Neuroscience - in Python. [Online]. Available: https://neuraldatascience.io/7-eeg/erp_filtering.html.

[14] L. J. Gonçales, K. Farias, L. Kupssinskü, and M. Segalotto, "The effects of applying filters on eeg signals for classifying developers' code comprehension," DOI: 10.14482/INDES.30.1.303.661. [Online]. Available: https://www.redalyc.org/journal/474/47471710003/html/#:~:text=Studies%20generally%20use%20a%20bandpass,an%20approximate%20frequency%20can%20occur..

[15] R. W. Schafer, *What Is a savitzky-golay filter?* DOI: https://doi.org/10.1109/MSP.2011.941097. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5888646.

[16] M. A. Bee and C. Micheyl, *The "cocktail party problem": What is it? how can it be solved? and why should animal behaviorists study it?* DOI: https://10.1037/0735-7036.122.3.235. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2692487/.

[17] T.-P. Jung, H. C. Makeig S, T. Lee, M. McKeown, V. Iragui, and T. Sejnowski, *Removing electroencephalographic artifacts by blind source separation*. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/10731767/.

[18] T.-P. Jung, W. W. Makeig S, E. Courchesne, and T. Sejnowski, *Removal of eye activity artifacts from visual event-related potentials in normal and clinical subjects*. DOI: https://doi.org/10.1016/s1388-2457(00)00386-2. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/11018488/.

[19] scikit-learn developers (BSD License), *3.1. cross-validation: Evaluating estimator performance*. [Online]. Available: https://scikit-learn.org/stable/modules/cross_validation.html/.

[20] scikit-learn Developers, *What is the k-nearest neighbours algorithm?* [Online]. Available: https://www.ibm.com/topics/knn.

[21] R. Gandhi, *Support vector machine — introduction to machine learning algorithms*, Towards Data Science. [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47g.

[22] D. A. Pisner and D. M. Schnyer, "Chapter 6 - support vector machine," in *Machine Learning*, A. Mechelli and S. Vieira, Eds., Academic Press, 2020, pp. 101–121, ISBN: 978-0-12-815739-8. DOI: https://doi.org/10.1016/B978-0-12-815739-8.00006-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128157398000067.

[23] V. J. Lawhern, A. J. Solon, N. R. Waytowich, S. M. Gordon, C. P. Hung, and B. J. Lance, "Eegnet: A compact convolutional neural network for eeg-based brain–computer interfaces," *Journal of Neural Engineering*, vol. 15, no. 5, p. 056 013, 2018. [Online]. Available: http://stacks.iop.org/1741-2552/15/i=5/a=056013.

[24] P. Schaldenbrand, *What is a power spectral density (psd)?* Community Knowledge Article. [Online]. Available: https://community.sw.siemens.com/s/article/what-is-a-power-spectral-density-psd.

[25] M. Demuru, S. L. C. Maurizio, S. M. Pani, and M. Fraschini, *A comparison between power spectral density and network metrics: An eeg study*. DOI: 10.1016/j.bspc.2019.101760. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S1746809419303416.

[26] S. Narkhede, *Understanding confusion matrix*, Towards Data Science. [Online]. Available: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

[27] B. L. GU, R. Ghosh, N. Deb, K. Sengupta, A. Phukan, and N. Choudhury, *Sam 40: Dataset of 40 subject eeg recordings to monitor the induced-stress while performing stroop color-word test, arithmetic task, and mirror image recognition task*. 2021. DOI: https://doi.org/10.6084/m9.figshare.14562090.v1.

[28] P. J. Lynch, *File:thoracic landmarks anterior view.svg*, Chest landmarks, for radiography and other chest imaging techniques. [Online]. Available: https://commons.wikimedia.org/wiki/File:Thoracic_landmarks_anterior_view.svg.

[29] G. Jun and K. G. Smitha, "Eeg based stress level identification," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2016, pp. 003 270–003 274. DOI: 10.1109/SMC.2016.7844738.

[30] R. Khosrowabadi, C. Quek, K. K. Ang, S. W. Tung, and M. Heijnen, "A brain-computer interface for classifying eeg correlates of chronic mental stress," pp. 757–762, 2011.

[31] F. Al-shargie, T. B. Tang, N. Badruddin, and M. Kiguchi, “Simultaneous measurement of eeg-fnirs in classifying and localizing brain activation to mental stress,” in *2015 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, 2015, pp. 282–286. DOI: 10.1109/ICSIPA.2015.7412205.

[32] Y. Zhang, Q. Wang, Z. Y. Chin, and K. K. Ang, “Investigating different stress-relief methods using electroencephalogram (eeg),” in *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, IEEE, 2020, pp. 2999–3002.

[33] F. Al-Shargie, M. Kiguchi, N. Badruddin, S. C. Dass, A. F. M. Hani, and T. B. Tang, “Mental stress assessment using simultaneous measurement of eeg and fnirs,” *Biomedical optics express*, vol. 7, no. 10, pp. 3882–3898, 2016.

[34] A. Secerbegovic, S. Ibric Hodzic, J. Nisic, N. Suljanovic, and A. Mujcic, “Mental workload vs. stress differentiation using single-channel eeg,” in Mar. 2017, pp. 511–515, ISBN: 978-981-10-4165-5. DOI: 10.1007/978-981-10-4166-2_78.

[35] O. Kayikcioglu, S. Bilgin, G. Seymenoglu, and A. Devecib, *State and trait anxiety scores of patients receiving intravitreal injections*. DOI: 10.1159/000478993. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6945947/.

[36] C. Sletten, *Automated stress detection using electroencephalogram signals*, 2022.

[37] R. T. Schirrmeister, J. T. Springenberg, L. Dominique, J. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball, *Deep learning with convolutional neural networks for eeg decoding and visualization*. DOI: 10.1002/hbm.23730. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/hbm.23730.

[38] X. Deng, B. Zhang, N. Yu, K. Liu, and K. Sun, “Advanced tsgl-eegnet for motor imagery eeg-based brain-computer interfaces,” *IEEE Access*, vol. 9, pp. 25 118–25 130, 2021. DOI: 10.1109/ACCESS.2021.3056088.

[39] K. K. Ang, Z. Y. Chin, H. Zhang, and C. Guan, “Filter bank common spatial pattern (fbcsp) in brain-computer interface,” pp. 2390–2397, 2008. DOI: 10.1109/IJCNN.2008.4634130.

[40] A. B. Buriro, B. Ahmed, G. Baloch, J. Ahmed, R. Shoorangiz, S. J. Weddell, and R. D. Jones, “Classification of alcoholic eeg signals using wavelet scattering transform-based features,” *Computers in biology and medicine*, vol. 139, p. 104 969, 2021.

[41] M. Andreux, T. Angles, G. Exarchakis, R. Leonarduzzi, G. Rochette, L. Thiry, J. Zarka, S. Mallat, J. andén, E. Belilovsky, J. Bruna, V. Lostanlen, M. Chaudhary, M. J. Hirn, E. Oyallon, S. Zhang, C. Cella, and M. Eickenberg, *Kymatio: Scattering transforms in python*, 2022. arXiv: 1812.11214 [cs.LG].

Appendix A

Code implementation

A.0.1 `genetic_alg.py`

```
import numpy as np
import random
import pandas as pd

from dataset import load_dataset, load_labels, convert_to_epochs, load_channels
from features import time_series_features, hjorth_features
from classifiers import KNN, SVM, NN
import variables as v

# This project is extended and a library called PyGAD is released to build the
# genetic algorithm.
# PyGAD documentation: https://pygad.readthedocs.io
# Install PyGAD: pip install pygad
# PyGAD source code at GitHub: https://github.com/ahmedfgad/GeneticAlgorithmPython

def cal_pop_fitness(equation_inputs, pop):
    # Calculating the fitness value of each solution in the current population.
    # The fitness function calculates the sum of products between each input and
    # its corresponding weight.
    # In our case this is 4*accuracy + 1*sensitivity + 1*specificity
    fitness = np.sum(pop*equation_inputs, axis=1)
    return fitness

def select_mating_pool(pop, fitness, num_parents):
    # Selecting the best individuals in the current generation as parents for
    # producing the offspring of the next generation.
    parents = np.empty((num_parents, pop.shape[1]), dtype='<U5')
    print(parents)
    for parent_num in range(num_parents):
        max_fitness_idx = np.where(fitness == np.max(fitness))
        max_fitness_idx = max_fitness_idx[0][0]
        parents[parent_num, :] = pop[max_fitness_idx, :]
        fitness[max_fitness_idx] = -99999999999
    return parents

def crossover(parents, offspring_size):
    # Produces offspring with a random combination of the two parents' genes
    n_offspring = offspring_size[0]
    n_genes_in_person = int(offspring_size[1])
    offspring = np.empty(offspring_size, dtype='<U5')
    # The point at which crossover takes place between two parents.
    # Usually it is at the center.
    crossover_point = round(n_genes_in_person/2)

    offspring_indx = 0
    for i in range(parents.shape[0]):
        for j in range(i+1, parents.shape[0]):
            gene_indx = 0
            while gene_indx < n_genes_in_person:
                if gene_indx < crossover_point:
                    rand_int = random.randint(0,7)
                    if parents[i][rand_int] not in offspring[offspring_indx]:
                        offspring[offspring_indx][gene_indx] = parents[i][rand_int]
                        gene_indx += 1
                else:
                    rand_int = random.randint(0,7)
                    if parents[j][rand_int] not in offspring[offspring_indx]:
                        offspring[offspring_indx][gene_indx] = parents[j][rand_int]
                        gene_indx += 1
            offspring_indx += 1
    return offspring


def make_init_pop(all_data, all_genes, num_genes_in_person, num_people):
    # Makes a random first population
    # Initialize empty population
    init_pop = np.empty([num_people,num_genes_in_person], dtype='<U5')
    person_index = 0

    while person_index!=num_people:
        # Initialize new person
        person = np.empty(num_genes_in_person, dtype='<U5')
        gene_index = 0

        while gene_index!=num_genes_in_person:
            # Gives a random index
            index = random.randint(0,len(all_genes)-1)
            # Checks if the gene is not already in the gene pool of the person
            if all_genes[index] not in person:
                person[gene_index] = all_genes[index]
                gene_index += 1
        init_pop[person_index] = person
        person_index +=1

    # Create labels to match the dataset
    # Creating labels
    subset_data = get_subset(all_data, all_genes, init_pop[0])
    dataset = convert_to_epochs(subset_data, num_genes_in_person, v.SFREQ)
    label = create_labels(dataset)
    return init_pop, label


def get_subset(data, all_genes, subset_genes):
    # Retrieves the data that belongs to the subset of genes
    subset_data = np.empty((120, 8, 3200))
    n_genes = 8

    j = 0
    for i in range(len(all_genes)):
        if j < (n_genes + 1) and all_genes[i] in subset_genes:
            subset_data[:,j,:] = data[:,i,:]
            j+=1
    return subset_data


def check_nan(array):
    # Checks if there is any NaN values in array
    # Used for debugging
    x = np.isnan(array)
    if True in x:
        print('NAN␣in␣array')
        return 0
    print('No␣NAN␣found')

def create_labels(dataset):
    # Loads labels into the correct shape
    labels = load_labels()
    label = pd.concat([labels['t1_math'], labels['t2_math'],
                labels['t3_math']]).to_numpy()
    label = label.repeat(dataset.shape[1])
    return label


def convert_pop_to_fitness(all_data, all_channels, current_pop, label, n_genes):
    # Calculates population fitness (accuracy, sensitivity, specificity)
    data = np.empty((3000, 16))
    new_pop_fitness = np.empty((len(current_pop),3))

    for i in range(len(current_pop)):
        subset_data = get_subset(all_data, all_channels, current_pop[i])
        dataset = convert_to_epochs(subset_data, n_genes, v.SFREQ)
        # hjorth features perform the best
        features = hjorth_features(dataset, n_genes,v.SFREQ)
        data = features

        # Use KNN or SVM
        new_pop_fitness[i] = KNN(data, label)
    return new_pop_fitness

def convert_parents_to_fitness(all_data, all_genes, parents, label,
                               num_parents_mating, n_genes):
     # Calculates parents' fitness (accuracy, sensitivity, specificity)
    new_pop_fitness = np.empty((num_parents_mating,3))
    data = np.empty((3000, 16))

    for i in range(num_parents_mating):

        subset_data = get_subset(all_data, all_genes, parents[i])
        dataset = convert_to_epochs(subset_data, n_genes, v.SFREQ)
        features = hjorth_features(dataset, n_genes,v.SFREQ)
        data = features
        print(f'Parent␣genes:␣{parents[i]}␣')
        results = KNN(data, label)
        print(results)
        new_pop_fitness[i] = results

    return new_pop_fitness
```

A.0.2 `channel_selection.ipynb`

```python
import genetic_alg as ga
import variables as v
from dataset import load_dataset, load_labels, convert_to_epochs, load_channels
from features import time_series_features, hjorth_features
from classifiers import KNN, SVM, NN

#importing ICA filtered two times
dataset_ica_2_ = load_dataset(data_type="ica2", test_type="Arithmetic")
channels = load_channels()
labels = ga.create_labels(convert_to_epochs(dataset_ica_2_, 32, v.SFREQ))

num_generations = 10
num_genes_in_person = 8
num_parents_mating = 5
num_people_in_pop = 15


equation_inputs = [1.5, 1, 1]  # weight for accuracy, sensitivity and specificity
init_pop, label = ga.make_init_pop(dataset_ica_2_, channels, num_genes_in_person,
num_people_in_pop)
print(init_pop)

pop_size = init_pop.shape[0]
curr_pop = init_pop

for generation in range(num_generations):
     print(f'Generation␣number:␣{generation}')
     # Measuring the fitness of each chromosome in the population.
     curr_pop_fitness = ga.convert_pop_to_fitness(dataset_ica_2_, channels,
     curr_pop, label, num_genes_in_person)
     fitness = ga.cal_pop_fitness(equation_inputs, curr_pop_fitness)
     print(fitness)

    # Selecting the best parents in the population for mating.
     parents = ga.select_mating_pool(curr_pop, fitness, num_parents_mating)

     # Generating next generation using the crossover.
     offspring_crossover = ga.crossover(parents, offspring_size=
     (pop_size-num_parents_mating, num_genes_in_person))

     # Creating the new population based on the parents and offspring.
     curr_pop[0:num_parents_mating, :] = parents
     curr_pop[num_parents_mating:, :] = offspring_crossover
```

A.0.3 `classifiers.py`

```
import numpy as np
from matplotlib import pyplot as plt
from utils.EEGModels import EEGNet, TSGLEEGNet, DeepConvNet,
                            ShallowConvNet, TSGLEEGNet
import utils.variables as v
import matplotlib.pyplot as plt
import utils.metrics as m

from sklearn.model_selection import GridSearchCV, PredefinedSplit
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import StratifiedKFold

import plotly.graph_objects as go
import numpy as np
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint
import plotly.subplots as p


def knn_classification(train_data, test_data, train_labels, test_labels):
    param_grid = {
        'leaf_size': range(1, 10),
        'n_neighbors': range(1, 5),
        'p': [1, 2]
    }
    scaler = StandardScaler()
    train_data = scaler.fit_transform(train_data)
    test_data = scaler.transform(test_data)

    knn_clf = GridSearchCV(KNeighborsClassifier(), param_grid, refit=True,
                            n_jobs=-1, cv = 10)
    knn_clf.fit(train_data, train_labels)

    y_pred = knn_clf.predict(test_data)
    y_true = test_labels

    print(knn_clf.best_estimator_)
    print(knn_clf.best_params_)
    results = knn_clf.cv_results_
    print(results)

    # extract the relevant scores
    leaf_sizes = results['param_leaf_size'].data
    n_neighbors = results['param_n_neighbors'].data
    accuracies = results['mean_test_score']

    print('Number␣of␣results:', len(accuracies))
    #print('n_neighbors:', n_neighbors)
    #print('leaf_sizes:', leaf_sizes)
    print('overall␣accuracy:', np.round(np.sum(accuracies)/len(accuracies)*100,2))
    # plot the results
    plt.figure(1)
    plt.plot(
        range(len(accuracies)),
        accuracies,
    )
    plt.xlabel('Iteration')
    plt.ylabel('Accuracy')
    plt.show()

    conf_matrix = metrics.confusion_matrix(y_true, y_pred)
    m.plot_conf_matrix_and_stats(conf_matrix)



def svm_classification(train_data, test_data, train_labels, test_labels):
    param_grid = {
        'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000, 1000000],
        'kernel': ['linear', 'poly', 'rbf', 'sigmoid']
    }
    scaler = RobustScaler()
    train_data = scaler.fit_transform(train_data)
    test_data = scaler.transform(test_data)

    svm_clf = GridSearchCV(SVC(), param_grid, refit=True, n_jobs=-1, cv = 10)
    svm_clf.fit(train_data, train_labels)

    y_pred = svm_clf.predict(test_data)
    y_true = test_labels

    print(svm_clf.best_estimator_)
    print(svm_clf.best_params_)

    # fit the grid search to get the results
    results = svm_clf.cv_results_
    print(results)

    # extract the relevant scores
    C_values = results['param_C'].data
    kernel_values = results['param_kernel'].data
    accuracies = results['mean_test_score']

    print('Number␣of␣results:', len(accuracies))
    #print('C_values:', C_values)
    #print('kernel_values:', kernel_values)
    print('overall␣accuracy:', np.round(np.sum(accuracies)/len(accuracies)*100,2))
    # plot the results
    plt.figure(2)
    plt.plot(
        range(len(accuracies)),
        accuracies
    )
    plt.xlabel('Iteration')
    plt.ylabel('Accuracy')
    plt.show()

    conf_matrix = metrics.confusion_matrix(y_true, y_pred)
    m.plot_conf_matrix_and_stats(conf_matrix)
```

```python
def svm_classification_SAM40(train_data, test_data, SAM40_data, train_labels,
                            test_labels, SAM40_labels):
    param_grid = {
        'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000, 1000000],
        'kernel': ['linear', 'poly', 'rbf', 'sigmoid']
    }
    print('Scaling␣training␣and␣testing␣data')
    scaler = RobustScaler()
    train_data = scaler.fit_transform(train_data)
    test_data = scaler.transform(test_data)

    print('Scaling␣SAM40␣data')
    SAM40_scaler = RobustScaler()
    SAM40_data = SAM40_scaler.fit_transform(SAM40_data)

    print('Finding␣the␣best␣model')
    svm_clf = GridSearchCV(SVC(), param_grid, refit=True, n_jobs=-1, cv = 10)
    svm_clf.fit(train_data, train_labels)

    print(svm_clf.best_estimator_)
    print(svm_clf.best_params_)

    print('Predicting␣on␣test␣data')
    y_pred = svm_clf.predict(test_data)
    y_true = test_labels

    # fit the grid search to get the results
    results = svm_clf.cv_results_

    # extract the relevant scores
    C_values = results['param_C'].data
    kernel_values = results['param_kernel'].data
    accuracies = results['mean_test_score']

    print('Number␣of␣results:', len(accuracies))
    #print('C_values:', C_values)
    #print('kernel_values:', kernel_values)
    print('accuracies:', accuracies)
    # plot the results
    plt.figure(2)
    plt.plot(
        range(len(accuracies)),
        accuracies
    )
    plt.xlabel('Iteration')
    plt.ylabel('Accuracy')
    plt.show()

    conf_matrix = metrics.confusion_matrix(y_true, y_pred)
    m.plot_conf_matrix_and_stats(conf_matrix)

    #SAM40
    print('Predicting␣on␣SAM40␣data')
    y_pred_SAM40 = svm_clf.predict(SAM40_data)
    y_true_SAM40 = SAM40_labels
    conf_matrix_SAM40 = metrics.confusion_matrix(y_true_SAM40, y_pred_SAM40)
    m.plot_conf_matrix_and_stats(conf_matrix_SAM40)
```

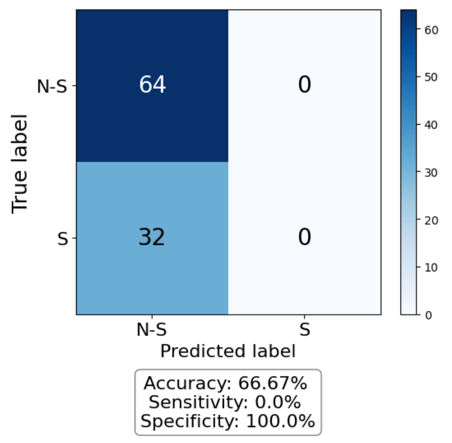
```
def kfold_EEGNet_classification(train_data, test_data, train_labels, test_labels,
                                n_folds, data_type, epoched = True):

    if epoched:
        if data_type == 'new_ica':
            model = EEGNet(nb_classes = 2, Chans = v.NUM_CHANNELS ,
                           Samples = v.EPOCH_LENGTH*v.NEW_SFREQ,
                           dropoutRate = 0.5, kernLength = 32, F1 = 8,
                           D = 2, F2 = 16, dropoutType = 'Dropout')
        else:
            model = EEGNet(nb_classes = 2, Chans = v.NUM_CHANNELS ,
                           Samples = v.EPOCH_LENGTH*v.SFREQ,
                           dropoutRate = 0.5, kernLength = 32, F1 = 8,
                           D = 2, F2 = 16, dropoutType = 'Dropout')
    else: #if not epoched
        if data_type == 'new_ica':
            model = EEGNet(nb_classes = 2, Chans = v.NUM_CHANNELS ,
                           Samples = v.NEW_NUM_SAMPLES,
                           dropoutRate = 0.5, kernLength = 32, F1 = 8,
                           D = 2, F2 = 16, dropoutType = 'Dropout')
        else:
            model = EEGNet(nb_classes = 2, Chans = v.NUM_CHANNELS ,
                           Samples = v.NUM_SAMPLES,
                           dropoutRate = 0.5, kernLength = 32, F1 = 8,
                           D = 2, F2 = 16, dropoutType = 'Dropout')

    model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
                  metrics = ['accuracy'])

    numParams    = model.count_params()

    checkpointer = ModelCheckpoint(filepath='/tmp/checkpoint.h5', verbose=1,
                                   save_best_only=True)

    class_weights = {0:1, 1:3}

    # Split into k-folds
    skf = StratifiedKFold(n_splits=n_folds)
    total_accuracy = 0

    for fold, (train_index, val_index) in enumerate(
                                          skf.split(train_data, train_labels)):
        print(f"\nFold␣nr:␣{fold+1}")
        train_data_fold = train_data[train_index]
        train_labels_fold = train_labels[train_index]
        val_data_fold = train_data[val_index]
        val_labels_fold = train_labels[val_index]

        history = model.fit(train_data_fold, train_labels_fold, batch_size = None,
                            epochs = 30, verbose = 2, validation_data =
                            (val_data_fold, val_labels_fold), callbacks =
                            [checkpointer], class_weight = class_weights)

        # load optimal weights
        model.load_weights('/tmp/checkpoint.h5')

        probs       = model.predict(test_data)
        preds       = probs.argmax(axis = -1)

        conf_matrix = metrics.confusion_matrix(test_labels, preds)
        m.plot_conf_matrix_and_stats(conf_matrix)

        # Plot Loss/Accuracy over time
        # Create figure with secondary y-axis
        fig = p.make_subplots(specs=[[{"secondary_y": True}]])
        # Add traces
        fig.add_trace(go.Scatter(y=history.history['val_loss'], name="val_loss"),
                                 secondary_y=False)
        fig.add_trace(go.Scatter(y=history.history['loss'], name="loss"),
                                 secondary_y=False)
        fig.add_trace(go.Scatter(y=history.history['val_accuracy'],
                                 name="val␣accuracy"), secondary_y=True)
        fig.add_trace(go.Scatter(y=history.history['accuracy'], name="accuracy"),
                                 secondary_y=True)
        # Add figure title
        fig.update_layout(title_text="Loss/Accuracy␣of␣k-folds␣EEGNet")
        # Set x-axis title
        fig.update_xaxes(title_text="Epoch")
        # Set y-axes titles
        fig.update_yaxes(title_text="Loss", secondary_y=False)
        fig.update_yaxes(title_text="Accuracy", secondary_y=True)
        fig.show()


def kfold_TSGL_classification(train_data, test_data, train_labels, test_labels,
                              n_folds, data_type, epoched = True):

    if epoched:
        if data_type == 'new_ica':
            model = TSGLEEGNet(nb_classes = 2, Chans = v.NUM_CHANNELS,
                               Samples = v.EPOCH_LENGTH*v.NEW_SFREQ,
                               dropoutRate = 0.5, kernLength = 128, F1 = 96,
                               D = 1, F2 = 96, dropoutType = 'Dropout')
        else:
            model = TSGLEEGNet(nb_classes = 2, Chans = v.NUM_CHANNELS,
                               Samples = v.EPOCH_LENGTH*v.SFREQ,
                               dropoutRate = 0.5, kernLength = 128, F1 = 96,
                               D = 1, F2 = 96, dropoutType = 'Dropout')
    else: #if not epoched
        if data_type == 'new_ica':
           model = TSGLEEGNet(nb_classes = 2, Chans = v.NUM_CHANNELS,
                              Samples = v.NEW_NUM_SAMPLES,
                              dropoutRate = 0.5, kernLength = 128, F1 = 96,
                              D = 1, F2 = 96, dropoutType = 'Dropout')
        else:
            model = TSGLEEGNet(nb_classes = 2, Chans = v.NUM_CHANNELS,
                               Samples = v.NUM_SAMPLES,
                               dropoutRate = 0.5, kernLength = 128, F1 = 96,
                               D = 1, F2 = 96, dropoutType = 'Dropout')

    # compile the model and set the optimizers
    model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
                  metrics = ['accuracy'])

    # count number of parameters in the model
    numParams    = model.count_params()

    # set a valid path for your system to record model checkpoints
    checkpointer = ModelCheckpoint(filepath='/tmp/checkpoint.h5', verbose=1,
                                   save_best_only=True)
    class_weights = {0:1, 1:3}

     # Split into k-folds
    skf = StratifiedKFold(n_splits=n_folds)

    for fold, (train_index, val_index) in enumerate(
                                          skf.split(train_data, train_labels)):
        print(f"\nFold␣nr:␣{fold+1}")
        train_data_fold = train_data[train_index]
        train_labels_fold = train_labels[train_index]
        val_data_fold = train_data[val_index]
        val_labels_fold = train_labels[val_index]

        history = model.fit(train_data_fold, train_labels_fold, batch_size = None,
                            epochs = 30, verbose = 2, validation_data =
                            (val_data_fold, val_labels_fold), callbacks =
                            [checkpointer], class_weight = class_weights)

        # load optimal weights
        model.load_weights('/tmp/checkpoint.h5')

        probs       = model.predict(test_data)
        preds       = probs.argmax(axis = -1)

        conf_matrix = metrics.confusion_matrix(test_labels, preds)
        m.plot_conf_matrix_and_stats(conf_matrix)

        # Plot Loss/Accuracy over time
        # Create figure with secondary y-axis
        fig = p.make_subplots(specs=[[{"secondary_y": True}]])
        # Add traces
        fig.add_trace(go.Scatter(y=history.history['val_loss'], name="val_loss"),
                                 secondary_y=False)
        fig.add_trace(go.Scatter(y=history.history['loss'], name="loss"),
                                 secondary_y=False)
        fig.add_trace(go.Scatter(y=history.history['val_accuracy'],
                                 name="val␣accuracy"), secondary_y=True)
        fig.add_trace(go.Scatter(y=history.history['accuracy'], name="accuracy"),
                                 secondary_y=True)
        # Add figure title
        fig.update_layout(title_text="Loss/Accuracy␣of␣k-folds␣EEGNet")
        # Set x-axis title
        fig.update_xaxes(title_text="Epoch")
        # Set y-axes titles
        fig.update_yaxes(title_text="Loss", secondary_y=False)
        fig.update_yaxes(title_text="Accuracy", secondary_y=True)
        fig.show()


def kfold_DeepConvNet_classification(train_data, test_data, train_labels,
                                     test_labels, n_folds, data_type,
                                     epoched = True):
    if epoched:
        if data_type == 'new_ica':
            model = DeepConvNet(nb_classes = 2, Chans = v.NUM_CHANNELS,
                                Samples = v.EPOCH_LENGTH*v.NEW_SFREQ,
                                dropoutRate = 0.5)
        else:
            model = DeepConvNet(nb_classes = 2, Chans = v.NUM_CHANNELS,
                                Samples = v.EPOCH_LENGTH*v.SFREQ,
                                dropoutRate = 0.5)
    else: #if not epoched
        if data_type == 'new_ica':
           model = DeepConvNet(nb_classes = 2, Chans = v.NUM_CHANNELS,
                               Samples = v.NEW_NUM_SAMPLES,
                               dropoutRate = 0.5)
        else:
            model = DeepConvNet(nb_classes = 2, Chans = v.NUM_CHANNELS,
                                Samples = v.NUM_SAMPLES,
                                dropoutRate = 0.5)

    # compile the model and set the optimizers
    model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
                metrics = ['accuracy'])

    # count number of parameters in the model
    numParams    = model.count_params()

    # set a valid path for your system to record model checkpoints
    checkpointer = ModelCheckpoint(filepath='/tmp/checkpoint.h5', verbose=1,
                                save_best_only=True)

    class_weights = {0:1, 1:3}

     # Split into k-folds
    skf = StratifiedKFold(n_splits=n_folds)

    for fold, (train_index, val_index) in enumerate(
                                          skf.split(train_data, train_labels)):
        print(f"\nFold␣nr:␣{fold+1}")
        train_data_fold = train_data[train_index]
        train_labels_fold = train_labels[train_index]
        val_data_fold = train_data[val_index]
        val_labels_fold = train_labels[val_index]

       history = model.fit(train_data_fold, train_labels_fold, batch_size = None,
                            epochs = 30, verbose = 2, validation_data =
                            (val_data_fold, val_labels_fold), callbacks =
                            [checkpointer], class_weight = class_weights)

        # load optimal weights
        model.load_weights('/tmp/checkpoint.h5')

        probs       = model.predict(test_data)
        preds       = probs.argmax(axis = -1)

        conf_matrix = metrics.confusion_matrix(test_labels, preds)
        m.plot_conf_matrix_and_stats(conf_matrix)

       # Plot Loss/Accuracy over time
        # Create figure with secondary y-axis
        fig = p.make_subplots(specs=[[{"secondary_y": True}]])
        # Add traces
        fig.add_trace(go.Scatter(y=history.history['val_loss'], name="val_loss"),
                                 secondary_y=False)
        fig.add_trace(go.Scatter(y=history.history['loss'], name="loss"),
                                 secondary_y=False)
        fig.add_trace(go.Scatter(y=history.history['val_accuracy'],
                                 name="val␣accuracy"), secondary_y=True)
        fig.add_trace(go.Scatter(y=history.history['accuracy'], name="accuracy"),
                             secondary_y=True)
        # Add figure title
        fig.update_layout(title_text="Loss/Accuracy␣of␣k-folds␣EEGNet")
        # Set x-axis title
        fig.update_xaxes(title_text="Epoch")
        # Set y-axes titles
        fig.update_yaxes(title_text="Loss", secondary_y=False)
        fig.update_yaxes(title_text="Accuracy", secondary_y=True)
        fig.show()

def kfold_ShallowConvNet_classification(train_data, test_data, train_labels,
                                        test_labels, n_folds, data_type,
                                        epoched = True):

    if epoched:
        if data_type == 'new_ica':
            model = ShallowConvNet(nb_classes = 2, Chans = v.NUM_CHANNELS,
                                   Samples = v.EPOCH_LENGTH*v.NEW_SFREQ,
                                   dropoutRate = 0.5)
        else:
            model = ShallowConvNet(nb_classes = 2, Chans = v.NUM_CHANNELS,
                                   Samples = v.EPOCH_LENGTH*v.SFREQ,
                                   dropoutRate = 0.5)
    else: #if not epoched
        if data_type == 'new_ica':
           model = ShallowConvNet(nb_classes = 2, Chans = v.NUM_CHANNELS,
                                  Samples = v.NEW_NUM_SAMPLES,
                                  dropoutRate = 0.5)
        else:
            model = ShallowConvNet(nb_classes = 2, Chans = v.NUM_CHANNELS,
                                   Samples = v.NUM_SAMPLES,
                                   dropoutRate = 0.5)

    # compile the model and set the optimizers
    model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
                  metrics = ['accuracy'])

    # count number of parameters in the model
    numParams    = model.count_params()

    # set a valid path for your system to record model checkpoints
    checkpointer = ModelCheckpoint(filepath='/tmp/checkpoint.h5', verbose=1,
                                   save_best_only=True)

    class_weights = {0:1, 1:3}

     # Split into k-folds
    skf = StratifiedKFold(n_splits=n_folds)
    for fold, (train_index, val_index) in enumerate(
                                          skf.split(train_data, train_labels)):
        print(f"\nFold␣nr:␣{fold+1}")
        train_data_fold = train_data[train_index]
        train_labels_fold = train_labels[train_index]
        val_data_fold = train_data[val_index]
        val_labels_fold = train_labels[val_index]

        history = model.fit(train_data_fold, train_labels_fold, batch_size = None,
                            epochs = 30, verbose = 2, validation_data =
                            (val_data_fold, val_labels_fold), callbacks =
                            [checkpointer], class_weight = class_weights)

        # load optimal weights
        model.load_weights('/tmp/checkpoint.h5')

        probs       = model.predict(test_data)
        preds       = probs.argmax(axis = -1)

        conf_matrix = metrics.confusion_matrix(test_labels, preds)
        m.plot_conf_matrix_and_stats(conf_matrix)

        # Plot Loss/Accuracy over time
        # Create figure with secondary y-axis
        fig = p.make_subplots(specs=[[{"secondary_y": True}]])
        # Add traces
        fig.add_trace(go.Scatter(y=history.history['val_loss'], name="val_loss"),
                                 secondary_y=False)
        fig.add_trace(go.Scatter(y=history.history['loss'], name="loss"),
                                 secondary_y=False)
        fig.add_trace(go.Scatter(y=history.history['val_accuracy'],
                                 name="val␣accuracy"), secondary_y=True)
        fig.add_trace(go.Scatter(y=history.history['accuracy'], name="accuracy"),
                                 secondary_y=True)
        # Add figure title
        fig.update_layout(title_text="Loss/Accuracy␣of␣k-folds␣EEGNet")
        # Set x-axis title
        fig.update_xaxes(title_text="Epoch")
        # Set y-axes titles
        fig.update_yaxes(title_text="Loss", secondary_y=False)
        fig.update_yaxes(title_text="Accuracy", secondary_y=True)
        fig.show()
```

Appendix B

Additonal method performance

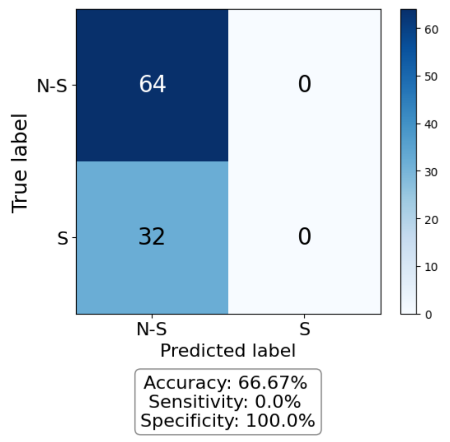
B.1 Performance of INIT Data with KNN and SVM with SS-Labels

As the results were less impressive for the SS-labeled dataset, it was decided to include the remaining results in the appendix.



(a) Performance of KNN with:

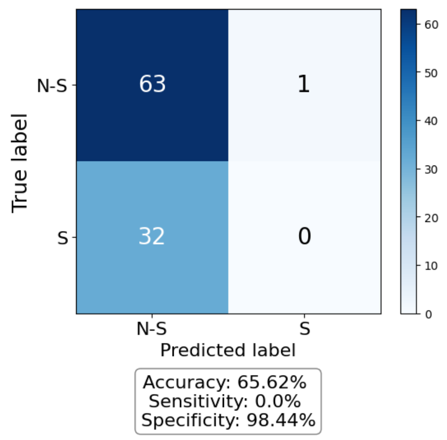
- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 1`



(b) Performance of SVM with:

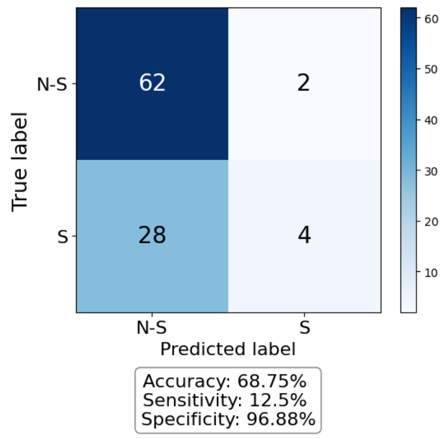
- `C = 10`
- `kernel = rbf`

Figure B.1: The performance for the best fold in 10-fold cross-validation of classifying the **full** INIT data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 83.91% for KNN and 80.64% for SVM.



(a) Performance of KNN with:

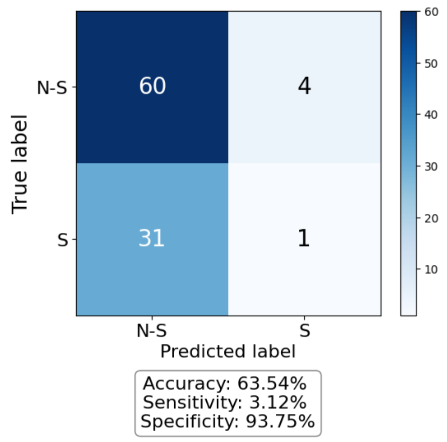
- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 2`



(b) Performance of SVM with:

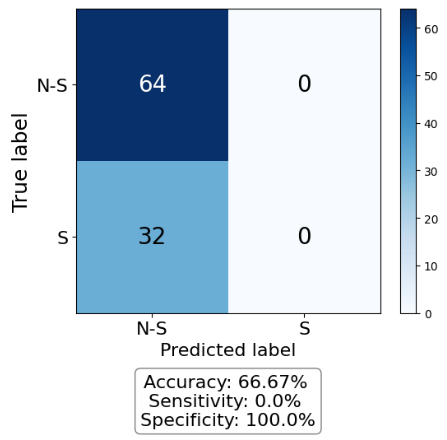
- `C = 1000`
- `kernel = rbf`

Figure B.2: The performance for the best fold in 10-fold cross-validation of classifying the **time series features** of INIT data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 78.4% for KNN and 74.54% for SVM.



(a) Performance of KNN with:

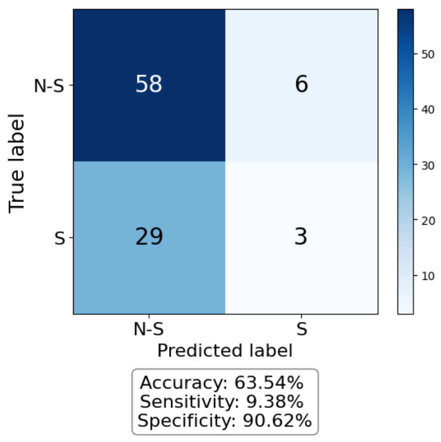
- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 1`



(b) Performance of SVM with:

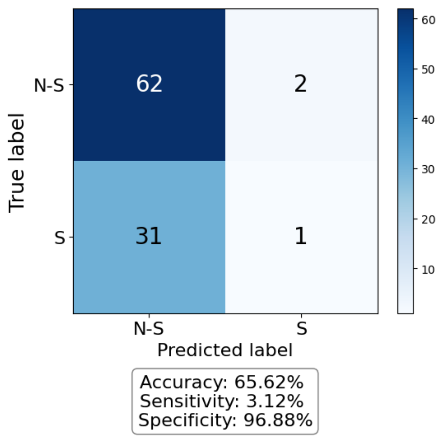
- `C = 0.01`
- `kernel = linear`

Figure B.3: The performance for the best fold in 10-fold cross-validation of classifying the **fractal features** of INIT data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 80.04% for KNN and 79.85% for SVM.



(a) Performance of KNN with:

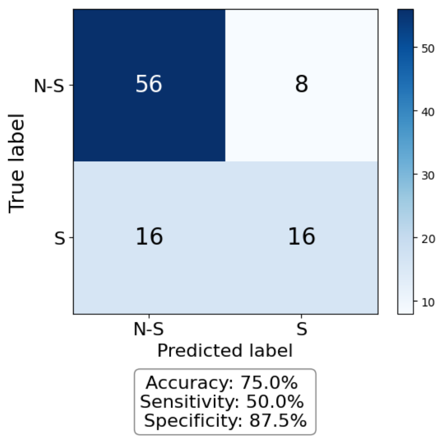
- `leaf_size = 1`
- `n_neighbors = 3`
- `p = 2`



(b) Performance of SVM with:

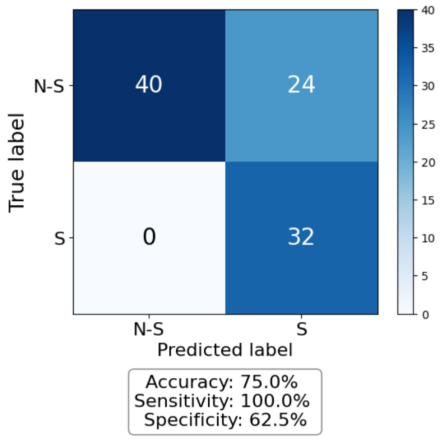
- `C = 10`
- `kernel = rbf`

Figure B.4: The performance for the best fold in 10-fold cross-validation of classifying the **entropy features** of INIT data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 76.45% for KNN and 76.81% for SVM.



(a) Performance of KNN with:

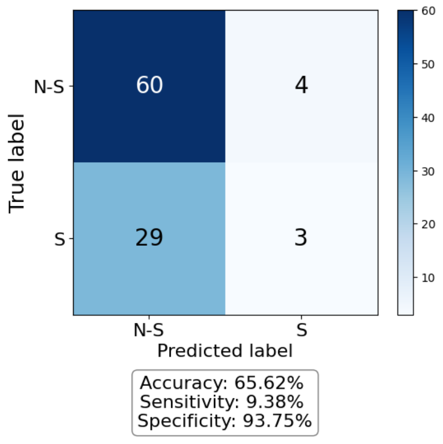
- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 2`



(b) Performance of SVM with:

- `C = 100 000`
- `kernel = linear`

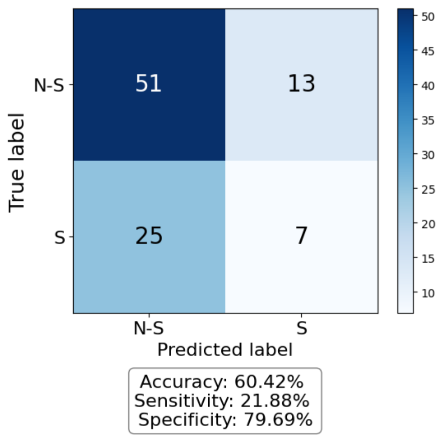
Figure B.5: The performance for the best fold in 10-fold cross-validation of classifying the **Hjorth features** of INIT data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 79.61% for KNN and 74.30% for SVM.



Accuracy: 65.62%
Sensitivity: 9.38%
Specificity: 93.75%

(a) Performance of KNN with:

- `leaf_size = 1`
- `n_neighbors = 2`
- `p = 1`



Accuracy: 60.42%
Sensitivity: 21.88%
Specificity: 79.69%

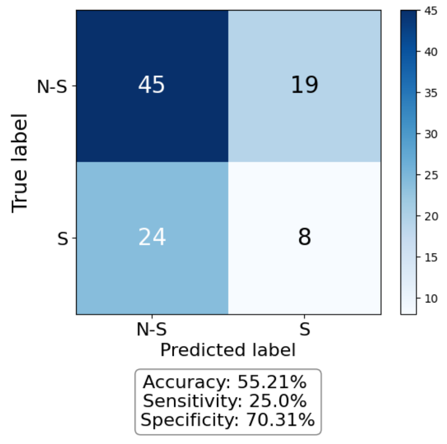
(b) Performance of SVM with:

- `C = 10`
- `kernel = rbf`

Figure B.6: The performance for the best fold in 10-fold cross-validation of classifying the **frequency band features** of INIT data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 85.12% for KNN and 77.21% for SVM.

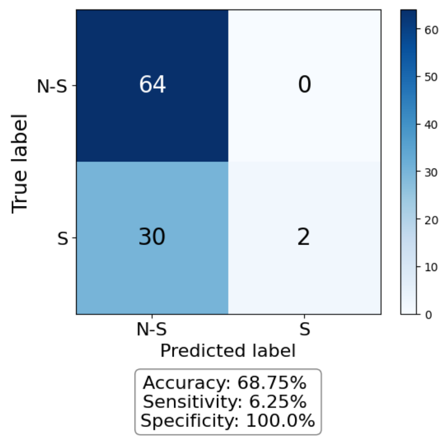
B.2 Performance of ICA Data with KNN and SVM with SS-Labels

Note: Runtime for full ICA data was >4 hours, thus it is not included in the results.



(a) Performance of KNN with:

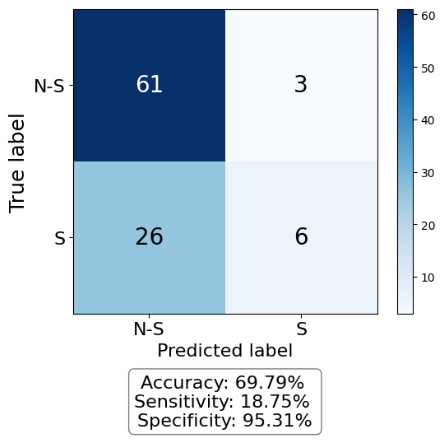
- `leaf_size = 1`
- `n_neighbors = 4`
- `p = 2`



(b) Performance of SVM with:

- `C = 100`
- `kernel = rbf`

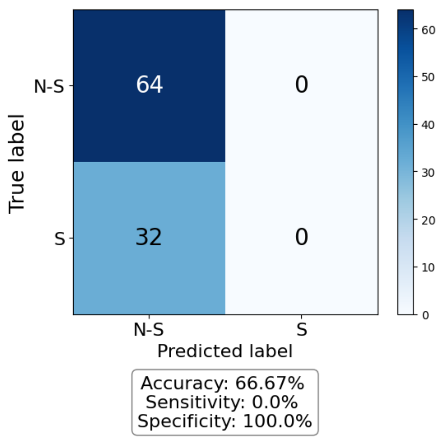
Figure B.7: The performance for the best fold in 10-fold cross-validation of classifying the **time series features** ICA data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 56.99% for KNN and 76.87% for SVM.



Accuracy: 69.79%
Sensitivity: 18.75%
Specificity: 95.31%

(a) Performance of KNN with:

- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 2`

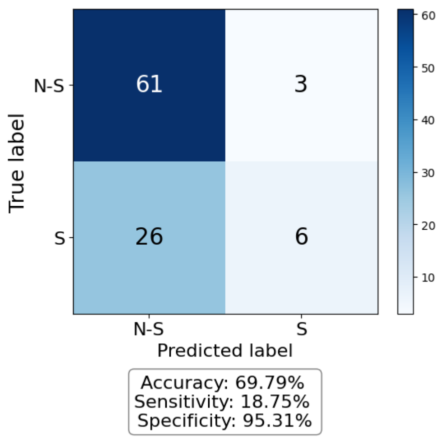


Accuracy: 66.67%
Sensitivity: 0.0%
Specificity: 100.0%

(b) Performance of SVM with:

- `C = 0.01`
- `kernel = linear`

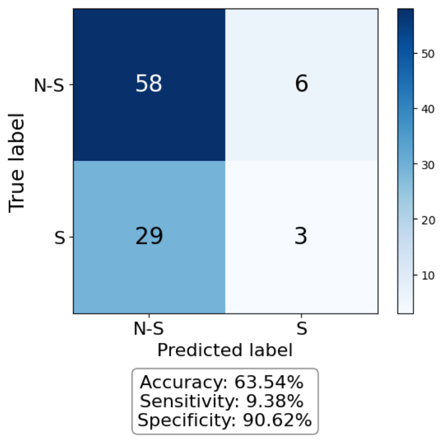
Figure B.8: The performance for the best fold in 10-fold cross-validation of classifying the **fractal features** of ICA data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 76.62% for KNN and 79.76% for SVM.



Accuracy: 69.79%
Sensitivity: 18.75%
Specificity: 95.31%

(a) Performance of KNN with:

- `leaf_size = 1`
- `n_neighbors = 2`
- `p = 2`

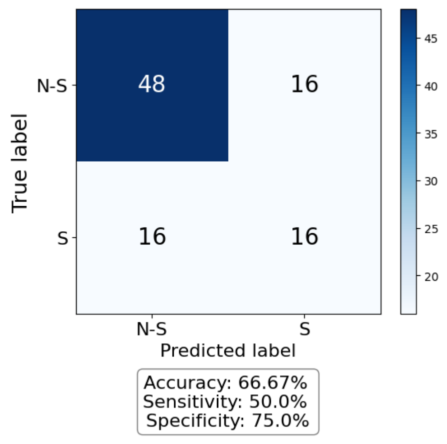


Accuracy: 63.54%
Sensitivity: 9.38%
Specificity: 90.62%

(b) Performance of SVM with:

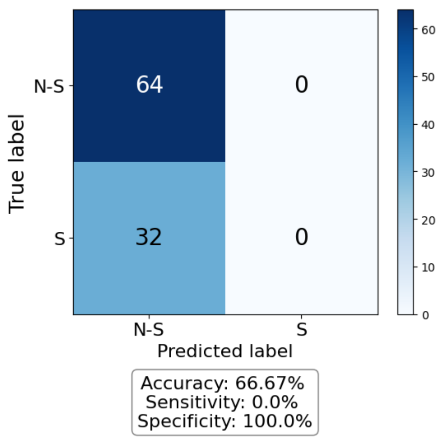
- `C = 10`
- `kernel = rbf`

Figure B.9: The performance for the best fold in 10-fold cross-validation of classifying the **entropy features** of ICA data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 79.19% for KNN and 76.26% for SVM.



(a) Performance of KNN with:

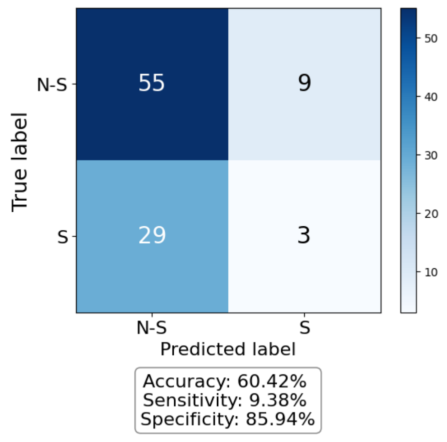
- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 1`



(b) Performance of SVM with:

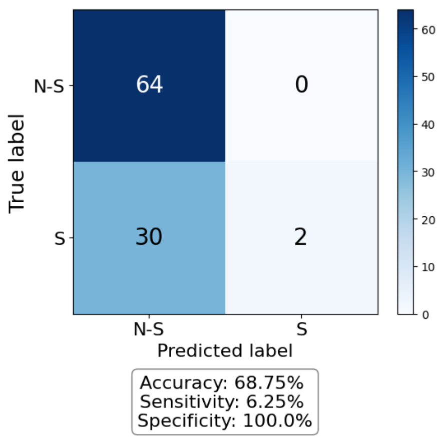
- `C = 10`
- `kernel = rbf`

Figure B.10: The performance for the best fold in 10-fold cross-validation of classifying the **Hjorth features** of ICA data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 77.38% for KNN and 71.62% for SVM.



(a) Performance of KNN with:

- `leaf_size = 1`
- `n_neighbors = 2`
- `p = 2`

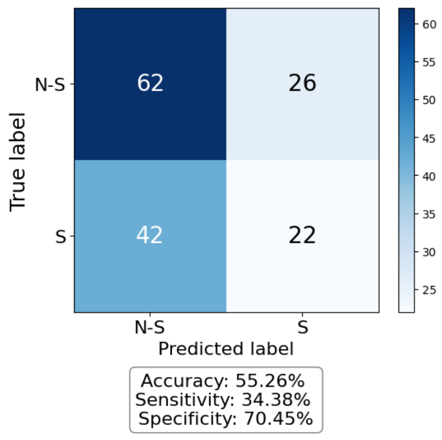


(b) Performance of SVM with:

- `C = 0.1`
- `kernel = poly`

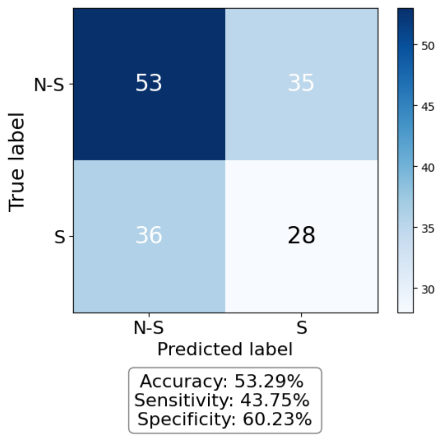
Figure B.11: The performance for the best fold in 10-fold cross-validation of classifying the **frequency band features** of ICA data with KNN and SVM and SS-labels. The overall accuracy for all combinations in the grid search was calculated to be 76.09% for KNN and 74.29% for SVM.

B.3 Performance of NEW_INIT data with KNN and SVM



(a) Performance of KNN with:

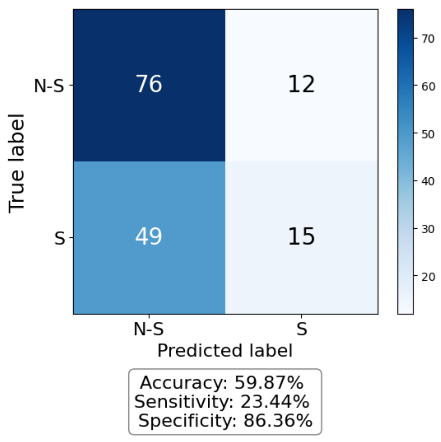
- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 2`



(b) Performance of SVM with:

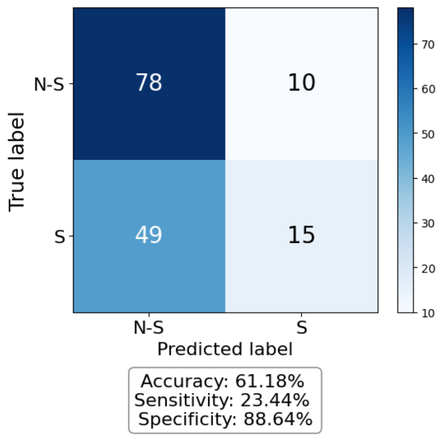
- `C = 10000`
- `kernel = rbf`

Figure B.12: The performance for the best fold in 10-fold cross-validation of classifying the **time series features** of NEW_INIT data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 64.95% for KNN and 62.35% for SVM.



(a) Performance of KNN with:

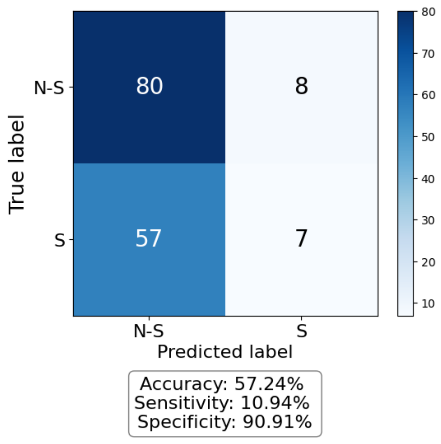
- `leaf_size = 1`
- `n_neighbors = 2`
- `p = 1`



(b) Performance of SVM with:

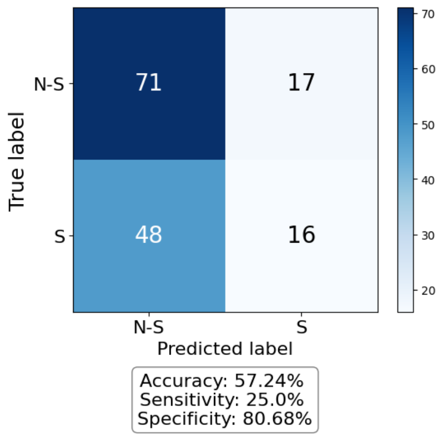
- `C = 0.001`
- `kernel = linear`

Figure B.13: The performance for the best fold in 10-fold cross-validation of classifying the **fractal features** of NEW_INIT data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 64.65% for KNN and 63.02% for SVM.



(a) Performance of KNN with:

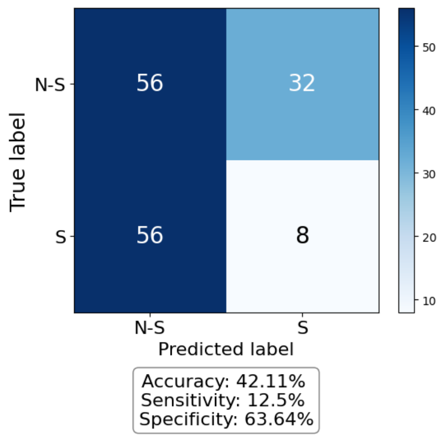
- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 2`



(b) Performance of SVM with:

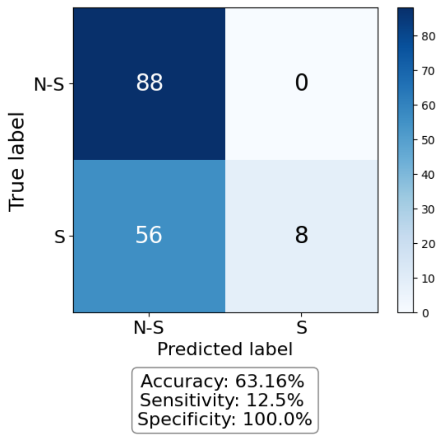
- `C = 100`
- `kernel = poly`

Figure B.14: The performance for the best fold in 10-fold cross-validation of classifying the **entropy features** of NEW_INIT data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 63.70% for KNN and 60.70% for SVM



(a) Performance of KNN with:

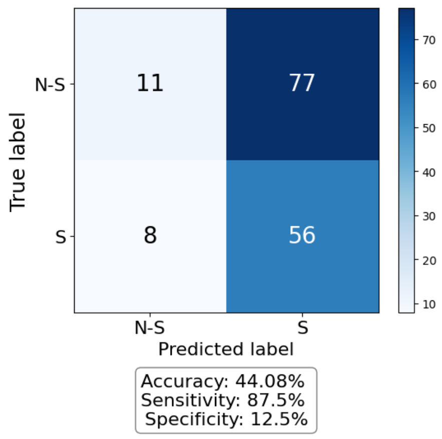
- `leaf_size = 1`
- `n_neighbors = 1`
- `p = 1`



(b) Performance of SVM with:

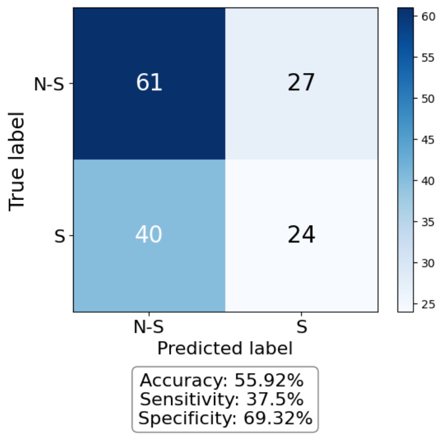
- `C = 100000`
- `kernel = poly`

Figure B.15: The performance for the best fold in 10-fold cross-validation of classifying the **hjorth features** of NEW_INIT data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 58.08% for KNN and 59.32% for SVM



(a) Performance of KNN with:

- `leaf_size = 1`
- `n_neighbors = 3`
- `p = 2`



(b) Performance of SVM with:

- `C = 10`
- `kernel = rbf`

Figure B.16: The performance for the best fold in 10-fold cross-validation of classifying the **frequency band features** of NEW_INIT data with KNN and SVM and STAI-Y-labels. The overall accuracy for all combinations in the grid search was calculated to be 43.32% for KNN and 58.23% for SVM

B.4 Performance of PSD with KNN and SVM

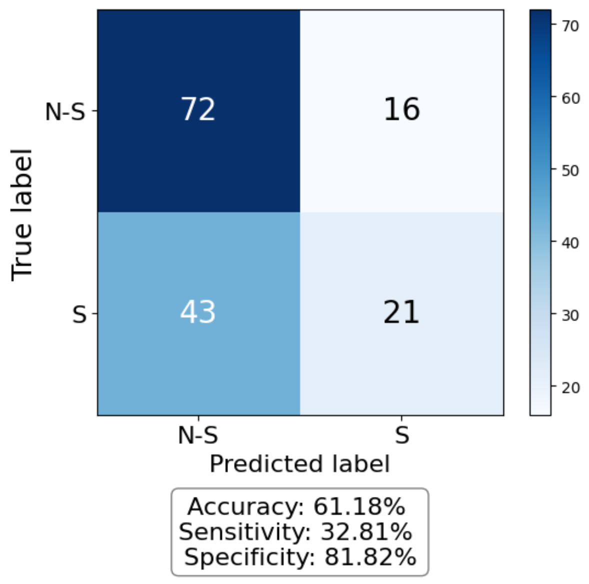


Figure B.17: The confusion matrix, accuracy, sensitivity and specificity for classifying the PSD of NEW_INIT data with KNN and STAI-Y-labels.

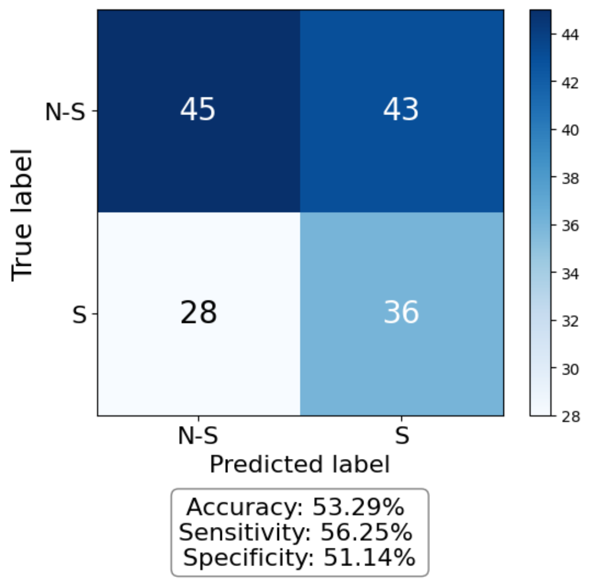


Figure B.18: The confusion matrix, accuracy, sensitivity and specificity for classifying the PSD of NEW_INIT data with SVM and STAI-Y-labels.

Appendix C

Additional Material

C.1 Consent form

Department of Engineering Cybernetics

DATA ACQUISITION CONSENT FORM

You are being invited to participate in a research study, which the Norwegian Center for Research Data (NSD) has reviewed and approved for conduction by the investigators named here. This form is designed to provide you - as a human subject - with information about this study. The investigator or his/her representative will describe this study to you and answer any of your questions. You are entitled to a copy of this form. If you have any questions or complaints about the informed consent process of this research study or your rights as a subject, please contact the PI or Co-PI [redacted]

Project Title: FlexEEG in Mental Health

Principal Investigators: Marta Molinas

Co-investigator: Andres Soler & Mohit Kumar

Thank you for agreeing to participate in this research project. This study involves research aimed at detecting the presence of psychological stress in the human body based on the analysis of EEG and PCG signals. You will participate in two separate data collection sessions. The first session will take place in the exam period of nov-dec 2022, and the second will take place after the holidays, early 2023. Before each session we will ask you to answer a self-evaluation questionnaire called 'State-Trait Anxiety Inventory'. This questionnaire will be used to determine whether you are stressed or not. During both sessions, you will be recorded twice: one five-minute period with no stressor, and one five-minute period with an Arithmetic stressor. You will be asked to rate your stress level on a scale from 1-10 after each recording. The Arithmetic stressor consists of different arithmetic statements presented on a screen. Your task will be to calculate each task in your head and click "T" on the keyboard if the statement is True, and "F" if it is False. This task is supposed to induce stress so please keep this in mind. Each session will last about 30 minutes. 10 of these minutes are for recording of EEG and PCG signals using Mentalab EEG and EkoDuo stethoscope. We will clean the areas of the scalp where the electrodes are placed with isopropyl alcohol. Electrode cap gel will be applied to the areas, but it is easily washed out with water and shampoo.

Participation in this study will take approximately 60 minutes of your time. We warn that the set-up of the EEG cap can lead to some discomfort, and the tasks you are given will (hopefully) induce some stress response. Your participation in this study is completely voluntary. Should you decide to discontinue participation or decline to answer any specific part of the study, you may do so without penalty.

Your participation in this study may help you understand the manifestations of stress on EEG signals. We are not asking you to place your name anywhere on the experimental booklet, so your participation is anonymous. None of your answers can be directly traced back to you. Should you have any further questions, please feel free to contact the study's principal investigator or co-PI, Marta Molinas and Andres Soler at the Department of Engineering Cybernetics. Her office is at Elektro D+B2 room D244, her phone number is [redacted], and her e-mail address is [redacted]

By signing below, I confirm that:

- I give my consent to participate in the research study entitled "FlexEEG in Mental Health".
- I hereby confirm that I have read the above information and have been informed about the content and purpose of the research.
- I fully understand that I may withdraw from this research project at any time without prejudice or effect on my standing with NTNU.
- I also understand that I am free to ask questions about techniques or procedures that will be undertaken.
- I give my consent for the collection and use of all data of the research "EEG and PCG in mental health" for use in research and teaching purposes.
- I give my consent to use my data for scientific purposes, its documentation and publications (including any exhibitions and further publications)
- I hereby declare that I am currently not diagnosed by a with any heart disease, or neurological disease
- I am also not on any medications affecting heart rate and/or brain wave function
- I hereby declare that I am not officially diagnosed with any mental illness

Date and place: ______________________________ and ______________________________

Participant's signature: ____

First and last name: ____

Date of Birth and current Age: ______________________________and __________________

I hereby certify that I have given an explanation to the above individual of the contemplated study and its risks and potential complications.

[illegible] 29/11/2022

______________________________ ________________

Principal Investigator's signature Date