Decoding EEG signals

Anthony Dai (4866592)
Joris van de Weg (5174821)

Delft University of Technology

TU Delft

Decoding EEG signals

by

Anthony Dai (4866592)
Joris van de Weg (5174821)

to obtain the degree of Bachelor of Science
at the Delft University of Technology,

Project duration:	April 24, 2023 – June 15, 2023	
Thesis committee:	Prof. dr. B. Hunyadi,	TU Delft, supervisor
	Prof. Dr. K. M. Dowling,	TU Delft
	Prof. Dr. D. Cavallo,	TU Delft

Cover: https://www.allerin.com/wp-blog/wp-content/uploads/2019/08/risks-of-brain-computer-interface-applications-of-brain-computer-interfaces.jpg

TU Delft

Abstract

In the context of designing a real-time brain-computer interface for playing a game using the OpenBCI Ultracortex ”Mark IV” headset, this paper focuses on the work of the decoding subgroup. The primary responsibility is to analyse EEG data retrieved from the OpenBCI headset and classify the intention of the user. Our objective is to achieve a high-accuracy classification of the EEG signals. The paper is structured into three main sections: preprocessing, feature extraction, and classification. Multiple methods for preprocessing and classification of motor execution EEG signals will be analysed, striving to contribute to the real-time implementation of the project. The results of our work provides valuable insights for future research and development in this field.

Preface

This paper is part of a Bachelor's graduation project, of which the goal is to develop a real-time BCI to play a game using EEG signals. The project consists of three groups: decoding, measurements and interface. Our group is the decoding group.

Our subgroup specifically is responsible for the decoding of EEG signals and providing an output to the interface group based on the user's intention. Numerous methods for preprocessing and classification are considered in the paper.

We would like to express our deepest gratitude to our daily supervisor, Prof. Dr. Borbála Hunyadi, for her guidance and support. We also extend our thanks to Prof. Dr. Ir. Leon Abelmann for his invaluable insights and advice throughout the project. Furthermore, we would like to thank appreciation the PhD students for their assistance. Finally, we are grateful to Ing. Martin Schumacher, who was willing to be a test subject for EEG data collection.

We hope that this project will be picked up and further developed after the completion of this thesis, as it is a fascinating subject with significant potential for future exploration and development.

Anthony Dai (4866592)
Joris van de Weg (5174821)
Delft, July 2023

Contents

1 Introduction

Brain Computer Interfaces (BCI) are a popular research topic for scientists and researchers. BCI is a system that is able to decode brain waves into a function for an external device such as a computer, exoskeleton, assistive technology etc. Electroencephalography (EEG) equipment is the most common way for recording brain signals in BCI systems, the main reasons being the relatively low cost, ease of use, and non-invasive implementation. They are mainly used for research and clinical applications now, but there are other non-biomedical applications too such as, gaming, art, transport, safety and control.

There are multiple ways to evoke brain signals, while some types are easier to extract, others are more difficult and require additional preprocessing. These so-called brain control signals are categorized into three classes: evoked signals, spontaneous signals, and hybrid signals.
This paper focuses exclusively on spontaneous signals. These signals are brain signals produced by a person with no external cues at their own will. Motor imagery (MI) is of this type and is widely used as the control signal for classification [15]. MI is imagining movements of the limbs without actually moving them.

The popularity of MI is due to its high potential in neurorehabilitation and neuroprosthetics. However, MI based BCI still has limitations and challenges due to the complex nature of EEG signals. EEG signals generally suffer from high dimensionality due to the number of channels used. EEG signals are non-Gaussian, non-stationary and non-linear. There are also a lot of different preprocessing methods that could be applied, each of which has its own advantages and disadvantages. The relationship between the neuroanatomical state of the user and the BCI's performance is still unclear [10]. The psychological state of the user especially affects the performance of MI based classification. Due to these problems, classifications can be complex and suffer from poor accuracy [13]. The biggest weakness of non-invasive BCI implementations compared to invasive implementations are the substantially poor signal-to-noise ratio. Mainly due to the electrodes being on the scalp, it does not only measure brain waves, but it also senses muscular artefacts, has a lower resolution, and is more susceptible to noise from outside the body. For these reasons, one of the challenges is creating MI based datasets that are well representative of the motor imagery that was imagined.

As a first step instead of using purely motor imagery, the decoder should be able to classify actual movements of the limbs (motor execution, ME). The goal of this paper is to develop a decoder that can accurately classify between two classes, based on the limb that is moved, to be used for real-time implementation. Motor execution has an overlap between movement and imagery of the same type and has the same spatial distributions [14]. Motor execution shows also shows higher activation levels compared to motor imagery in low frequency bands (<30Hz) [14]. So it is expected that classification with motor execution will have the same if not better results compared to motor imagery with the same model.

The main reason for using motor related brain activity as a control signal is that in the end a game has to be played that uses EEG as a controller. It will most likely involve controlling something that should move on the computer screen, such as a cursor or a character. For example, intuitively it would make more sense to move the left arm for a left-related movement in the game.

The problem with the acquired EEG signals is that they often contain a lot of information that needs to be processed and have a poor signal to noise ratio. Therefore preprocessing is crucial in analyzing EEG. Furthermore for real-time implementation, the algorithms and techniques used also need to be computationally efficient to prevent a noticeable delay and a bad experience when playing the game. However, a shorter delay means less time to analyze the data and therefore a simpler, less complex analysis model which can be detrimental to the accuracy. Therefore a lot of considerations need to be taken into account between satisfactory accuracy and a short delay in the design.

1.1. General Structure

In order to generate an output from the brain while using the BCI, several steps are involved in the processing of the EEG signals. The framework and total overview from the raw EEG signals measured with, for example, the OpenBCI to decode it to a certain label that can be used for the game is discussed here.

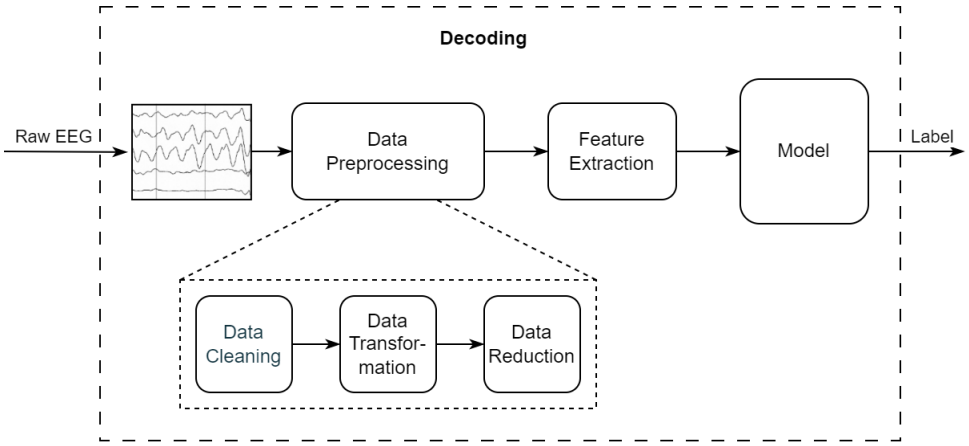


Figure 1.1: Overview of the decode framework

The general structure of the BCI from the brain to controlling a game is as follows:

1.1.1. Signal acquisition

Raw EEG signals are measured with the electrodes on the scalp at a frequency of 250Hz. It has to be ensured that the measured data has a high signal to noise ratio. The measurement group is responsible for this part of the project. The electrode placement layout used for our own recordings on the OpenBCI headset is based on the 10-20 International system. In Figure 1.2 the possible electrode placement with a top view can be seen.

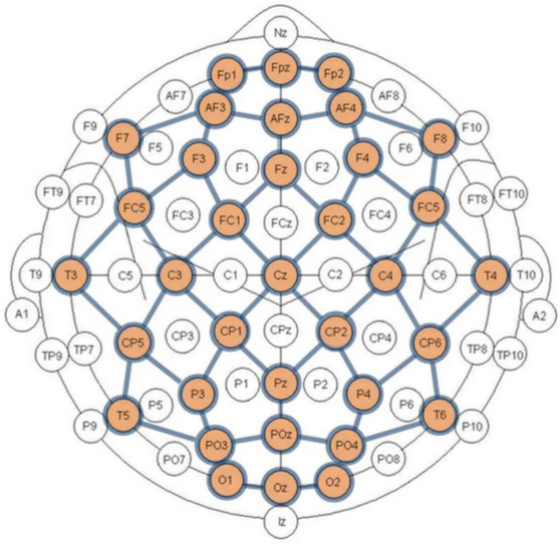


Figure 1.2: Electrode layout of OpenBCI Ultracortex Mark IV headset (In orange) [1]

It is decided by the measurement group that only for 8 positions the EEG signals are recorded. In Table 1.1 the chosen electrodes for EEG recordings can be seen.

Table 1.1: Electrodes used in the OpenBCI headset

Channel	Position
1	Fp1
2	Fp2
3	Fz
4	Cz
5	C3
6	C4
7	P3
8	P4

The channel order is chosen arbitrarily. The position name will be called when referring to a specific electrode in this paper.

1.1.2. Pre-processing

Raw EEG signals contain a lot of noise and other components which can interfere with the extraction of meaningful information. Therefore, the incoming data must be pre-processed to remove much noise as possible without removing the signal of interest. Multiple techniques will be discussed which try to achieve this.

1.1.3. Feature extraction

By only looking at the EEG signals itself it is hard to see what the user's intention is. That is why different algorithms need to be applied to extract features from the data that is discriminatory and related to the user's intention.

1.1.4. Classification

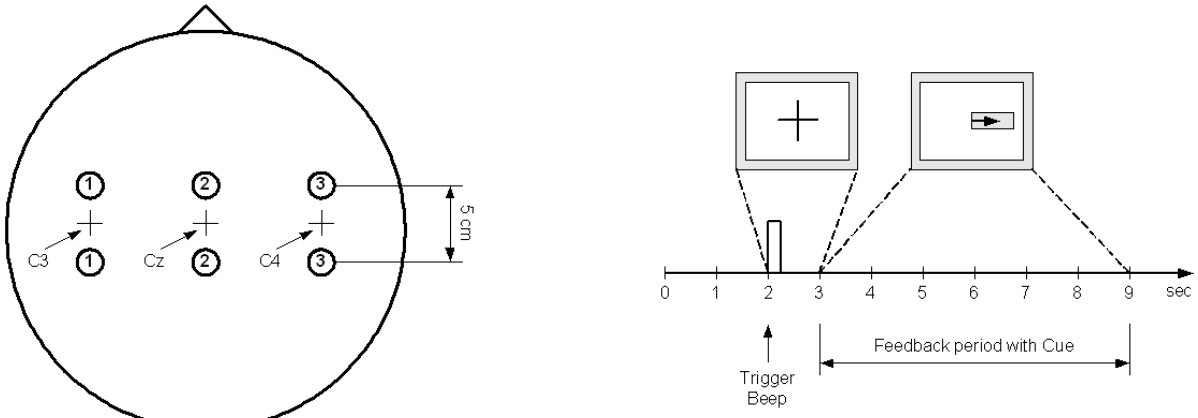
The features are then used as input for a model to decode the user's intention. This model has to be trained and validated.

[1] https://shop.openbci.com/products/ultracortex-mark-iv

1.1.5. Computer Interface
After the user's intention has been decoded, it can then be used to control something in the game.

1.2. Dataset description
Several datasets are used in this paper. One online dataset obtained from the BCI competitions: BCI competition II dataset III [2]. A blinking dataset created by Mohit Agarwal and Raghupathy Sivakumar [1]. And several datasets were measured from our own OpenBCI headset.

1.2.1. BCI competition II dataset III
To test the framework for training a model, a public dataset was selected. The dataset is provided by the Department of Medical Informatics, Institute for Biomedical Engineering at the University of Technology Graz. The dataset consists of recordings from one subject who performed a task while seated in a relaxing chair. The task involved manipulating a bar in the direction of the arrow displayed on the screen. The arrow, indicating left or right, appeared at the 3-second mark of the video. Electroencephalogram (EEG) measurements were collected from three bipolar channels, specifically C3, Cz, and C4, as illustrated in the figure 1.3. The sampling rate used was 128Hz, and the data underwent filtering between 0.5 and 30Hz. This dataset is chosen due to the task of decoding the actions left and right, which has a similar measurement process to the data recorded using the OpenBCI headset.



(a) Placement of electrodes used in the Graz Dataset

(b) The timeline of one recording containing timing beep and arrow.

Figure 1.3: Characteristics of the Graz dataset; BCI competition II dataset III

1.2.2. Online blinking dataset
The blink data are collected based on the OpenBCI headset. The dataset used is called EEG-IO and consists of voluntary single eye-blinks with external stimulation. The EEG is recorded for the frontal electrodes (Fp1, Fp2) with an OpenBCI headset for 20 subjects. Electrode gel is used to ensure contact between the electrodes and the skin. The left and right earlobes are used as references. Each subject is asked to sit in a front of computer and perform a single eye blink when a green plus appears on the computer screen. Each session that is conducted includes around 25 blinks per subject

1.2.3. Own measured data
Several datasets are created with our own OpenBCI headset, measured by the measurements group.

- Own dataset A: This dataset consists of motor execution tasks performed by one subject (middle-aged, male, healthy). Each measurement contains 10 trials (5 right and 5 left), with a 10 second rest period before the trials. At the beginning of each trial, a cross is shown on the screen and after 2 seconds an audio cue is played to attend the subject that stimulation is coming. One second later either a left or right arrow is shown on the screen for 2 seconds. The subject then clenches his left or right hand depending on the direction of the arrow shown. Afterwards, a final 5 seconds of black screen will be shown as a rest period. After that, the same trial is repeated. In total 100 trials are measured.

- Own dataset B: This dataset contains blinking. One subject (21-year-old, male, healthy) was asked to stay still and not blink for the first 10 seconds. At the 10-second mark, a sound cue is played and the subject has to blink once. This is done every other 3 seconds after the 10-second mark. The plots start from the 10-second mark. The plots are ranked from left to right on the distance between the electrode and the eyes. In total 10 trials have been run and 100 blinks with ground truths are recorded.

1.3. Evaluation metrics

In order to interpret the results a well-established set of evaluation metrics need to be defined. Different evaluations may be used for different types of implementation.

Table 1.2: Confusion Matrix

		Predicted Class	
		Class 1	Class 2
True Class	Class 1	True Negative (TN)	False Positive (FP)
	Class 2	False Negative (FN)	True Positive (TP)

The computed models are tested by making predictions on the test data, where these results can be viewed as shown in Table 1.2. The negative class, class 1, in the case of the Graz dataset, is the action left and the positive class, class 2, is the action right. To assess and evaluate the results and the effectiveness of these models, multiple performance measures are used, including model accuracy, balanced accuracy, F1-score, and Kappa. Model accuracy provides an overview of the overall performance in terms of predictions and is simply calculated by dividing the correct predictions by the total number of predictions. Looking at Table 1.2 the accuracy is defined as shown in Equation 1.1. The balanced accuracy is similar to accuracy but can be very useful for unbalanced datasets. Although the classes are balanced in the datasets that are used in this project, the split of training and testing the data is random. The true positive rate and the true negative rate are combined and averaged. This balanced accuracy provides, calculated as shown in Equation 1.2 a more reliable measure of performance.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{1.1}$$

$$\text{Balanced Accuracy} = \left(\frac{TP}{TP + FP} + \frac{TN}{TN + FN} \right) \cdot 0.5 \tag{1.2}$$

The third measure of performance is the f1-score which is calculated using the precision and recall of the model. Precision is the rate of correctly predicted positive instances, true positives, out of the total instances predicted as positive, true positives plus false positives, shown in Equation 1.3. Where recall is the ratio between correctly predicted positive instances, true positives, and all actual positive instances, true positives plus false negatives, shown in Equation 1.4. Precision is important to optimize if false positives should be avoided. Recall, on the other hand, is important to optimize when the false negatives have to be avoided. The f1-score combines these measures to provide a balanced estimation of the model's performance that takes into account both false positives and false negatives. For this project, the cost of the FNs and FPs can be considered the same. Therefore the models are not optimized for this measure of performance, but it can be useful to understand the division in the predictions.

$$\text{Precision} = \frac{TP}{TP + FN} \tag{1.3}$$

$$\text{Recall} = \frac{TP}{TP + FP} \tag{1.4}$$

$$\text{F1-score} = \frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}} \tag{1.5}$$

2

Program of Requirements

In this chapter, the requirements for the design are stated, providing a detailed outline of the essential criteria and specifications that should be met. The following requirements serve as a foundation for the design phase, helping to define the system's functionality, performance objectives and operational constraints.

Must:

- Must use the Graz dataset subsection 1.2.1, to test and evaluate the framework of the decoding before applying it to the OpenBCI dataset. The classification model, which distinguishes between two classes, must have an accuracy of at least 80 %.
- Must make a classification model to distinguish between 2 classes, with an accuracy of at least 60 percent based on measurements from the OpenBCI dataset.
- Multiple preprocessing must be considered and evaluated for removing potential artefacts and noise.
  - Must be able to detect and remove eye blink artefacts from EEG.
- The computation must have a maximum latency of 500 ms. In other words, the preprocessing and feature extraction combined computation time must not exceed this value.

Should:

- The project should assess different classification algorithms and compare their performances on the dataset.
- Should make a model to distinguish between 3 classes. Where the third class represent no action.
- Make a model to distinguish eye blinks in EEG, with an accuracy of at least 90% percent based on own measurements.
- The product should be ready for integration with the other subgroups. Therefore it should also work for a continuous input signal and must provide a continuous output signal

Could:

- Could make a model to distinguish between 5 classes, implementing the cursor characteristics.
- The product could have high compatibility, meaning the model should give an accuracy of 70 percent for every person for own recorded data.

3

Preprocessing

Raw EEG signal readings are very sensitive and are often contaminated with artefacts. EEG is affected by artefacts both from biological and environmental sources. Biological artefacts are sources from within the body and environmental from outside. Some biological artefacts are: eye blinking, heartbeats, and muscular movement and some environmental artefacts are: the 50Hz power line interference, movement of the electrodes, EM waves from other devices and so on.

Since EEG signals are already difficult to analyze due to their non-linear and non-stationary behaviour, as much noise as possible should be removed in order to obtain clean EEG signals. This is especially true if the signal of interest is weak like the sensorimotor rhythms (SMR), which are oscillatory events in EEG signals from the motor cortex area of the brain [16], since other sources such as the visual cortex or muscle artefacts produce strong signals in the same frequency band. Another reason why it is needed is that the BCI we are designing should also work in real time. This means that the analysis is based on single trials. The noise during the usage of the BCI may vary with each trial. So preprocessing is needed to standardize it more and increase signal quality. However, with noise removal on EEG signals, it is difficult to distinguish if the signal of interest has not also been removed. The way to justify using a certain preprocessing technique in this chapter is to see whether or not it improves on the classifier accuracy after adding it. Another requirement is that the preprocessing in its entirety should not take too long since in the end the classifier should work in real-time.

This creates an additional consideration in the design where the combination of the preprocessing techniques should not be too large. Again, the total delay of the entire classifier as stated in the programme of requirements 2, should be smaller than 500ms. The reason for a delay of 500ms specifically is because all the groups decided that a game can still be built around this delay and be enjoyable. The main evaluation metric for using a certain preprocessing method is mainly the accuracy that is gained in comparison to without using it. This chapter discusses what preprocessing methods are chosen initially, without evaluating it. The results and evaluation of the preprocessing methods will be discussed in the results section 6.

The preprocessing methods that will be considered are:

- Bandpass filter
- Eye blink detection with thresholding
- Eye blink detection with random forest (RF) classification
- Eye blink removal with Independent Component Analysis (ICA)
- Common Spatial Patterns (CSP)

These preprocessing methods are chosen mainly due to their popularity in EEG preprocessing. Among these techniques, 22% use independent component analysis (ICA), 14% use common spatial patterns (CSP)[15].

3.1. Bandpass filter

Since motor imagery is the control signal for our feature extraction and classification, unwanted signals present in the EEG should be removed. One of the signals related to motor imagery is sensorimotor rhythm (SMR). Brain activity recorded via EEG is typically classified into different types depending on the predominant frequency content, f, which can be seen in Table 3.1.

Table 3.1: Frequency range for each type of brain activity

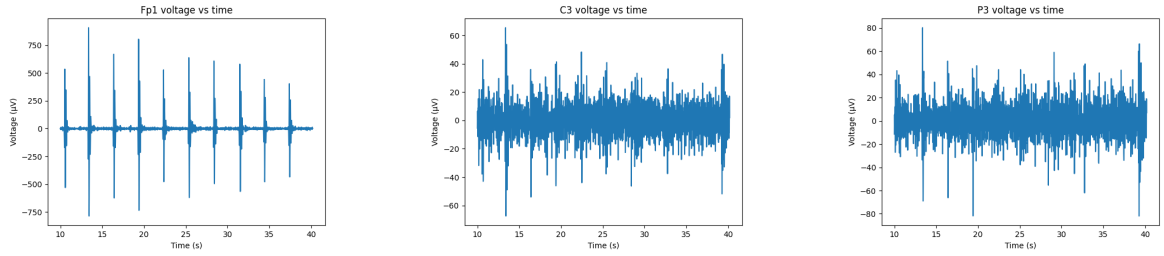
Type	Frequency range
Delta activity	f <4Hz
Theta activity	4Hz <f <7Hz
Alpha activity	7Hz <f <12Hz
Beta activity	12Hz <f <30Hz
Gamma activity	f >30Hz

In the literature, alpha activity recorded from the sensorimotor region is known as mu activity. Changes in the mu and beta activity are used to classify the type of motor imagery task [16]. Gamma activity could also be used to classify MI tasks, however, gamma signals do not reach the scalp with high enough resolution [16]. So gamma activity will not be considered in our design. The frequency range of interest for MI is mainly in the alpha and beta activities.

A 4th-order butterworth bandpass filter will be used in our design to include these frequencies and filter out the unnecessary frequencies. The bandpass filter is between [7Hz 30Hz]. In chapter 6 how we came to this interval is explained.

3.2. Eye blink artefacts

The most common biological artefacts in EEG signals are blinking artefacts. The EEG is measured on the scalp with the electrodes. So blinking can be seen in the EEG signals as very high amplitudes relative to the normal EEG signals because they show propagation over the anterior scalp regions. This interference overlaps with the motor imagery frequency range of interest [5]. These amplitude spikes should be the most noticeable in the Fp1 and Fp2 electrodes, based on the International 10–20 system electrode placement, since they are placed nearest in distance to the eyes. After a test with the OpenBCI headset with blinking where the voltage versus time is plotted, this statement is confirmed. Figure 3.1 shows a comparison of several electrodes during a segment in an EEG recording with blinking where the exact time stamps are known when the subject has blinked, ranked from left to right on its distance to the eyes.



(a) EEG of Fp1 **(b)** EEG of C3 **(c)** EEG of P3

Figure 3.1: Own dataset B 1.2.3

From Figure 3.1 it can be seen that on Fp1, the amplitudes due to blinking are orders of magnitudes higher than the segments with no blinking. In C3 and P3, the blinking artefacts can still be distinguished but are now significantly lower in amplitude compared to Fp1. From these plots, it can be concluded that blinking affects every electrode and that it gradually gets weaker in amplitude the farther the electrode is placed from the eyes. It affects C3 and C4 which are crucial for motor imagery/execution classification. So we should expect an increase in classification accuracy after trying to remove these blinking

artefacts, since these they are not related to motor imagery/execution.

While some biological artefacts like muscular artefacts can be minimized by training the user to relax and to avoid facial expressions, it is unrealistic to expect the user to stop blinking every time when imagining for classification. Therefore, removing eye blink artefacts should be beneficial for the accuracy of the classifier since blinking will occur during usage. In order to remove eye blinks, it has to be detected in the first place. Therefore eye blink artefact removal is done in two steps. First, an algorithm to detect the eye blink needs to be implemented. After the identification of eye blinks in an EEG segment, another algorithm is implemented for the removal of the artefact. Another reason to have eye blink detection as a feature is that it can potentially be used as an additional control signal if needed. A top-level overview of the blinking removal model can be seen at 3.2.

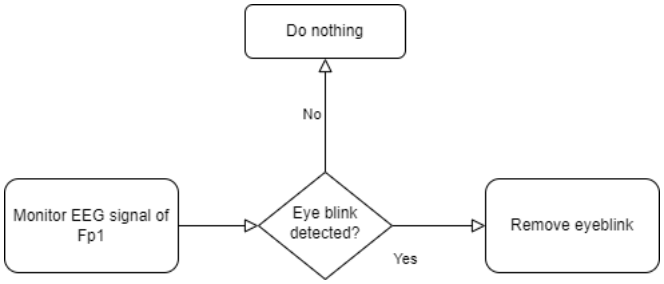


Figure 3.2: Block diagram of blinking removal model

3.3. Eye blink detection

In recent years there are emerging works that try to implement real-time EEG applications. However, these works have a large computational waste and have an oversimplified blink modelling problem [20]. A high detection accuracy can be achieved with a complex blinking detection model, but that will have as cost a higher delay. On the other hand, a simpler model can be used but then the detection accuracy will be lower.

Two methods for detecting eye blinks will be considered: thresholding and random forest classifier.

- Detection with thresholding will serve as a basic way of detecting eye blinks. Depending on the results we can justify if a more complex method has to be implemented.
- Zhang et al. [20] proposes a short-windowed and random forest-based method for real-time blink detection, which they call RT-Blink. It balances the computation complexity while also being computationally efficient enough to work in real time. The algorithm uses a sliding window and a random forest-based blink detection method which balances the computation complexity and processing speed.

3.3.1. Eye blink detection with thresholding

A very simple model but computationally efficient method for eye blink detection is to use thresholding. Thresholding is done by calculating the mean of an EEG segment and then setting the threshold to $n * SD + \mu$, where n can be any positive number, and SD is the standard deviation and μ is the mean. If any value in the EEG segment exceeds this value then an eye blink is detected.

3.3.2. Eye blink detection with random forest

RF is chosen as one possibility for eye blink detection due to its high speed and accuracy. RF eliminates overfitting and can achieve generalization for small training samples.

The random forest classifier is an ensemble of decision trees. A decision tree is a flowchart-like structure where each internal node represents a feature or attribute, each branch represents a decision rule, and each leaf node represents the outcome or class label.

The random forest classifier combines the predictions of multiple decision trees to make a final prediction. The random forest algorithm begins by randomly selecting a subset of the training data (with replacement) to build each decision tree. When selecting a subset of the training data with replacement, it means that during the random sampling process, each data point is chosen independently, and it has the possibility of being selected multiple times or not at all. In other words, for each decision tree in the random forest classifier, the same data point may appear multiple times in the bootstrap sample, while other data points may be excluded.
In classification tasks, each tree votes for a class label, and the class with the majority of votes is selected as the final prediction. In Figure 3.3 the general structure of the random forest classifier is visualized.

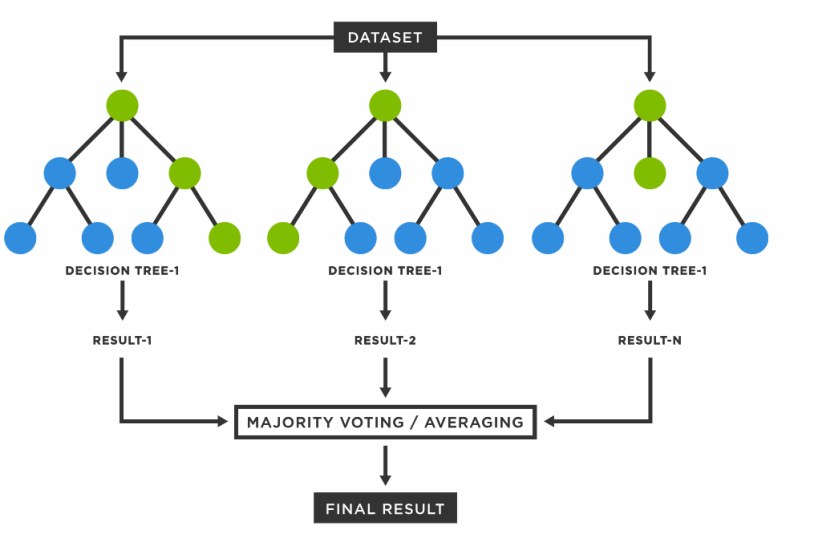


Figure 3.3: General structure of random forest classifier. [1]

3.3.3. Features for Random forest classifier

From Figure 3.1 it can be seen that Fp1 has the most distinguished amplitudes compared to the other channels so this electrode is used as a reference to get the features for blinking detection. Three features are chosen for RF: standard deviation, range of amplitude, and range of grade.
1) SD: Standard deviation is an indicator of how much a signal deviates from its mean and is calculated as the square root of the variance.

$$SD = \sqrt{\frac{\Sigma_i^N |X_i - \mu|^2}{N-1}} \tag{3.1}$$

Where N is the length of the EEG data X and μ is the mean of the EEG data. The standard deviation of an EEG segment with blinking would be expected to be higher than one with no blinking since the amplitudes of blinking as mentioned before are orders of magnitudes higher. The range of amplitude and range of grade tell the outline of the blink itself.
2) RA: The range of amplitude is defined as the difference in amplitude between the maximum and minimum values.

$$RA = D_{max} - D_{min} \tag{3.2}$$

Where D_{max} and D_{min} is the maximum and minimum value respectively. 3) RG: The rate of grade indicates the steepness of the data, which is calculated as:

$$RG = \frac{D_{max} - D_{min}}{i_{max} - i_{min}} \tag{3.3}$$

Where D_{max} and D_{min} is again the maximum and minimum value, and i_{max} and i_{min} are the index positions of these values respectively.

[1] https://www.tibco.com/reference-center/what-is-a-random-forest

The detection of eye blinks follows the diagram in Figure 3.4.

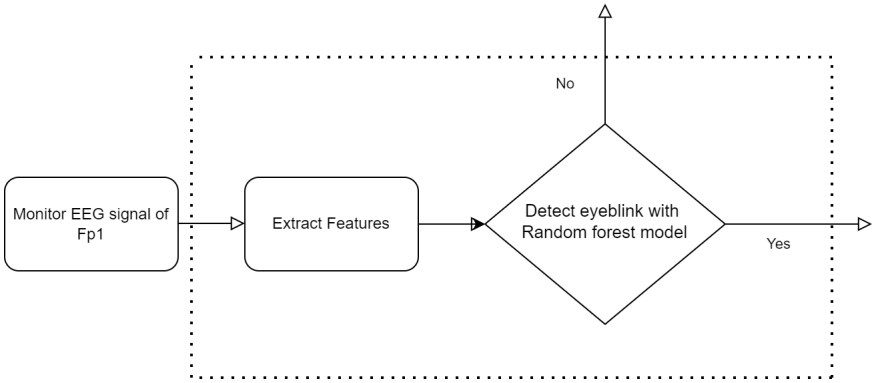


Figure 3.4: Detection of eye blink with random forest

3.4. Eye blink removal

After detecting the eye blinks, its artefacts have to be removed. In this design, independent component analysis (ICA) is considered for eye blink removal. This is chosen due to their success in eye blink removal in other papers [5]. Independent Component Analysis is a method used to estimate independent signal sources from a mixed signal using a multichannel signal. ICA is mainly used for audio blind source separation. For ICA to work best the signal should consist of signal sources that are independent in relation to each other. Since the interference due to blinking does not come from the brain itself, it should be independent of the EEG signals of interest. Therefore ICA is a great tool to remove it. ICA algorithms follow one of two general principles: maximization of non-Gaussianity and minimization of mutual information [19].

All ICA algorithms begin by whitening the data to eliminate any existing correlation. In the case of EEG signals, which consist of a mixture of brain activity and various noise components, whitening ensures that the variance is equal on both axes, and there is no correlation in the projection of the data on any axis. Following the whitening phase, the algorithm involves rotating the resulting axis of the matrix to minimize the Gaussianity of the projection on all axes. The complete transformation from the original space is represented by the weight matrix.

$$S = W * X \tag{3.4}$$

The equation describes the transformation, where X is the data in the original space, W is the weight matrix, and S represents the different independent sources. The goal of the ICA algorithm is to determine the weight matrix W that decomposes the EEG signals into independent components (ICs). There are several variants on ICA. In this paper, the FastICA variant is chosen since it is a computationally efficient algorithm to perform ICA[19].

How the blink is removed with ICA is by first extracting the independent components (ICs) from all the channels. Then the average cross correlation is calculated between each IC squared and the Fp1 and Fp2 electrodes time series squared. If the average cross correlation exceeds a certain threshold then the respective IC will be zeroed out, otherwise nothing is done. After each IC is checked, reverse ICA is performed by applying W^T to S to obtain X with eye blinks removed. This process is described in a block diagram in Figure 3.5.

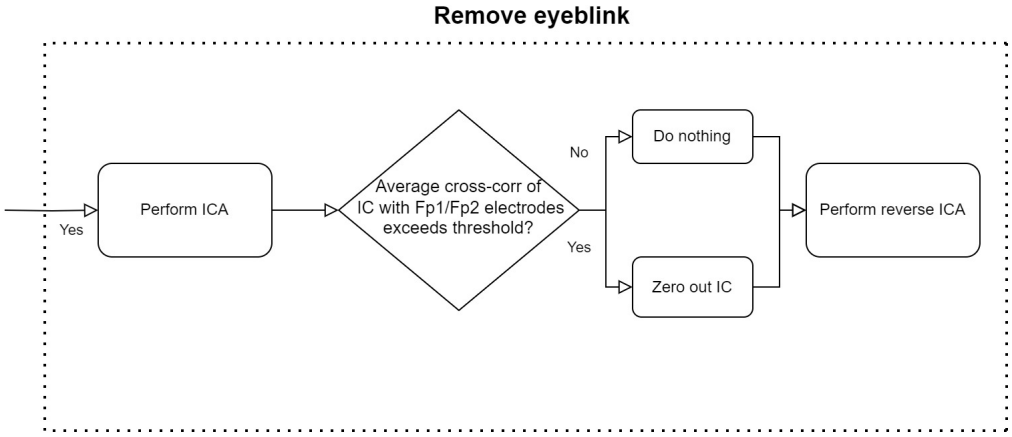


Figure 3.5: Remove eye blink block worked out in detail

3.5. Common Spatial Patterns

Common Spatial Patterns (CSP) is a commonly used spatial filter for transforming EEG signals. It is mostly used to extract frequency band variances as features by spatially transforming a multichannel EEG signal. The transformed signal will have a maximum variance for one class and a minimum for the other[3].

Suppose $X \in \mathbb{R}^{NxT}$, where X is your EEG signal in matrix form with N channels and T time samples. X is also assumed to be centered and scaled. CSP transforms this matrix into: $X_{csp}(t) = W^T X(t)$), where $W \in \mathbb{R}^{NxN}$ is a matrix of which its column vectors is composed of spatial filters. Each row X_{csp} is then one CSP component in the filtered domain. The spatial filters are chosen such that for each CSP component, the variance should be maximal for one class and minimal for the other. So as a feature, the variance is taken for each CSP component and used for model training. To compute W the following simultaneous diagonalization of two covariance matrices are solved:

$$\begin{aligned} &W^T\Sigma^+W = \Lambda^+, \\ &W^T\Sigma^-W = \Lambda^-, \text{with } \Lambda^c \text{ diagonal} \end{aligned} \tag{3.5}$$

Where Σ^c is the covariance matrix of class c. And Λ^c is the diagonal matrix of class c. The scaling of W is fixed with the constraint:

$$\Lambda^+ + \Lambda^- = I \tag{3.6}$$

The simultaneous diagonalization is solved by finding the solution to the following general eigenvalue problem:

$$\Sigma^+ w = \lambda \Sigma^- w \tag{3.7}$$

Then Eq. 3.5 is satisfied with the generalized eigenvectors $w_j (j = 1, ..., N)$ obtained from Eq. 3.7. Notice from Eq. 3.5 and Eq. 3.7 that by left multiplying Eq. 3.7 with w^T, the equation can be rewritten to $\lambda = \frac{\lambda_j^+}{\lambda_j^-}$. So λ in Eq. 3.7 is not the same as in Eq. 3.5. The numerator and denominator is one entry in the diagonal matrix in Eq. 3.5 corresponding to its class. λ_j^c is the variance in class c due to spatial filter w_j. Due to the constraint in $\lambda_j^+ - \lambda_j^- = 1$, a large value λ_j^+ corresponds to a small value in λ_j^- and vice versa. So in order to discriminate the best between the two classes, the generalized eigenvector from Eq. 3.7 should be chosen such that its corresponding eigenvalue λ is as large or as small as possible.

Finally after transforming X to X_{csp} the log variance of each row in is taken as feature. It is calculated as:

$$\log(W^T X X^T W) \tag{3.8}$$

4

Feature Extraction

This chapter will illustrate and elaborate on the framework for feature extraction methods. Various features can be used to represent the properties of the EEG signal, enabling the extraction of relevant information. The effectiveness and the elaboration of the chosen features will be discussed in the following sections.

4.1. Segmentation

In the scope of this project, the features are computed for different sizes of the EEG signal in order to find the most ideal sample window. The ideal sample window should be a balance between classification accuracy and time accuracy. Before the features are extracted from the time window, the window is multiplied by a function. The rectangular, Hamming and Hanning functions were considered and their time domain representation is shown in Figure 4.1. These functions are used to mitigate the spectral leakage that can occur when a portion or segment of data after fragmentation is transformed from the time domain to the frequency domain. The Hanning and Hamming functions decrease the influence of boundary data and therefore change the signal to a more periodic one. [17] An often-used overlapping percentage of 50% was applied for the segmentation of the dataset.[6]

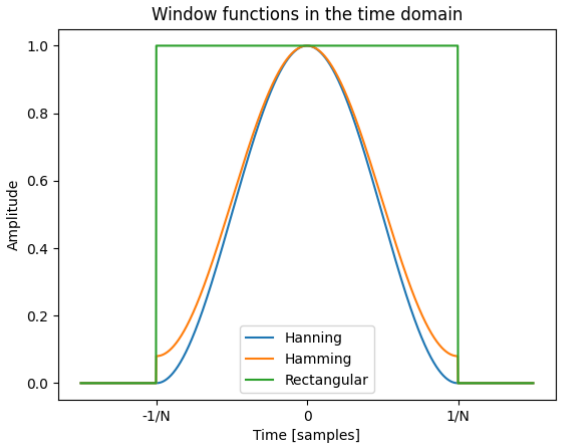


Figure 4.1: Hanning, Hamming and the rectangular window. N is the window size in samples.

Figure 4.2 displays a recording of the Graz dataset. As described in chapter 1, the first three seconds of the recording contain no action of the user. Therefore the first three seconds are removed to classify the signal for 2 classes, left and right. Consequently, the remaining portion will be subjected to fragmentation and then feature extraction. The figure shows the fragmentation achieved using a time window of 1 second and a 50% overlap, indicated by the broken lines.

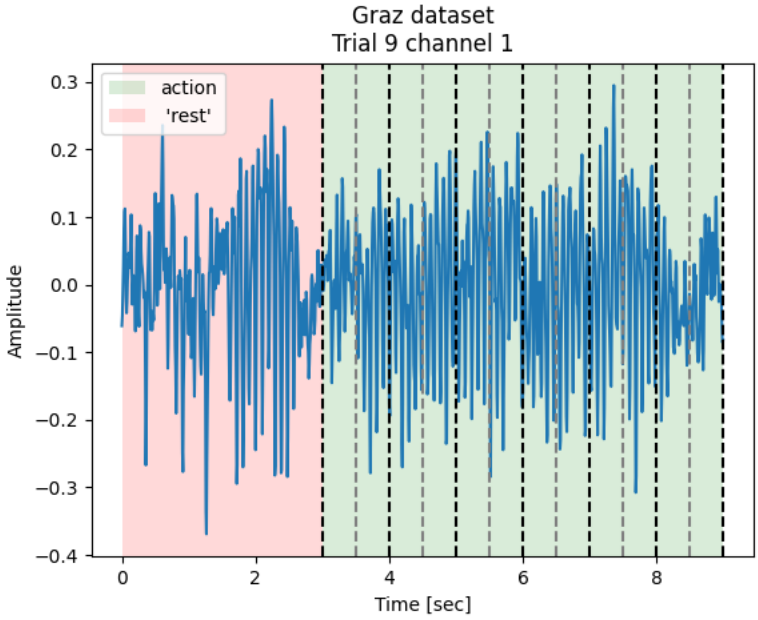


Figure 4.2: Example

4.2. Features

Time domain features represent the temporal characteristics of EEG signals by analyzing the time series of the EEG data. When examining the time domain signal, the first features used are statistical features, mean, median, variance, skewness, and kurtosis. The mean is calculated over the absolute values of the signal, which will represent the average magnitude of the signal. The variance quantifies the amount of variation in the signal and the median is the middle value of the data.

$$\text{Mean} = \frac{1}{N-1}\sum_{i=1}^{N}\text{abs}(x[i]) = \overline{x} \tag{4.1}$$

$$\text{Variance} = \frac{1}{N-1}\sum_{i=1}^{N}(x[i]-\overline{x})^2 = \sigma^2 \tag{4.2}$$

Skewness provides insights into the asymmetry of the data distribution, indicating whether it is skewed to the left or right. Kurtosis can be used as a measure of the distribution's tail behaviour, indicating the degree to which it deviates from a Gaussian distribution. A non-statistical feature that can have importance in classifying the signal is the energy. It is computed by squaring the whole signal and thus can emphasize amplitude dynamics that vary from the usual amplitude range. [4]

$$\text{Skewness} = \frac{1}{N-1}\sum_{i=1}^{N}\frac{(x[i]-\overline{x})^3}{\sigma^3} \tag{4.3}$$

$$\text{Kurtosis} = \frac{1}{N-1}\sum_{i=1}^{N}\frac{(x[i]-\overline{x})^4}{\sigma^4} \tag{4.4}$$

$$\text{Energy} = \sum_{i=1}^{N}(x[i]^2) \tag{4.5}$$

In addition to the statistical features, more advanced techniques are used for extracting information from EEG signals. One such measure is Shannon entropy, which provides a measure of the signal's complexity and unpredictability. It ranges between 0 and 1, where the higher the value the higher the disorder and unpredictability. This feature can add value for capturing complex patterns and variations

within the data. [4] Equation 4.6 shows the formula to calculate the Shannon entropy, where p_i is the probability of an occurrence of each sample.

$$\text{Entropy} = -\sum_i p_i \log(p_i) \tag{4.6}$$

Line length, sometimes referred to as the curve length, is the total vertical length of the signal. It has been suggested as a relevant feature for tasks such as seizure detection, as it captures the characteristic abrupt changes and fluctuations associated with epileptic events. Although the effect of motor control and imagery does not cause major changes, the line length can still be used to determine subtle changes. [7]

$$\text{Line Length} = \sum_{i=1}^{N-1} |x[i] - x[i-1]| \tag{4.7}$$

Another feature which can be used for the classification is the nonlinear energy (NE). It extends on the concept of energy, mentioned earlier. To generate an oscillating signal it can be assumed that energy is the squared product of the signal's amplitude and the signal's frequency. A high NE can thus represent high-frequency components and amplitudes there is a high. This feature which takes the oscillation into account is shown in Equation 4.8. [11]

$$\text{Non-Linear Energy} = \sum_{i-1}^{N-2} (x^2[i] - x[i+1]x[i-1]) \tag{4.8}$$

Changes in the EEG signal due to motor imagery or control can be observed both in the power frequency spectrum and the time domain. Features derived from these domains, such as the relative power spectral density (PSD) of the theta, alpha, and beta frequency bands, are often chosen for analysis [18]. During motor imagery, event-related desynchronization (ERD) typically occurs in the alpha band (7-12 Hz), contrary to the notion of increased activity. Other frequencies can also reveal valuable information. To calculate these features, the power spectral density for all frequencies in the band is added up together and divided by the total power of all the signals. This division ensures compatibility across recordings with different characteristics, enabling them to be compared during classification. It's also worth noting that other methods for feature extraction from EEG signals, such as time-frequency analysis, can provide more information but will not be used in this project.

Statistical features can also be computed for the power spectral density of EEG signals. The first feature used is the weighted mean frequency, WMeanF, also known as the mean frequency, which can give a good measure of the average frequency of the EEG signal. It computes the mean of the frequency distribution, using the normalized PSD as the weight for each frequency, S_n. The weighted variance of the frequency or the standard deviation of the frequency squared, the WVarF, assesses the spread or variability of the frequency components around the mean frequency. A higher variance of frequency can indicate a larger range and greater variability in the EEG signal. The mode frequency, or the frequency with the highest power in the PSD, is also used as a feature.

$$\text{WMeanF} = \sum_k S_n[k] \cdot f[k] \tag{4.9}$$

$$\text{WVarF} = \sum_k S_n[k](f[k] - \text{WMeanF})^2 \tag{4.10}$$

Using the found mean frequency and standard deviation of the frequency, the skewness and kurtosis can be calculated according to Equation 4.11 and Equation 4.12. These measures could provide insights into the shape of the frequency distribution. The last feature applied for the classification is the spectral entropy. The spectral entropy can be calculated using the formula shown in Equation 4.6 where x is in this case the normalized PSD.

$$\text{WSkF} = \sum_k^N \frac{(S_n[k] - \text{WMeanF})^3}{(N-1) * \text{WVarF}^{1.5}} \tag{4.11}$$

$$\text{WKF} = \sum_k^N \frac{(S_n[k] - \text{WMeanF})^4}{(N-1) * \text{WVarF}^2} \tag{4.12}$$

5 Classification

In this chapter, the classification process will be discussed. This includes the distribution of the test and training data and the diverse classifiers to compare the different methods.

The classification is the last step of the decoding of the EEG signals. For classifying the signals, a machine learning algorithm or model is used, referred to as a classifier. This classifier is trained where it analyzes patterns and relationships, to assign and predict a label for every fragment. To train and test the effectiveness of the model, features, computed for all the recordings/fragments of recordings, have to be divided into a train and test set. The most common method is K-fold cross-validation, which can also be used as Leave One Subject Out (LOSO). K-fold cross-validation splits the fragments, where for every fragment features were computed, in k number of train and test groups. Therefore, the model is trained and then tested k times with a different division of the data. A visualization is given in Figure 5.1. For LOSO this k value is the number of fragments or samples available, and therefore all but one fragment is used for training the model, and one is used for testing. The division of the dataset is only done with respect to the recordings and therefore not on all different segments generated. Otherwise, the model can be trained with data similar to the test data.

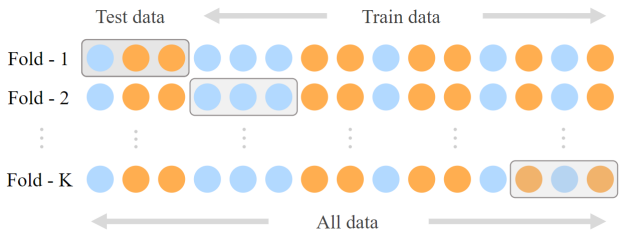


Figure 5.1: K-fold cross-validation for the total data. *Picture from:* `https://www.philschmid.de/k-fold-as-cross-validation-with-a-bert-text-classification-example`

For this project, 10-fold cross-validation is used. This will result in 10 different sets of training and test data with a corresponding ratio of 90% and 10%. This methodology is employed to evaluate the model. Furthermore, on the training set, 9-fold cross-validation is applied. This will further divide the training data into a smaller training dataset, referred to as the validation training set, and a separate validation test dataset. This process is shown in Figure 5.2. This further cross-validating is done to prevent overfitting and determine appropriate parameters for the model. The subset enables fine-tuning of the models and parameters of the models and dimension reduction will be tuned according to these validation datasets. Before the features are used as training data for the classifier, they undergo dimensionality reduction. Principle Component Analysis (PCA) is chosen for this task, because of its efficient performance.[8] PCA is a procedure that reduces the dimensionality while preserving the variance of the original feature space, This technique is employed in the validation process, where it will be tested for different hyper-parameters. This will allow for comparing and investigating the impact of varying the dimensionality on the performance of the computed models. For every number of the

components, PCA is applied to the training data, which will result in reduced feature space. This reduced feature set is used in the grid search technique given by the sklearn package. Grid search is a method for performing hyper-parameter optimization and thus finding the parameters of the classifiers that will result in the highest accuracy. After the 9-fold cross-validation, the average is taken over the obtained accuracies for all different numbers of components. The highest accuracy value and the most used parameter configuration are then used for evaluating the model using the test set.

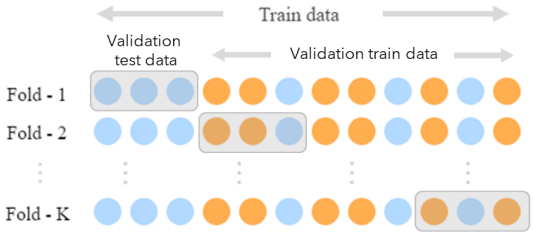


Figure 5.2: K-fold cross-validation for the parameter optimization. *Picture from:* `https://www.philschmid.de/k-fold-as-cross-validation-with-a-bert-text-classification-example`

5.1. Classifiers

In this paper, the performance of the preprocessing and features extraction mentioned in the previous chapter is compared with multiple classifiers: Logistic Regression (LR), Linear Discriminant Analysis (LDA), K-Nearest Neighbours (KN), Decision Tree, Gaussian, and Support Vector Machine. The LDA will be used for the evaluation of selecting the best preprocessing techniques to obtain the highest accuracy. The classifiers will then be compared to one other, with the optimal techniques.

Linear Discriminant Analysis

Linear discriminant analysis (LDA) is a dimensionality reduction and classification algorithm that learns the most discriminative axes between the classes. LDA aims to find the best linear projections to maximise the distance between the mean and minimize the scattering, the standard deviation, of the class. This relation is called Fisher's criterion and showed in equation 5.1.

$$J(W) = \frac{(\mu_1 - \mu_2)}{s_1^2 + s_2^2} = \frac{\text{Between-class variance}}{\text{Within-class variance}} \tag{5.1}$$

LDA reduces the dimensionality of the dataset until the dimensions are equal to the unique classes minus 1. Important to mention is that the features whose dimensions are reduced are approximated by a Gaussian distribution. LDA has the hyper-parameter 'solver', which indicates which algorithmic approach is used to find the projections. There are three solver types where one will be chosen for the evaluation of the test set, i.e., Singular Value Decomposition, Least Squares Solution and Eigenvalue Decomposition. For example, eigenvalue decomposition extracts the eigenvectors associated with the largest eigenvalues. These eigenvectors define the directions in the feature space along which the data will be projected. [9]

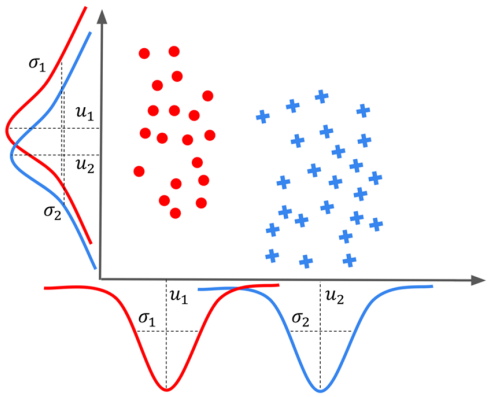


Figure 5.3: Two different LDA axes and their sketches. The axis, seen as the y-axis, is not maximized by the Fisher Criterion and the other axis, the x-axis, is.

5.1.1. Logistic Regression

Logistic regression is a binary classifier that estimates the probability, by evaluating a logistic function to model the relationship between the features and their corresponding labels. If the estimated probability is greater than 50%, the model predicts the positive class, in the other case the model predicts the negative class.

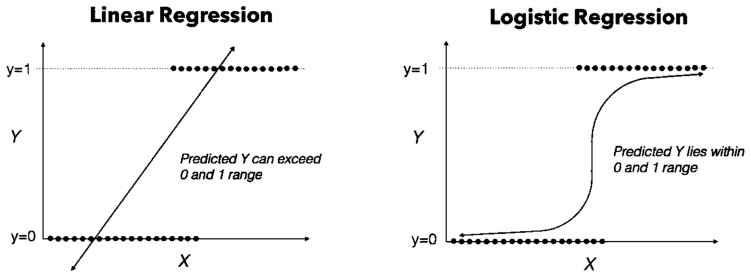


Figure 5.4: The concept of Linear Regression versus Logistic Regression on a binary dataset. *Picture from:* `https://towardsdatascience.com/`

Where linear regression uses the 'least squares' method that minimizes the error of the relationship, logistic regression uses the maximum likelihood which will act as its cost function to determine a relationship with the labels. For every sample, the likelihood is computed and summed together, which will give the likelihood of the data given the relationship. To optimize and find the highest likelihood, the logistic regression uses regularization a penalty term to the loss functions. Logistic regression can use Lasso regularization (l1) and Ridge regularization (l2) as its hyper-parameter 'penalty'. Lasso regression adds a penalty term to the loss function that is proportional to the sum of the absolute values of the coefficients, the parameters or the weights assigned to each feature variable. This means that the size of the penalty is determined by the total magnitude of the coefficients. This penalty promotes the model to select a few features by changing some coefficients to zero. In other words, Lasso regression performs feature selection by shrinking less important coefficients to zero, effectively removing those features from the model, which can be useful for high-dimensional datasets. Ridge regression on the other hand adds a penalty term that is proportional to the sum of the squared values of the coefficients. This penalty term encourages the model to spread the weight across all the features and avoids extreme or large coefficient values. [9]

K Nearest Neighbors

K-nearest neighbours classifier (KN) is a rather simple classifier, that is a type of instance-based or memory-based learning. KNN compares the new data points which have to be classified, to the training data points and finds the k nearest neighbouring points. Majority voting is then used on these k nearest

neighbouring points to determine the class label. The value k is a hyperparameter, which causes overfitting when chosen too low, and underfitting when chosen too high. Therefore multiple values for k are evaluated to obtain the highest accuracy. 'Weights' is another hyper-parameter that will be tuned for the KN classifier. It can be set to uniform where every point is weighted equally, or distance, where the weight is higher when the points are close. Another parameter is 'algorithm', which indicates which algorithm is used to find the closest data points. This will be set on 'auto', which will attempt to decide the most appropriate algorithm based on the values to the fit method. Brute force for example computes the distances between the new point and all training points but is not used in the parameter search process.

Decision Tree

One of the few classifiers that do not need feature scaling or centring is the decision tree classifier. The classifier builds a tree where every node contains conditions or features that split the data. The classifiers can have different ways of feature selection and assessing the quality of the node split. Also, there are some stopping criteria like the maximum depth of the tree (the maximum number of levels in the tree), the minimum samples needed for a split and more to prevent the tree from growing and overfitting. The decision tree is a classifier where a lot of hyperparameters can be defined by the user, but in this project, only the first two hyperparameters will be used and optimized. On the end of the tree, the location of the leaves, the labelling takes place. One more advantage of the decision tree is the fact that the model is not a black box model but a white box model and therefore the labelling can be explained if needed. A disadvantage of this classifier is the need for orthogonal decision boundaries to an axis, thus they are sensitive to data rotations. This issue can be limited by applying PCA, which can result in a better rotation of the data.

Gaussian Naive Bayes

Gaussian Naive Bayes (GNB), is a classifier that assumes the features can be explained as a Gaussian distribution and are independent from each other. The classifier is trained by estimating the mean and variance of each feature. When classifying new data, the Gaussian NB classifier applies Bayes' theorem to calculate the posterior probability of each class. The posterior can be calculated by multiplying the class prior with the class-conditional probabilities for each feature value. The predicted class is the one with the highest posterior probability. Only the hyper-parameter 'var smoothing' is optimized for this classifier, which adds a value to the variance. This variance is calculated for the training data and can be very small in some cases. The hyper-parameter prevents the Gaussian formula can have a very large output.

Support Vector Machine

A support vector machine (SVM) is one of the most popular models in machine learning. This classification algorithm will try to find the optimal separation between different classes of data points. The algorithm is based on the concept of maximizing the margin between the so-called hyperplane that separates the classes and the nearest data points from each class, known as support vectors. This classifier is, however, sensitive to different feature scales, in order to get the best-performing model. For this final classifier, 3 hyper-parameters are used to optimize the accuracy C, kernel and degree. A kernel is the most important parameter of the classifier because it transforms the data into a space where it can more easily be separated. The degree represents the order of the polynomial of the kernel function. A high degree can cause overfitting. To prevent the model to overfit, for example for outliers, the SVM does not have to separate all the data from the classes. This is called soft margin classification. C will determine how much miss classification is to be avoided, which is shown in Figure 5.5. [9]

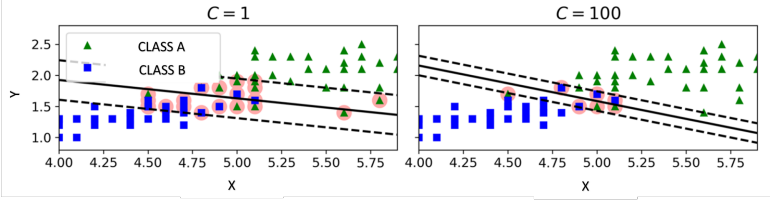


Figure 5.5: The concept of Support Vector Machine with the hyper-parameter [9]

6 Results

The chapter presents the findings and analyses derived from the data collected during the study. The primary objective of this section is to provide a comprehensive overview of the outcomes and observations obtained through research and analysis. By presenting the results, we aim to address the program of requirements outlined in chapter 2. It is crucial to acknowledge the limitations and potential sources of error, and we will address these in the discussions in chapter 7.

6.1. Eye blink detection and removal

The main metric used to evaluate eye blink detection is the F1-score. This is because it is important for the model's ability to both capture blinks and be accurate with the blinks it does capture. First, the results of eye blink detection with thresholding are shown in Table 6.1 with different threshold values. The results are obtained with EEG segments of one second with both datasets filtered with a 4th-order bandpass filter [7Hz 30Hz]. Both datasets (detailed description in section 1.2) are segmented such that there is an equal amount of blinks and no blinks. Dataset B contains 100 blinks and 100 no blinks. EEG-IO contains 468 blinks and 468 no blinks.

Table 6.1: Average scores with thresholding for eye blink detection

Threshold	Dataset	Recall	Precision	F1 score
1.5SD + μ	EEG-IO	1.00	0.50	0.66
	own dataset B	1.00	0.50	0.66
3SD + μ	EEG-IO	0.92	0.53	0.67
	own dataset B	1.00	0.53	0.70
3.5SD + μ	EEG-IO	0.70	0.58	0.64
	own dataset B	0.98	0.67	0.80
4.5SD + μ	EEG-IO	0.19	0.58	0.29
	own dataset B	0.39	0.79	0.52

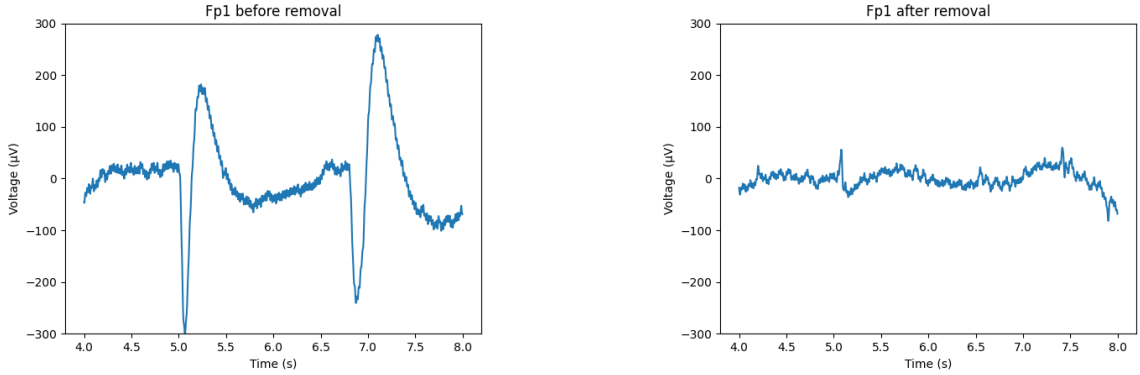
It can be seen that for a standard deviation of 1.5 the recall is 1.0 for both datasets while the precision is 0.5. This indicates that the threshold is too low and that every EEG segment is predicted as containing a blink. For 4.5 SD the recall score for both datasets is on the lower end. This indicates that now the threshold is too high and that most EEG segments is predicted as containing no blinks. The threshold value with the best scores for both datasets is 3.5 SD $+\mu$, since these have on average the best F1 score.

In Table 6.2, the average score for random forest eye blink detection in a 10-fold cross-validation with the same conditions as eye blink detection with thresholding can be seen. Comparing this to Table 6.1 the value for each score for the random forest is better. So, random forest will be used to detect eye blinks for removal.

Table 6.2: Average scores of 10-fold cross-validation for random forest for eye blink detection for EEG segments of 1 second. Both datasets are filtered with a 4th-order bandpass filter [7Hz 30Hz].

Dataset	Recall	Precision	F1 score
EEG-IO	0.83 ± 0.15	0.83 ± 0.06	0.84 ± 0.09
Own dataset B	0.97 ± 0.06	0.97 ± 0.04	0.97 ± 0.04

In Figure 6.1, two blinks are removed in an EEG segment of own dataset B. This shows that the blink removal does work. Additional examples of blink removals can be seen in section A.5



(a) EEG before blink removal

(b) EEG after blink removal

Figure 6.1: Example of blink removal for the Fp1 electrode with the algorithm

6.2. Classification

In the following sections, observations regarding the results will be discussed for the different datasets mentioned chapter 1. To maintain clarity and an organized results section, specific results are presented in this chapter. All results containing more detailed performance metrics are shown in the Appendix A. The tables shown contain performance measures in the format of mean $\pm$ standard deviation.

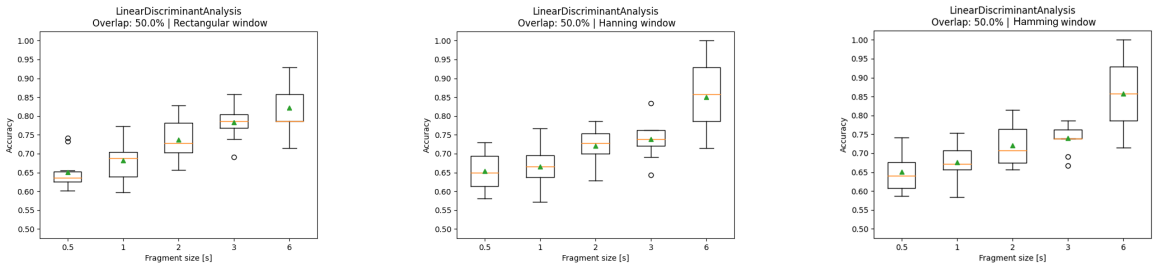
6.2.1. Graz dataset

Raw data

In the case of the Giraz dataset, the raw data has been pre-processed using a bandpass in the range from 0.5 to 30 Hz. Looking at the accuracies shown in Figure 6.2, a few things are noteworthy. Firstly, when a time window applied of 6 seconds, the accuracy is higher for the Hanning and Hamming window in comparison with the rectangular window. For smaller time windows, the rectangular window performs better or the same as the two other windows. The rectangular window especially performs better than the other window functions when a 3 seconds time window is used. Examining the tables shown in subsection A.1.1, the standard deviation remains relatively stable across different window functions. The behaviour of the balanced accuracy is roughly comparable to the accuracy, although the Hanning window displays a slightly lower extreme, shown in Figure 6.3.

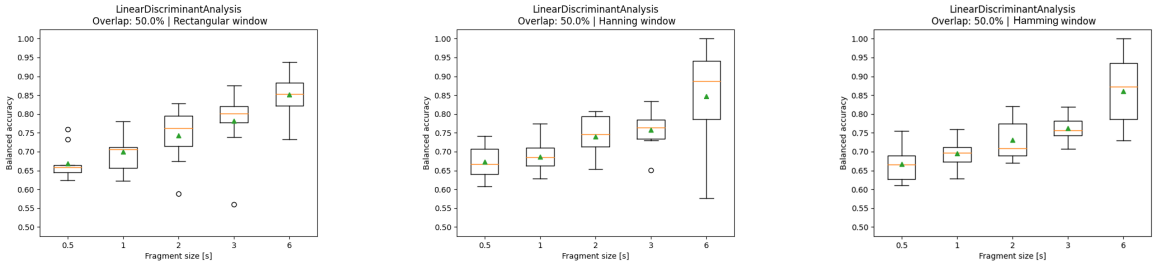
Table 6.3: Accuracy for raw Graz dataset for all window sizes.

Time window [s]	Rectangular window	Hanning window	Hamming window
0.5	0.65 ± 0.05	0.65 ± 0.05	0.65 ± 0.05
1	0.68 ± 0.05	0.66 ± 0.05	0.67 ± 0.05
2	0.74 ± 0.05	0.72 ± 0.05	0.72 ± 0.05
3	0.78 ± 0.04	0.73 ± 0.05	0.74 ± 0.04
6	0.82 ± 0.06	0.85 ± 0.10	0.85 ± 0.08



(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure 6.2: Accuracy for different window sizes



(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

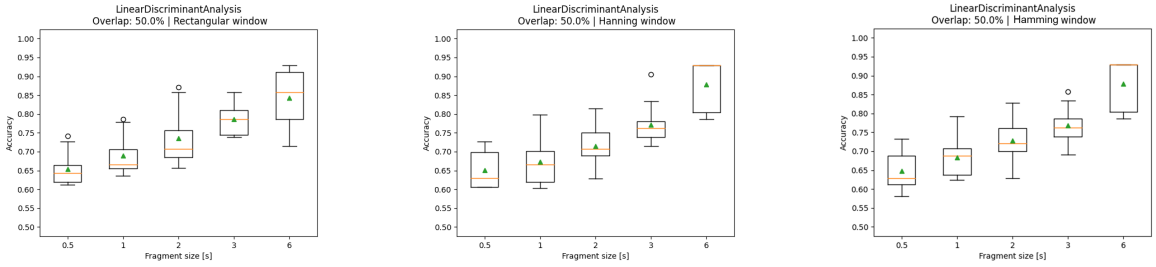
Figure 6.3: Balanced accuracy for different window sizes

Filtered data

This section shows and compares the results for different window functions when the data is filtered using a 7-30 Hz bandpass filter. As depicted, in the accuracy plots of Figure 6.2, both the data when multiplied by Hanning and Hamming windows show increased accuracies in comparison with the raw or partially filtered data, shown in Figure 6.2 and Table 6.4. This can also be confirmed by the tables presented in subsection A.1.2. The accuracy is still higher for the Hanning and Hamming window in comparison when the rectangular window for the 6 seconds window. Moreover, their accuracies are higher for all time windows, whereas the rectangular window accuracies remained about the same. The Principal Component Analysis parameter that yields the highest accuracy in the validation process becomes lower the higher the time window. In this case, Hanning and Hamming applied the dimension feature space remains bigger, in comparison when the rectangular window is employed.

Table 6.4: Performance measures for a window size of 0.5 seconds

Time window [s]	Rectangular Window	Hanning Window	Hamming Window
0.5	0.65 ± 0.04	0.65 ± 0.05	0.65 ± 0.05
1	0.69 ± 0.05	0.67 ± 0.058	0.69 ± 0.05
2	0.74 ± 0.07	0.71 ± 0.06	0.73 ± 0.06
3	0.79 ± 0.04	0.77 ± 0.06	0.77 ± 0.05
6	0.84 ± 0.07	0.88 ± 0.06	0.88 ± 0.06



(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

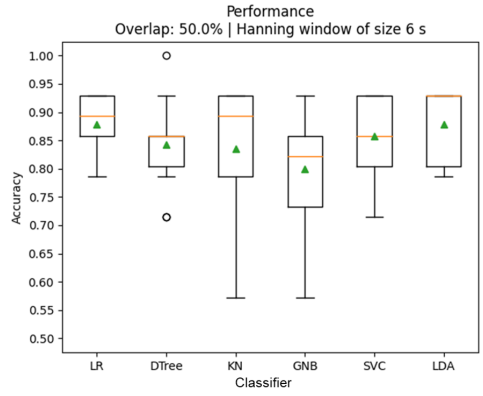
Figure 6.4: Accuracy for different window sizes

Comparing Classifiers

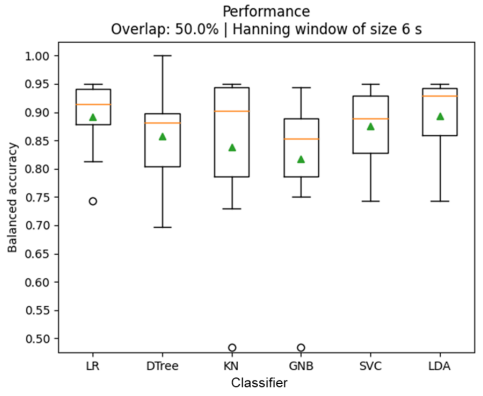
The previous subsection showed that the highest performances were obtained using the Hanning window and applying the extra filter. Therefore these attributes will be used to compare how other classifiers perform and if they can increase the accuracy. These results are shown in Figure 6.5. The highest accuracies are obtained using the classifiers: Logistic Regression and Linear Discriminant Analysis. Applying PCA seems to have the most impact on the accuracy for Logistic Regression, Gaussian Naive Bayes, Support Vector Machine and especially Linear Discriminant Analysis. For the classifiers K-Nearest Neighbours and Decision Tree, the accuracy does change that much when applying different numbers of components for the dimensionality reduction.

Table 6.5: Performance measures for a window size of 6 seconds with different classifiers on Hanning window

	Accuracy	Balanced Accuracy
LR	0.87 ± 0.05	0.89 ± 0.06
DTree	0.84 ± 0.08	0.85 ± 0.09
KN	0.83 ± 0.11	0.83 ± 0.14
GNB	0.80 ± 0.10	0.81 ± 0.13
SVC	0.85 ± 0.07	0.87 ± 0.07
LDA	0.87 ± 0.06	0.89 ± 0.07



(a) Accuracy



(b) Balanced accuracy

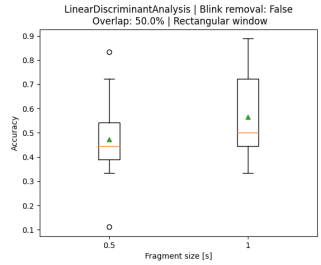
Figure 6.5: Accuracy's for different classifiers obtained using the Hanning window function

6.2.2. Classification of OpenBCI dataset

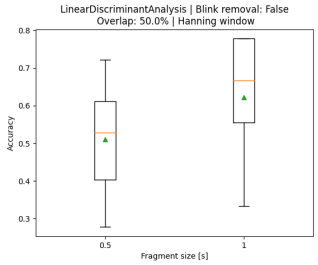
In this subsection the performance of the OpenBCI dataset is discussed. Important to note is that the dataset contains 3 recordings of 12 actions.

Raw

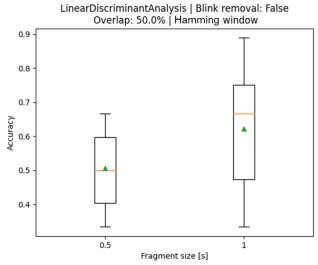
Figure 6.6 shows the accuracies for the different time functions and for the time windows of 0.5 and 1 second. The accuracies when a time window of 0.5 second is used around 50 percent. A time window of 1 second, gives a higher accuracy overall. The results show a high variance.



(a) Rectangular window



(b) Hanning window



(c) Hamming window

Figure 6.6: Accuracy for different window sizes

Table 6.6: Accuracy for all time windows

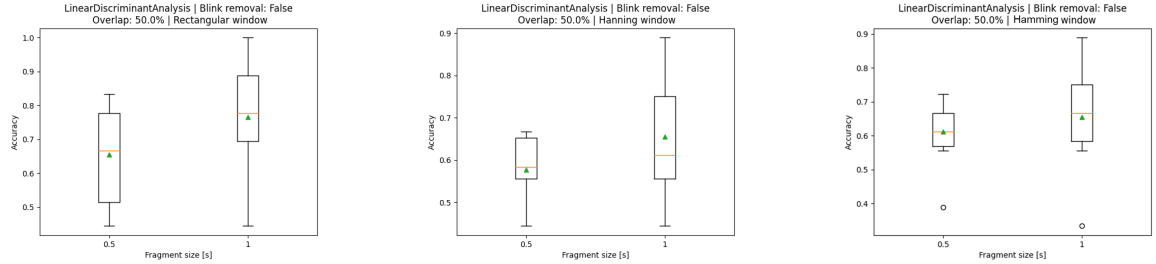
Time window [s]	Rectangular Window	Hanning Window	Hamming Window
0.5	0.56 ± 0.20	0.62 ± 0.16	0.62 ± 0.19
1	0.65 ± 0.14	0.57 ± 0.07	0.61 ± 0.09

Filtered

Filtering the OpenBCI data improves the accuracy for all function windows, but the magnitude of the change varies. The rectangular window function, shows the highest accuracy and increase of accuracy for both time window sizes. The other windows increase but not as much. The results are shown in Figure 6.7 and subsection A.2.2.

Table 6.7: Performance measures for a window size of 1 second

Time window [s]	Rectangular Window	Hanning Window	Hamming Window
0.5	0.66 ± 0.14	0.57 ± 0.07	0.60 ± 0.09
1	0.76 ± 0.16	0.65 ± 0.14	0.65 ± 0.14

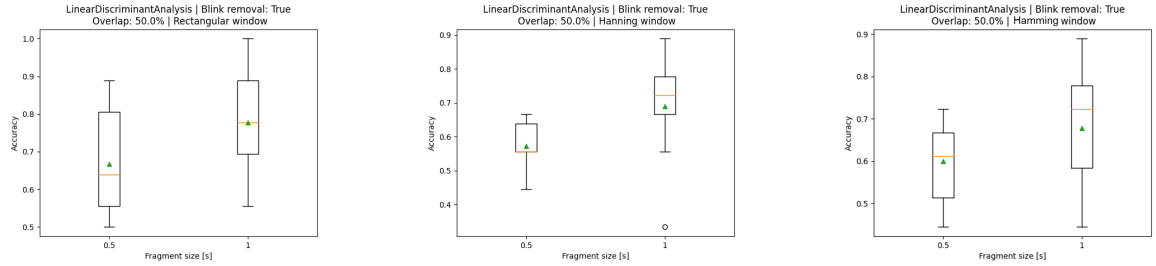


(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure 6.7: Accuracy for different window sizes

Filter and blink removal

Adding the blink removal to the pre-processing process gives a small increase in the accuracy for all window functions in comparison when only a bandpass filter is applied. The use of the blink removal does not always result in a higher accuracy for a time window of 0.5 seconds, but it does for the 1 second window.



(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure 6.8: Accuracy for different window sizes

Comparing Classifiers

The previous subsection showed that the highest performances were obtained using the rectangular window and applying next to the bandpass filter a blink artefact removal. These attributes will be used to compare how other classifiers perform and if they can increase the accuracy. These results are shown in Figure 6.9. The highest accuracies are obtained using the classifiers: Logistic Regression, Support Vector Machine and Linear Discriminant Analysis. Applying PCA seems to have the least impact on the accuracy of the classifier K-Nearest Neighbours. The plots are shown in subsection A.2.4

Table 6.8: Performance measures for a window size of 1 second

Time window [s]	Rectangular Window	Hanning Window	Hamming Window
0.5	0.66 ± 0.14	0.57 ± 0.07	0.60 ± 0.09
1	0.76 ± 0.16	0.65 ± 0.14	0.65 ± 0.14

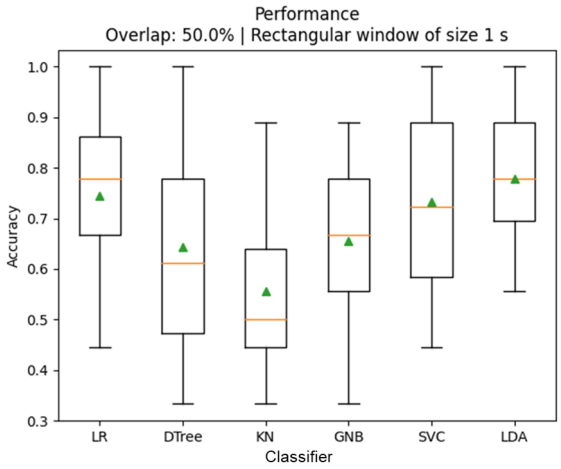


Figure 6.9: Accuracy for different classifiers

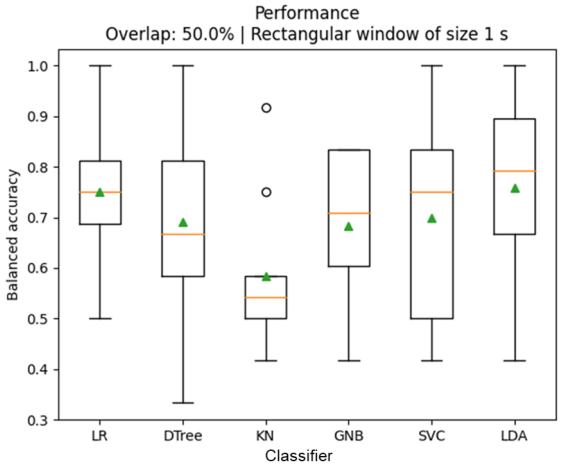


Figure 6.10: Balanced accuracy for different classifiers

6.3. Time delay

The average computing times for both the features and eye blink removal algorithm can be seen in the following two tables. The average calculation time for the eye blink removal algorithm does not increase significantly even when the window size goes up to 1500 samples. It is assumed that the average calculation time is around 24.4ms for every window size below 1500 samples. The longest delay recorded on average is 26.79ms. The longest delay in the feature calculation time is 7.42ms for a window size of 768 samples. Then the longest delay combining the feature extraction and eye blink removal is 34.21ms.

Table 6.9: Time delay of the feature calculation for different window sizes

Window size [samples]	Average feature calculation time [ms]
64	4.67
128	6.09
256	6.40
384	5.64
768	7.42

Table 6.10: Average time delay random forest blink removal for different window sizes

Window size [samples]	Time [ms]
125	24.77
250	22.98
375	23.22
500	26.79
1500	24.47

7

Discussion

As discussed in chapter 6, the blink removal algorithm gives an increase in accuracy. But it can most likely still removes signals that could be of interest. The EEG signal shape after blink removal in the regions between blinks is clearly different, which is shown in section A.5. It was assumed that the eight electrodes used in our OpenBCI only picked up eight independent sources, but in reality, there are most probably more independent sources at play. A requirement for ICA to work is that the amount of observations (channels) is equal to the number of independent sources. It is hard to tell exactly how many independent sources there are. Therefore it could be argued that ICA is not the perfect solution for blink removal. We have also not tested the extent to which the relevant signals have been removed.

Only the Butterworth filter was considered for a bandpass filter. Other types of filters could have been considered, which might have improved the overall accuracy. Furthermore, the cut-off frequencies of the bandpass filter were determined by the alpha- and beta-band frequency ranges. However, there is variation between studies about the frequency range of the alpha activity. In a study by Kirar, Jyoti Singh and Agrawal, R. K.[12], it is mentioned that alpha activity lies between 7-12Hz, however in a study by Padfield, N.[16] it states that it lies between 8 and 12 Hz. We have found more studies that either use the 7-12 or 8-12 range for the alpha band. So it is not entirely clear which frequency range is correct for the alpha band.

All training and validation data for random forest eye blink detection are based on voluntary blinks. There was not enough time to set up a validation set with involuntary blinks (natural blinks), which could have led to a different accuracy.

In the analysis of the results of the Graz dataset, it was observed that the use of the Hanning window function with a size of 6 seconds resulted in the highest accuracy. Applying the Hamming window gave a little lower but close accuracy, which can be expected due to the similar behaviour of the two functions. Rectangular function on the other hand gives for the same window size a significantly lower accuracy. It could be speculated that the boundary data, therefore, is not so useful. The Graz dataset consists of recordings of 9 seconds, where the last 6 seconds are supposed to be action. The documentation does not really give clarity if those 6 seconds could still contain some periods, most likely located at the edges, where the subject did not perform an action. This could mean that the segmentation of the data will cause incorrect labels for the segment placed at the edges, and smaller time windows are not useful for comparison. For lower window sizes, the rectangular window function has about the same accuracy or even higher than the other two functions. So, it is uncertain which time window function performs the best when applied to another dataset. On this dataset, the Hanning function gave the highest performance, with the Hamming function as a close second.

Applying the bandpass filter causes an increase in the accuracy of all time windows, which can suggest that those lower frequencies still contain components that are not beneficial for classifying motor imagery or motor executions. This is in line with the literature, which states that these activities are

more noticeable in the alpha band, after 7 Hz and before 12 Hz.

In the classification process, the PCA has a high impact on the accuracy. Overall, larger time windows resulted in more distinct features, allowing the training data to be reduced to a smaller number of feature spaces to achieve higher accuracy. Contrarily, smaller time windows showed a higher number of components or feature spaces to achieve the highest accuracy. This could be due to the fact that smaller windows contain less data, and therefore needs more features for effective analysis.

The last point of discussion for the Graz dataset is about the results of the different classifiers. The Linear Discriminant Analysis and Logistic Regression gave the best-performing models for the 6-second Hanning window. The pre-processing methods are selected using the LDA classifier, therefore it is expected that the model works relatively well and so other models have a disadvantage. The results also indicate that LR could be a possible alternative for this type of analysis.

In the analysis of the results of the OpenBCI dataset, and the recordings made by the measurement group, it can directly be seen that the variance is very high. This is due to the small amount of data these recordings consist of. Although the mean accuracy can give an intuition of the performance of the model, it is hard to say which model and methods will work better when more data is recorded and added to the training set.

8 Conclusion

The goal of this subgroup is to develop a decoder of EEG signals that can accurately produce outputs based on the limb that is moved to be used in real time. In order to realize this goal several requirements are set as stated in chapter 2.

Firstly, a framework for a model is developed based on the Graz dataset. This model can distinguish between two classes with an accuracy of at least 80%. Classification of the data for three classes can be accomplished, but the results were not made due to limited time.

A 4th-order Butterworth is used to filter out frequencies outside the interval of [7Hz 30Hz]. After applying this filter to the Graz dataset for the Hanning and Hamming window both the data show increased accuracies in comparison with the raw or partially filtered data of the Graz dataset. The average accuracy in classification between two classes for the Hanning window on a window size of 6 seconds is the best with the LDA classifier, with an accuracy of 0.88 ± 0.06.

Another requirement met, is that eye blink artefacts have to be removed. In order to be able to remove eye blinks it has to be detected first. By using a random forest classifier, eye blinks that are present in EEG from the OpenBCI headset can be detected with an f1-score of 97%. Using ICA it is then possible to remove these eye blinks.

The last requirement is that the decoder has to be developed that can distinguish between two classes on our own OpenBCI dataset, with an accuracy of at least 60%. After employing both the Butterworth bandpass filtering and eye blink removal algorithm with an LDA classifier. An average accuracy can be achieved $0.76\% \pm 0.16$ on a rectangular window of one second.

Since the decoder is going to be used in real-time a requirement is that the decoder must have a maximum latency of 500ms. The latency is defined as the combined computation time of preprocessing and feature extraction. The longest delay for the eye blink removal algorithm is 26.79ms. The biggest delay for feature calculation on average was 7.24ms for a window size of 768 samples. Then the total contribution to the time delay on a window size of 768 for a single classification is 34.21ms. Any window size lower than 768 will have a lower time delay. So the decoder is suitable for real-time application but this has not been demonstrated.

The main requirements have been met and it is possible to continue with this model for integration with the other subgroups and implement it in an actual real-time BCI. The other 'side' requirements (shoulds and coulds), can be finished when there are more recordings given from the measurement group.

8.1. Future Work

Other preprocessing methods such as CAR and Surface Laplacian which are also popular methods have been considered in this paper to implement, however, the requirements for these methods to work

properly were not met so they were not tested. Common average referencing (CAR) is mostly used as a basic dimensionality reduction technique. This method decreases noise across all electrodes by computing the average of the signal of all electrodes and then subtracting it from the EEG at every electrode for every sample. The motivation behind using this method is that the head can be assumed as a sphere and assuming that the potential recorded on the whole sphere due to current sources is zero. However, in practice, to achieve such an ideal reference one would require a large number of electrodes that cover the whole head uniformly, which is not the case in our implementation where 8 channels are used and are not distributed uniformly. This method could be attempted if more electrodes are available.

Surface Laplacian (SL) refers to a method of displaying EEG with a high spatial resolution. The EEG scalp potential of SL is generally measured based on two main assumptions: 1) the electrodes are equidistant, and 2) the surface of the scalp in the near-electrode region is as flat as possible. The new value for each electrode is the average of the sum of four electrodes in the near region. The reason this was not tested is that our layout with eight electrodes could not meet the two assumptions. Again this is mainly due to not having enough electrodes. This method could be attempted if more electrodes are available.

In this paper only ICA is considered for blink removal. Other algorithms exist, such as an algorithm based on empirical mode decomposition (EMD) which may work better in retaining the signals of interest after removing blinks.
There is still a lot of room for improvement in the feature extraction. Firstly, the paper does not evaluate the different features that are implemented, apart from the literature. Therefore, it is not known whether some features actually add information to the model. By comparing the feature and their impact on the model the best features can be selected and the worst-performing ones can be left out. The second revision that could be done is adding features from the time-frequency domain, especially wavelets, which are very powerful according to the read literature.

References

[1] Mohit Agarwal and Raghupathy Sivakumar. “Blink: A Fully Automated Unsupervised Algorithm for Eye-Blink Detection in EEG Signals”. In: *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. 2019, pp. 1113–1121. DOI: 10.1109/ALLERTON.2019.8919795.

[2] *BCI competition II dataset III*. URL: https://www.bbci.de/competition/ii/.

[3] Benjamin Blankertz et al. “Optimizing Spatial filters for Robust EEG Single-Trial Analysis”. In: *IEEE Signal Processing Magazine* 25.1 (2008), pp. 41–56. DOI: 10.1109/MSP.2008.4408441.

[4] Poomipat Boonyakitanont et al. “A review of feature extraction and performance evaluation in epileptic seizure detection using EEG”. In: *Biomedical Signal Processing and Control* 57 (2020), p. 101702. ISSN: 1746-8094. DOI: https://doi.org/10.1016/j.bspc.2019.101702. URL: https://www.sciencedirect.com/science/article/pii/S1746809419302836.

[5] Elie Bou Assi, Sandy Rihana, and Mohamad Sawan. “Kmeans-ICA Based Automatic Method for Ocular Artifacts Removal in a Motor Imagery Classification”. In: vol. 2014. Aug. 2014. DOI: 10.1109/EMBC.2014.6945154.

[6] Zhenghua Chen et al. “Robust human activity recognition using smartphone sensors via CT-PCA and online SVM”. In: *IEEE Transactions on Industrial Informatics* 13.6 (2017), pp. 3070–3080. DOI: 10.1109/tii.2017.2712746.

[7] R. Esteller et al. “Line length: An efficient feature for seizure onset detection”. In: *2001 Conference Proceedings of the 23rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society* (). DOI: 10.1109/iembs.2001.1020545.

[8] Q. Fournier and D. Aloise. “Empirical Comparison between Autoencoders and Traditional Dimensionality Reduction Methods”. In: *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*. Los Alamitos, CA, USA: IEEE Computer Society, June 2019, pp. 211–214. DOI: 10.1109/AIKE.2019.00044. URL: https://doi.ieeecomputersociety.org/10.1109/AIKE.2019.00044.

[9] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O’Reilly Media, 2017. ISBN: 978-1491962299.

[10] Moritz Grosse-Wentrup and Bernhard Schölkopf. “A Review of Performance Variations in SMR-Based Brain−Computer Interfaces (BCIs)”. In: Mar. 2013, pp. 39–51. ISBN: 978-3-642-36082-4. DOI: 10.1007/978-3-642-36083-1_5.

[11] J.F. Kaiser. “On a simple algorithm to calculate the ’energy’ of a signal”. In: *International Conference on Acoustics, Speech, and Signal Processing*. 1990, 381–384 vol.1. DOI: 10.1109/ICASSP.1990.115702.

[12] Jyoti Singh Kirar and R. K. Agrawal. “Relevant Frequency Band Selection using Sequential Forward Feature Selection for Motor Imagery Brain Computer Interfaces”. In: *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2018, pp. 52–59. DOI: 10.1109/SSCI.2018.8628719.

[13] S. Marchesotti et al. “Quantifying the role of motor imagery in brain-machine interfaces”. In: *Scientific Reports* 6 (Apr. 2016). ISSN: 2045-2322. DOI: 10.1038/srep24076. URL: https://doi.org/10.1038/srep24076.

[14] Kai J. Miller et al. “Cortical activity during motor execution, motor imagery, and imagery-based online feedback”. In: *Proceedings of the National Academy of Sciences* 107.9 (2010), pp. 4430–4435. DOI: 10.1073/pnas.0913697107. URL: https://www.pnas.org/doi/abs/10.1073/pnas.0913697107.

[15] M. F. Mridha et al. “Brain-Computer Interface: Advancement and Challenges”. In: *Sensors* 21.17 (2021). ISSN: 1424-8220. DOI: 10.3390/s21175746. URL: https://www.mdpi.com/1424-8220/21/17/5746.

[16] Natasha Padfield et al. “EEG-Based Brain-Computer Interfaces Using Motor-Imagery: Techniques and Challenges”. In: *Sensors* 19.6 (2019). ISSN: 1424-8220. DOI: 10.3390/s19061423. URL: https://www.mdpi.com/1424-8220/19/6/1423.

[17] Prajoy Podder et al. “Comparative performance analysis of hamming, Hanning and Blackman Window”. In: *International Journal of Computer Applications* 96.18 (2014), pp. 1–7. DOI: 10.5120/16891-6927.

[18] Theerat Saichoo, Poonpong Boonbrahm, and Yunyong Punsawad. “Investigating user proficiency of motor imagery for EEG-based BCI system to control simulated wheelchair”. In: *Sensors* 22.24 (2022), p. 9788. DOI: 10.3390/s22249788.

[19] Leontin Tuță et al. “Real- Time EEG Data Processing Using Independent Component Analysis (ICA)”. In: *2022 14th International Conference on Communications (COMM)*. 2022, pp. 1–4. DOI: 10.1109/COMM54429.2022.9817209.

[20] Yuang Zhang et al. “RT-Blink: A Method Toward Real-Time Blink Detection From Single Frontal EEG Signal”. In: *IEEE Sensors Journal* 23.3 (2023), pp. 2794–2802. DOI: 10.1109/JSEN.2022.3232176.

A

All Results

Please note that there are a few typographical errors present in the plots presented in this appendix. However, these errors do not affect the content or findings of the paper, and they are correctly referred to and discussed in the accompanying text. We apologize for any confusion that these typos may cause. In some plots where the hamming time window is used to obtain the results, the label incorrectly reads 'humming' instead of 'hamming.' This is purely a typographical mistake and does not alter the method or results. In the plots comparing classifiers, the x-label is mislabeled as 'fragment size [s],' while it should correctly be labelled as 'classifier.'
The measure of evaluating the model is Cohen's Kappa coefficient. Kappa is a statistical measure which is used to assess the agreement between two raters. To be clear, the agreements are the true-positive and negative Table 1.2. The coefficient gives an indication of how much better the agreement is compared to what would be expected by chance alone. The calculation of Cohen's Kappa involves the observed agreement, P_o, and expected agreement P_e. The observed agreement is defined as the total number of agreements divided by the total number of observations, which is the same as overall accuracy. The expected agreement represents the agreement that would be expected by chance. It is calculated as the total number of class one predictions divided by the total number of observations multiplied by the number of actual class one's divided by the total number of observations. The same is done for class two and then added together. The formula for the expected agreement is shown in Equation A.1. The Cohen Kappa coefficient is calculated as shown in Equation A.2. Cohen's Kappa coefficient ranges from -1 to 1, where the value 1 implies a perfect agreement, 0 indicates agreement by chance, and negative values indicate agreement worse than chance. It provides a valuable measure of the model's reliability and the quality of its predictions compared to the ground truth.

$$P_e = \frac{(TN + FN) \cdot (TN + FP)}{TP + FP + TN + FN} + \frac{(TP + FP) \cdot (TP + FN)}{TP + FP + TN + FN} \tag{A.1}$$

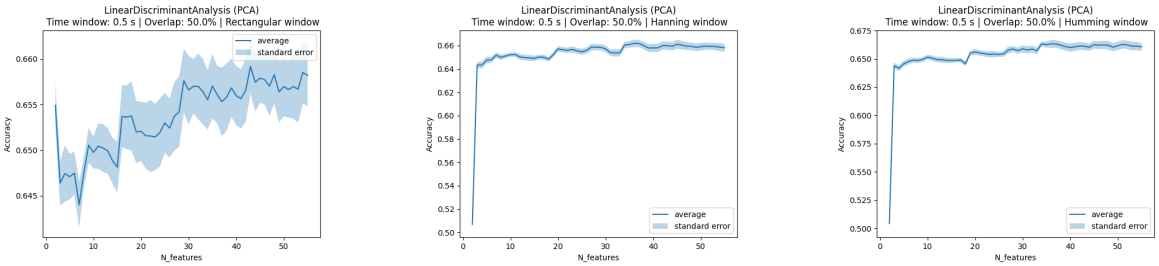
$$\kappa = \frac{P_o - P_e}{1 - P_e} \tag{A.2}$$

A.1. Graz dataset

A.1.1. Raw

Table A.1: Performance measures for 0.5 sec window

	Rectangular window	Hanning window	Hamming window
Accuracy	0.6516 ± 0.0456	0.6543 ± 0.0481	0.6506 ± 0.0499
Balanced Accuracy	0.6685 ± 0.0410	0.6729 ± 0.0414	0.6680 ± 0.0446
Kappa	0.3044 ± 0.0878	0.3126 ± 0.0913	0.3040 ± 0.0957
F1-Score	0.6370 ± 0.0768	0.6412 ± 0.0749	0.6376 ± 0.0781
Matthew Correlation Coef	0.3208 ± 0.0827	0.3295 ± 0.0849	0.3201 ± 0.0902

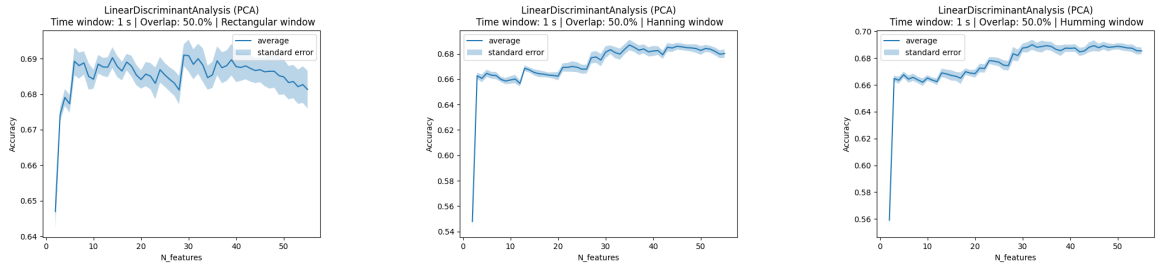


(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.1: Accuracy for different number of components used for PCA, on a time window of 0.5 seconds

Table A.2: Performance measures for 1 seconds window

	Rectangular window	Hanning window	Hamming window
Accuracy	0.6825 ± 0.0544	0.6662 ± 0.0512	0.6760 ± 0.0467
Balanced Accuracy	0.6994 ± 0.0503	0.6870 ± 0.0420	0.6957 ± 0.0348
Kappa	0.3644 ± 0.1117	0.3384 ± 0.0929	0.3544 ± 0.0834
F1-Score	0.6683 ± 0.1005	0.6547 ± 0.0898	0.6645 ± 0.0790
Matthew Correlation Coef	0.3825 ± 0.1067	0.3576 ± 0.0857	0.3740 ± 0.0729

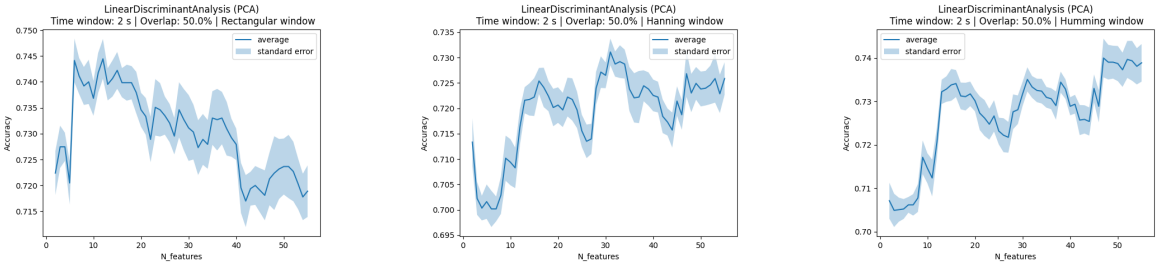


(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.2: Accuracy for different number of components used for PCA, on a time window of 1 second

Table A.3: Performance measures for 2 seconds window

	Rectangular window	Hanning window	Hamming window
Accuracy	0.7371 ± 0.0520	0.7214 ± 0.0492	0.7214 ± 0.0516
Balanced Accuracy	0.7436 ± 0.0687	0.7407 ± 0.0540	0.7307 ± 0.0518
Kappa	0.4534 ± 0.1377	0.4365 ± 0.1010	0.4280 ± 0.1122
F1-Score	0.7073 ± 0.1264	0.7060 ± 0.0881	0.7005 ± 0.0943
Matthew Correlation Coef	0.4690 ± 0.1391	0.4589 ± 0.1046	0.4425 ± 0.1077

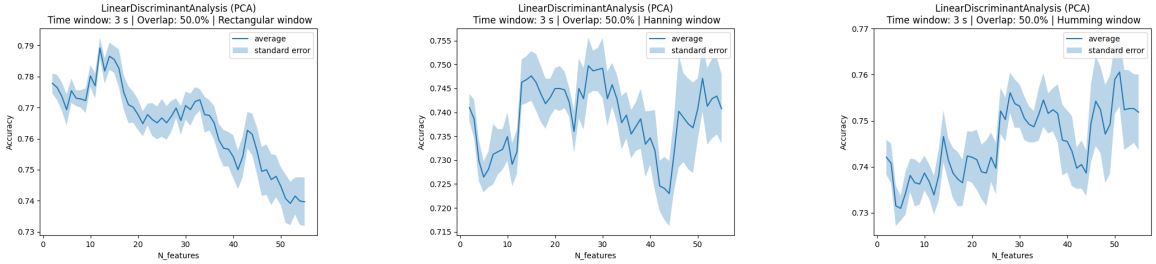


(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.3: Accuracy for different number of components used for PCA, on a time window of 2 seconds

Table A.4: Performance measures for 3 seconds window

	Rectangular window	Hanning window	Hamming window
Accuracy	0.7833 ± 0.0445	0.7381 ± 0.0476	0.7405 ± 0.0360
Balanced Accuracy	0.7825 ± 0.0827	0.7584 ± 0.0463	0.7619 ± 0.0356
Kappa	0.5331 ± 0.1505	0.4719 ± 0.1069	0.4762 ± 0.0796
F1-Score	0.7494 ± 0.1538	0.7229 ± 0.1014	0.7265 ± 0.0848
Matthew Correlation Coef	0.5496 ± 0.1561	0.4953 ± 0.1012	0.5014 ± 0.0746

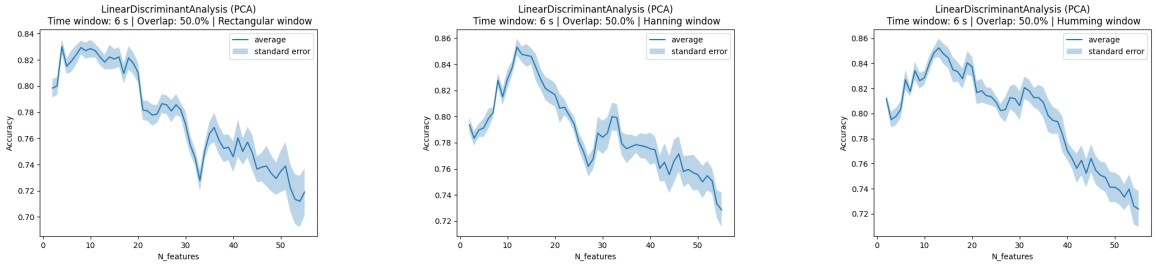


(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.4: Accuracy for different number of components used for PCA, on a time window of 3 seconds

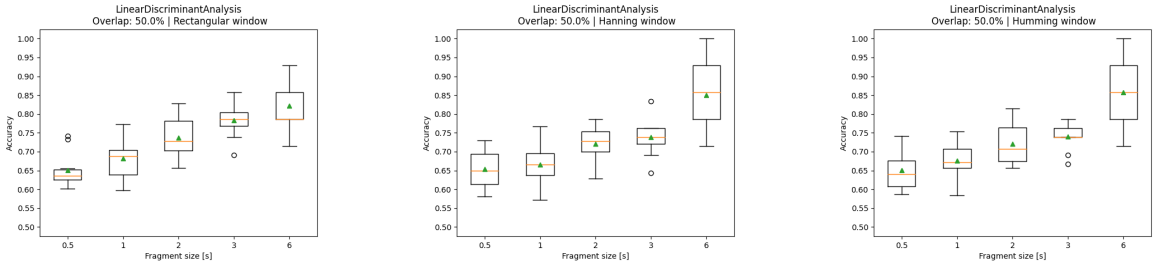
Table A.5: Performance measures for 6 seconds window

	Rectangular window	Hanning window	Hamming window
Accuracy	0.8214 ± 0.0659	0.8500 ± 0.0929	0.8571 ± 0.0845
Balanced Accuracy	0.8510 ± 0.0586	0.8478 ± 0.1218	0.8607 ± 0.0904
Kappa	0.6389 ± 0.1345	0.6677 ± 0.2351	0.6933 ± 0.1831
F1-Score	0.8148 ± 0.0803	0.8197 ± 0.1794	0.8402 ± 0.1211
Matthew Correlation Coef	0.6712 ± 0.1242	0.6802 ± 0.2364	0.7035 ± 0.1813



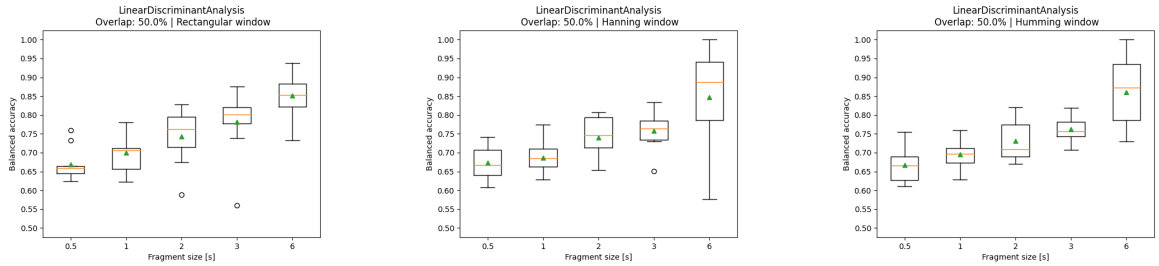
(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.5: Accuracy for different number of components used for PCA, on a time window of 6 seconds



(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.6: Accuracy for different window sizes



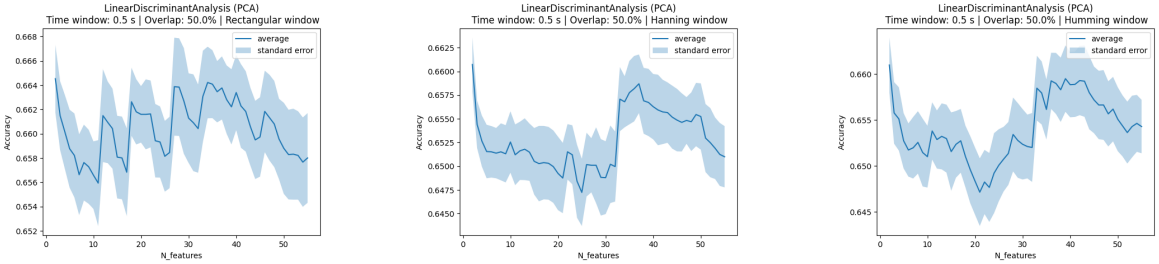
(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.7: Balanced accuracy for different window sizes

A.1.2. Filtered

Table A.6: Performance measures for a window size of 0.5 seconds

	Rectangular Window	Hanning Window	Hamming Window
Accuracy	0.6543 ± 0.0440	0.6503 ± 0.0506	0.6478 ± 0.0506
Balanced Accuracy	0.6793 ± 0.0326	0.6709 ± 0.0370	0.6680 ± 0.0387
Kappa	0.3201 ± 0.0771	0.3086 ± 0.0889	0.3029 ± 0.0903
F1-Score	0.6377 ± 0.0682	0.6299 ± 0.0751	0.6247 ± 0.0760
Matthew Correlation Coef	0.3419 ± 0.0667	0.3271 ± 0.0780	0.3209 ± 0.0803

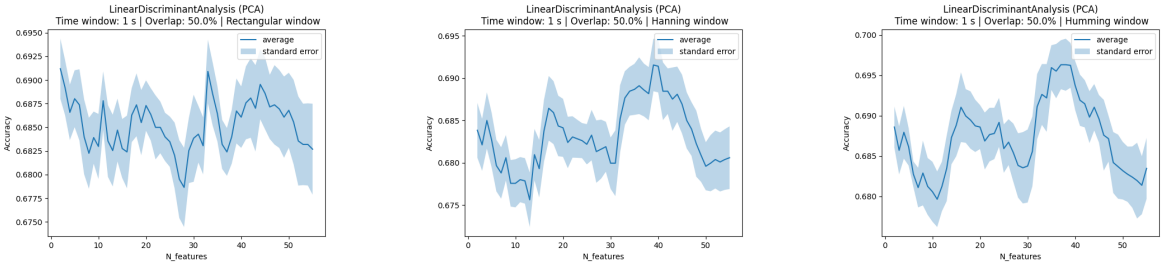


(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.8: Accuracy for different number of components used for PCA, on a time window of 0.5 seconds

Table A.7: Performance measures for a window size of 1 second

	Rectangular Window	Hanning Window	Hamming Window
Accuracy	0.6896 ± 0.0512	0.6727 ± 0.0579	0.6838 ± 0.0495
Balanced Accuracy	0.7157 ± 0.0406	0.6938 ± 0.0482	0.7086 ± 0.0429
Kappa	0.3867 ± 0.0948	0.3508 ± 0.1077	0.3735 ± 0.0906
F1-Score	0.6723 ± 0.0753	0.6636 ± 0.0793	0.6772 ± 0.0697
Matthew Correlation Coef	0.4115 ± 0.0836	0.3698 ± 0.0991	0.3960 ± 0.0831

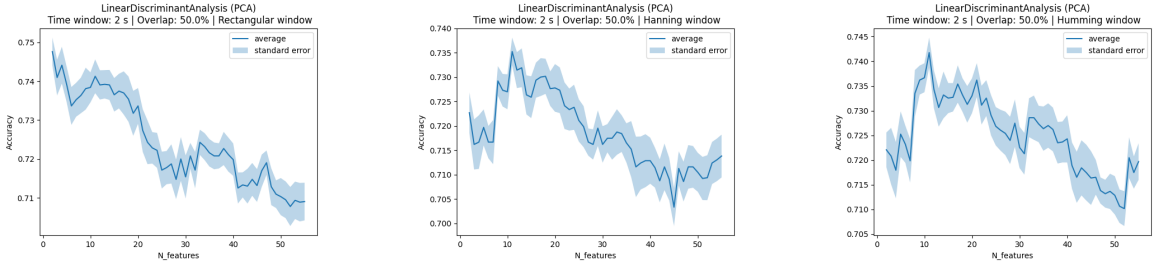


(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.9: Accuracy for different number of components used for PCA, on a time window of 1 second

Table A.8: Performance measures for a window size of 2 seconds

	Rectangular Window	Hanning Window	Hamming Window
Accuracy	0.7357 ± 0.0715	0.7143 ± 0.0575	0.7286 ± 0.0553
Balanced Accuracy	0.7535 ± 0.0751	0.7413 ± 0.0480	0.7540 ± 0.0491
Kappa	0.4645 ± 0.1534	0.4344 ± 0.1095	0.4581 ± 0.1050
F1-Score	0.7157 ± 0.1138	0.7042 ± 0.0853	0.7177 ± 0.0747
Matthew Correlation Coef	0.4868 ± 0.1513	0.4615 ± 0.0998	0.4841 ± 0.0965

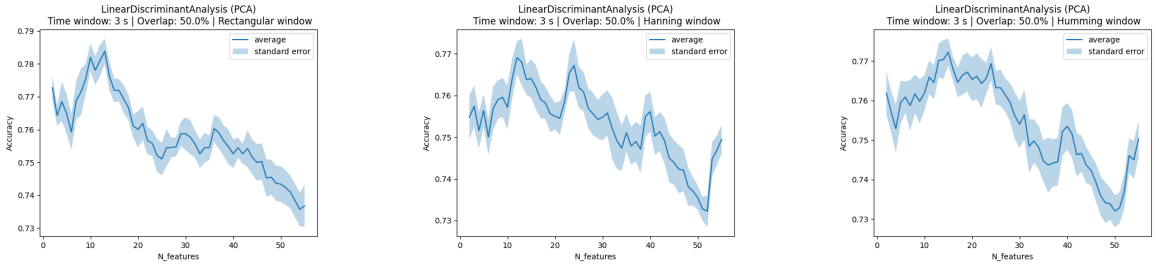


(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.10: Accuracy for different number of components used for PCA, on a time window of 2 seconds

Table A.9: Performance measures for a window size of 3 seconds

	Rectangular Window	Hanning Window	Hamming Window
Accuracy	0.7857 ± 0.0398	0.7714 ± 0.0555	0.7690 ± 0.0465
Balanced Accuracy	0.8047 ± 0.0368	0.7874 ± 0.0576	0.7861 ± 0.0460
Kappa	0.5609 ± 0.0905	0.5298 ± 0.1193	0.5266 ± 0.0895
F1-Score	0.7688 ± 0.0858	0.7544 ± 0.0885	0.7559 ± 0.0684
Matthew Correlation Coef	0.5853 ± 0.0833	0.5492 ± 0.1185	0.5464 ± 0.0850

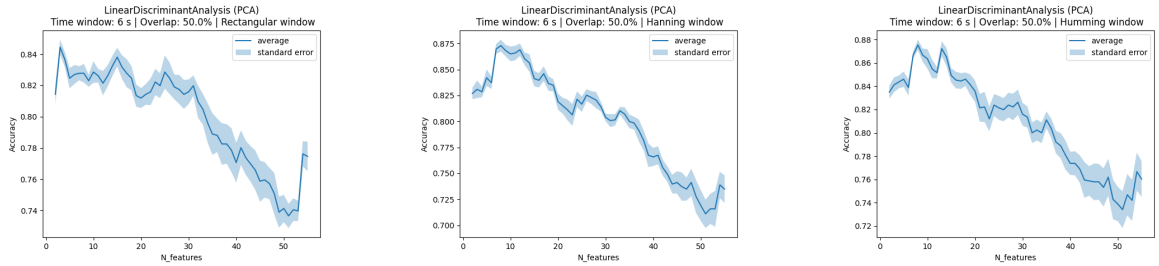


(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.11: Accuracy for different number of components used for PCA, on a time window of 3 seconds

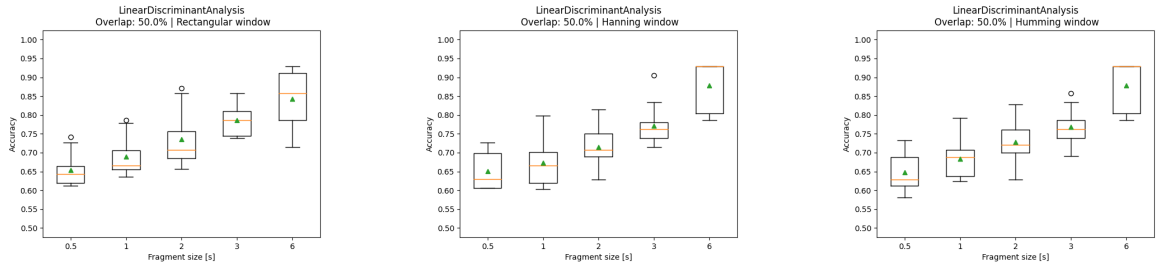
Table A.10: Performance measures for a window size of 6 seconds

	Rectangular Window	Hanning Window	Hamming Window
Accuracy	0.8429 ± 0.0700	0.8786 ± 0.0643	0.8786 ± 0.0643
Balanced Accuracy	0.8561 ± 0.0751	0.8927 ± 0.0664	0.8927 ± 0.0664
Kappa	0.6679 ± 0.1560	0.7417 ± 0.1490	0.7417 ± 0.1490
F1-Score	0.8250 ± 0.1102	0.8616 ± 0.1097	0.8616 ± 0.1097
Matthew Correlation Coef	0.6897 ± 0.1548	0.7632 ± 0.1391	0.7632 ± 0.1391



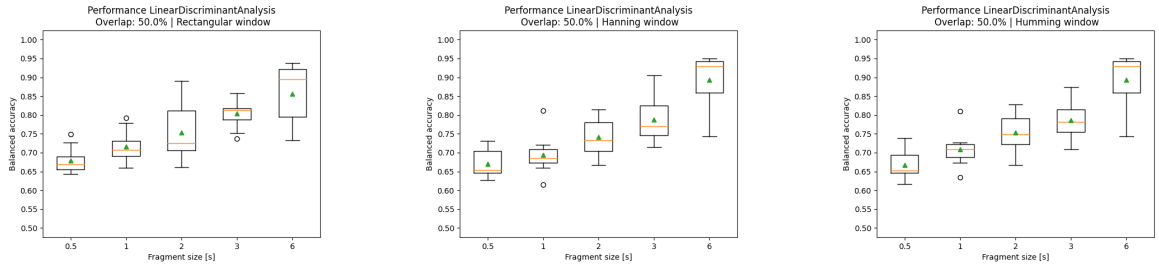
(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.12: Accuracy for different number of components used for PCA, on a time window of 6 seconds



(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.13: Accuracy for different window sizes



(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.14: Balanced accuracy for different window sizes

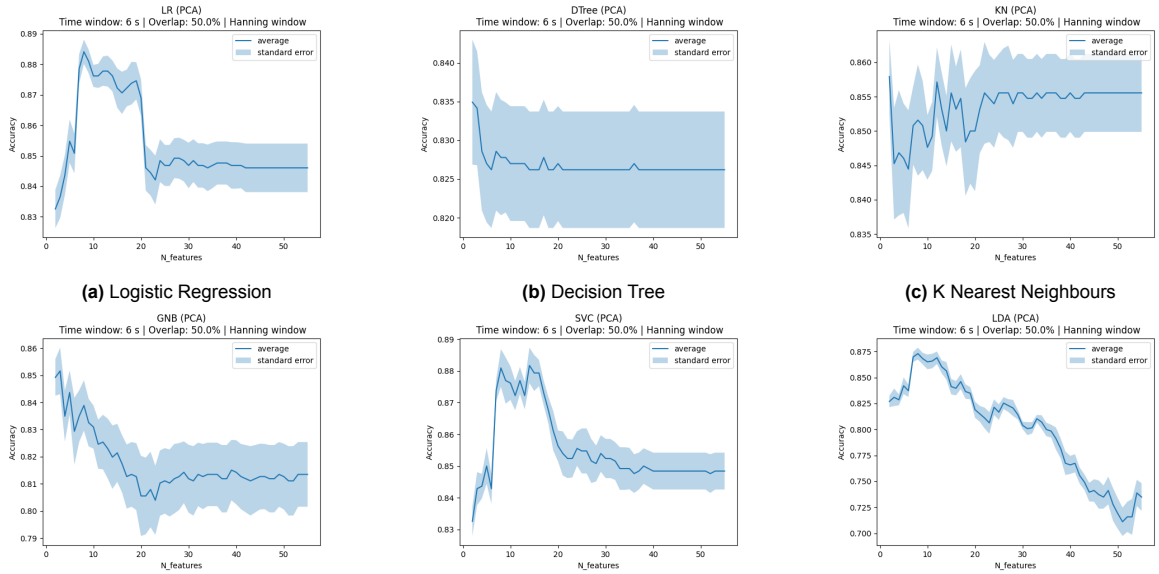
A.1.3. Comparing classifiers

Table A.11: Performance measures for a window size of 6 seconds with different classifiers

	Accuracy	Balanced Accuracy	Kappa	F1-Score	Matthew Correlation Coef
LR	0.8786 ± 0.0558	0.8915 ± 0.0637	0.7404 ± 0.1350	0.8605 ± 0.1070	0.7613 ± 0.1292
DTree	0.8429 ± 0.0833	0.8572 ± 0.0896	0.6724 ± 0.1825	0.8200 ± 0.1347	0.6962 ± 0.1772
KN	0.8357 ± 0.1154	0.8381 ± 0.1411	0.6494 ± 0.2671	0.8081 ± 0.1988	0.6604 ± 0.2702
GNB	0.8000 ± 0.1050	0.8178 ± 0.1265	0.5895 ± 0.2422	0.7652 ± 0.1877	0.6267 ± 0.2407
SVC	0.8571 ± 0.0714	0.8759 ± 0.0661	0.7033 ± 0.1525	0.8400 ± 0.1074	0.7316 ± 0.1390
LDA	0.8786 ± 0.0643	0.8927 ± 0.0664	0.7417 ± 0.1490	0.8616 ± 0.1097	0.7632 ± 0.1391

Table A.12: PCA Components and Classifier Parameters for Different Models

Model	PCA N components	Parameters classifier
LR	8	{'max_iter': 10000, 'penalty': 'l1', 'solver': 'saga'}
DTree	2	{'max_depth': 1, 'min_samples_leaf': 1}
KN	2	{'n_neighbors': 9, 'weights': 'uniform'}
GNB	3	{'var_smoothing': 1.0}
SVC	14	{'C': 0.1, 'degree': 1, 'kernel': 'linear'}
LDA	8	{'solver': 'svd'}



(a) Logistic Regression **(b)** Decision Tree **(c)** K Nearest Neighbours

(d) Gaussian Naive Bayes **(e)** Support Vector Machine **(f)** Linear Discriminant Analysis

Figure A.15: Accuracy for different number of components used for PCA, for different classifiers

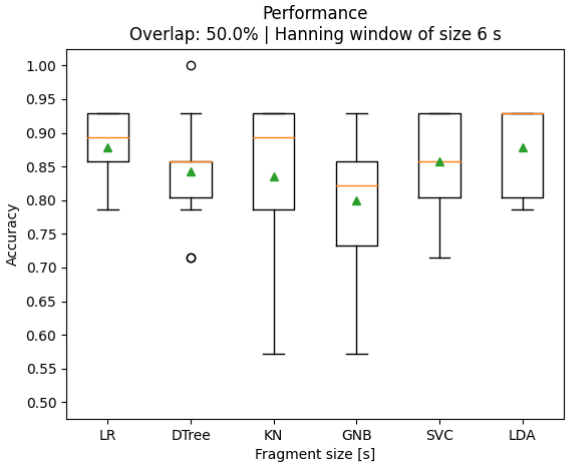


Figure A.16: Accuracy for different classifiers

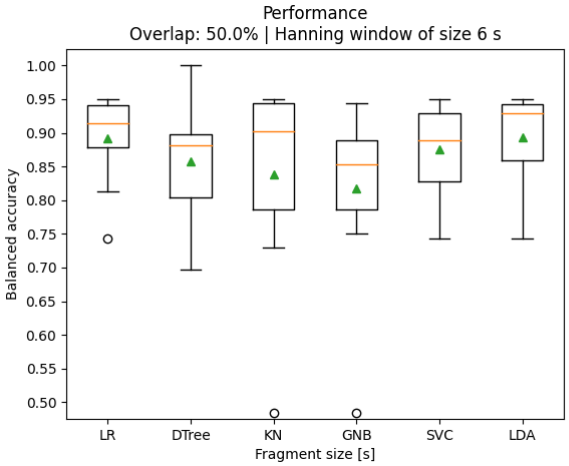


Figure A.17: Balanced accuracy for different classifiers

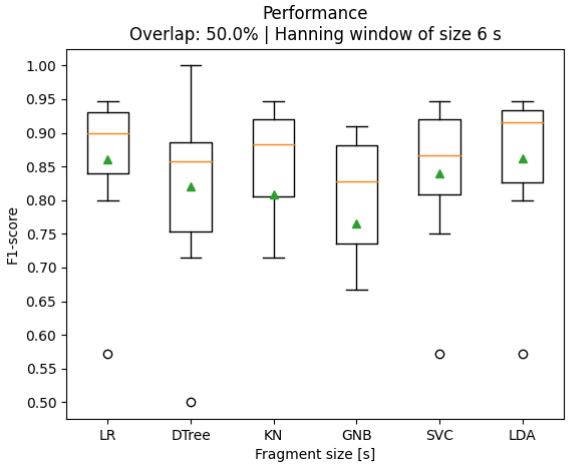


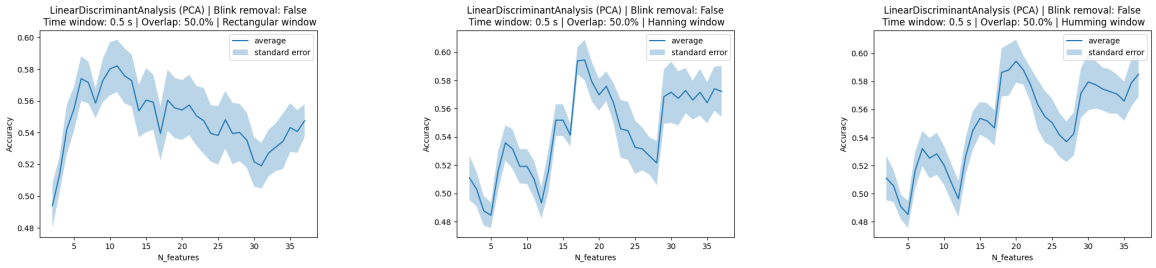
Figure A.18: F1-score for different classifiers

A.2. Own OpenBCI dataset

A.2.1. Raw

Table A.13: Performance measures for a window size of 0.5 seconds

	Rectangular Window	Hanning Window	Hamming Window
Accuracy	0.4722 ± 0.1912	0.5111 ± 0.1401	0.5056 ± 0.1124
Balanced Accuracy	0.5167 ± 0.1920	0.5583 ± 0.1410	0.5583 ± 0.1137
Kappa	0.0370 ± 0.3273	0.0989 ± 0.2246	0.0978 ± 0.1863
F1-Score	0.4747 ± 0.1901	0.5161 ± 0.1360	0.5026 ± 0.1014
Matthew Correlation Coef	0.0227 ± 0.3726	0.1094 ± 0.2904	0.1122 ± 0.2274

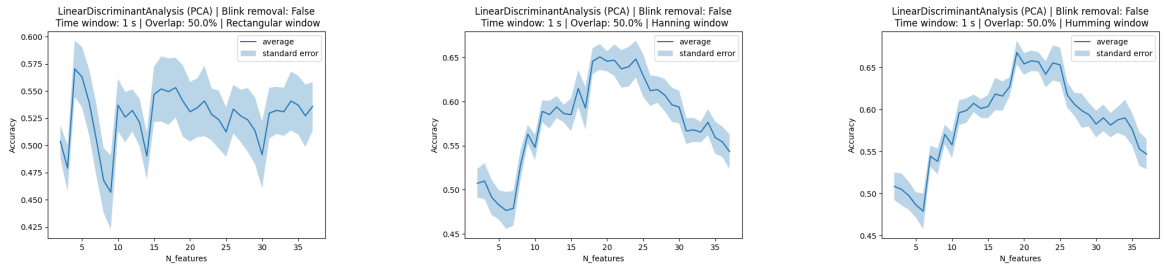


(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.19: Accuracy for different number of components used for PCA, on a time window of 0.5 seconds

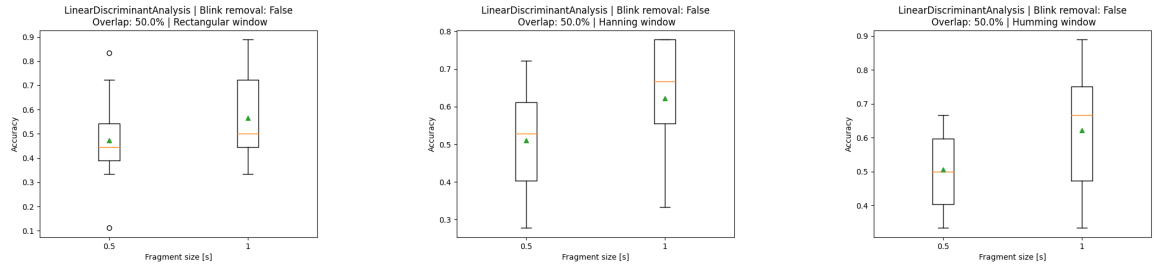
Table A.14: Performance measures for a window size of 1 second

	Rectangular Window	Hanning Window	Hamming Window
Accuracy	0.5667 ± 0.2015	0.6222 ± 0.1663	0.6222 ± 0.1937
Balanced Accuracy	0.5917 ± 0.2022	0.6417 ± 0.1446	0.6667 ± 0.1900
Kappa	0.1621 ± 0.3922	0.2690 ± 0.2667	0.2965 ± 0.3377
F1-Score	0.6038 ± 0.2075	0.6038 ± 0.1493	0.6070 ± 0.1743
Matthew Correlation Coef	0.1898 ± 0.4165	0.2847 ± 0.2903	0.3219 ± 0.3664



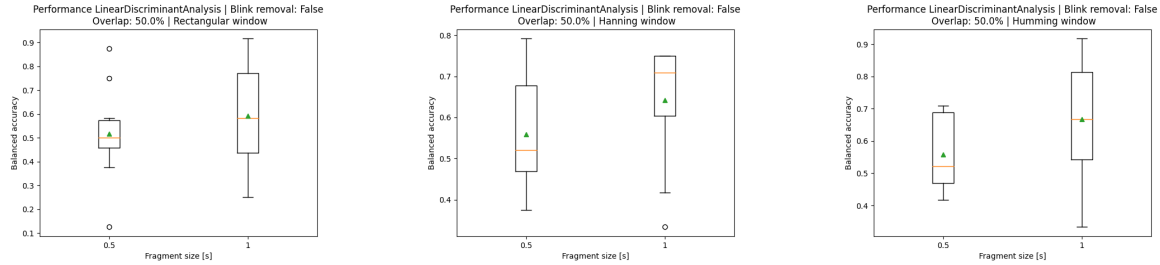
(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.20: Accuracy for different number of components used for PCA, on a time window of second



(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.21: Accuracy for different window sizes



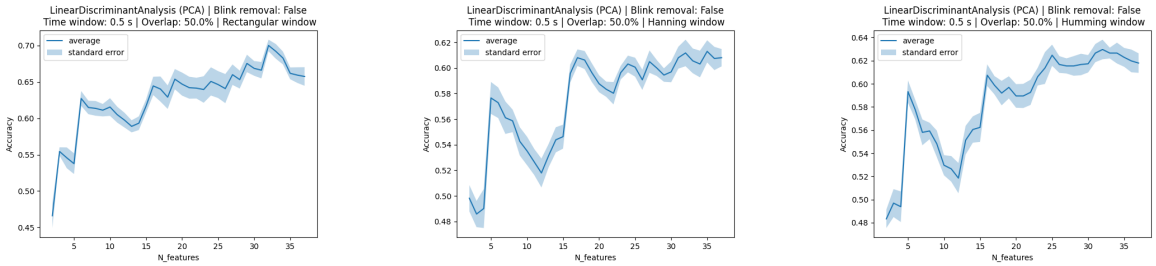
(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.22: Balanced accuracy for different window sizes

A.2.2. Filtered

Table A.15: Performance measures for a window size of 0.5 seconds

	Rectangular Window	Hanning Window	Hamming Window
Accuracy	0.6556 ± 0.1423	0.5778 ± 0.0793	0.6111 ± 0.0930
Balanced Accuracy	0.6625 ± 0.1425	0.6083 ± 0.1041	0.6292 ± 0.1061
Kappa	0.3017 ± 0.2707	0.1802 ± 0.1731	0.2269 ± 0.1848
F1-Score	0.6285 ± 0.1910	0.5376 ± 0.1311	0.5909 ± 0.1469
Matthew Correlation Coef	0.3184 ± 0.2769	0.2123 ± 0.2055	0.2478 ± 0.2034

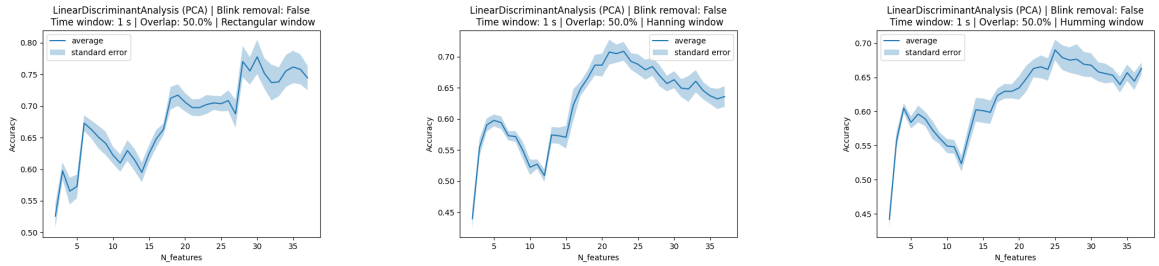


(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.23: Accuracy for different number of components used for PCA, on a time window of 0.5 seconds

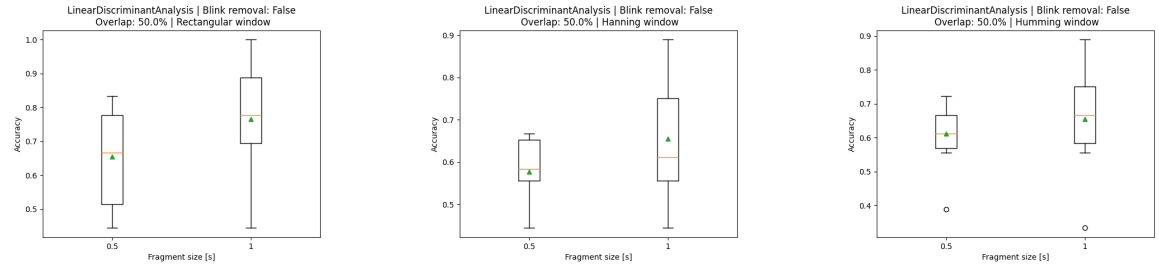
Table A.16: Performance measures for a window size of 1 second

	Rectangular Window	Hanning Window	Hamming Window
Accuracy	0.7667 ± 0.1606	0.6556 ± 0.1444	0.6556 ± 0.1444
Balanced Accuracy	0.7417 ± 0.1766	0.6917 ± 0.1446	0.6750 ± 0.1417
Kappa	0.4803 ± 0.3627	0.3345 ± 0.2648	0.3107 ± 0.2660
F1-Score	0.7104 ± 0.2735	0.6405 ± 0.1828	0.6075 ± 0.2270
Matthew Correlation Coef	0.5083 ± 0.3678	0.3847 ± 0.2667	0.3280 ± 0.2893



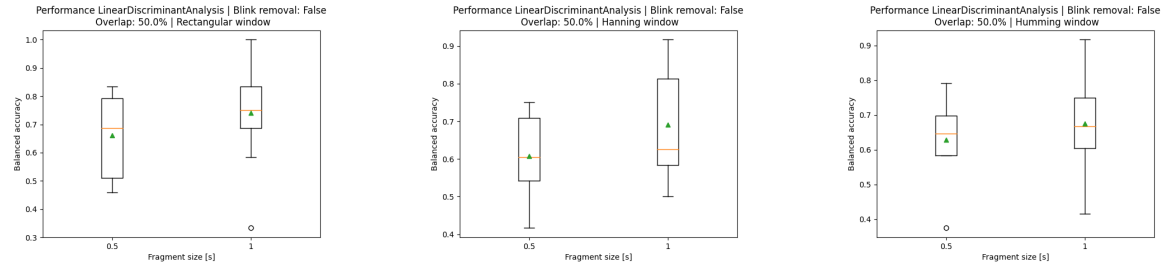
(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.24: Accuracy for different number of components used for PCA, on a time window of 1 second



(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.25: Accuracy for different window sizes



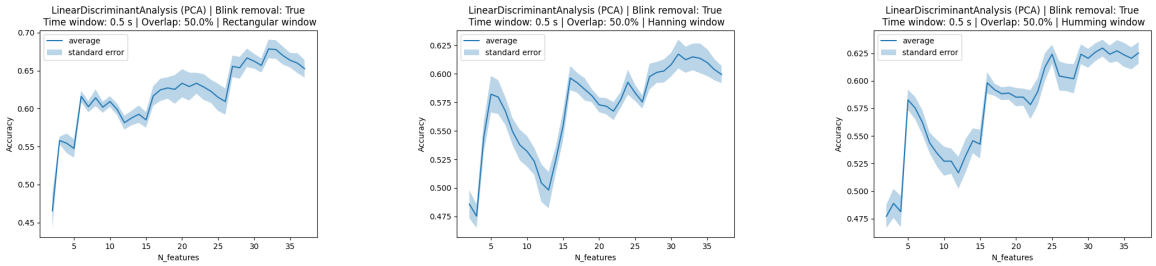
(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.26: Balanced accuracy for different window sizes

A.2.3. Filtered and blink removal

Table A.17: Performance measures for a window size of 0.5 seconds

	Rectangular Window	Hanning Window	Hamming Window
Accuracy	0.6667 ± 0.1427	0.5722 ± 0.0705	0.6000 ± 0.0923
Balanced Accuracy	0.6708 ± 0.1412	0.6000 ± 0.0878	0.6292 ± 0.1028
Kappa	0.3221 ± 0.2776	0.1688 ± 0.1443	0.2225 ± 0.1761
F1-Score	0.6402 ± 0.1888	0.5352 ± 0.1170	0.5762 ± 0.1392
Matthew Correlation Coef	0.3382 ± 0.2814	0.1936 ± 0.1715	0.2529 ± 0.2018

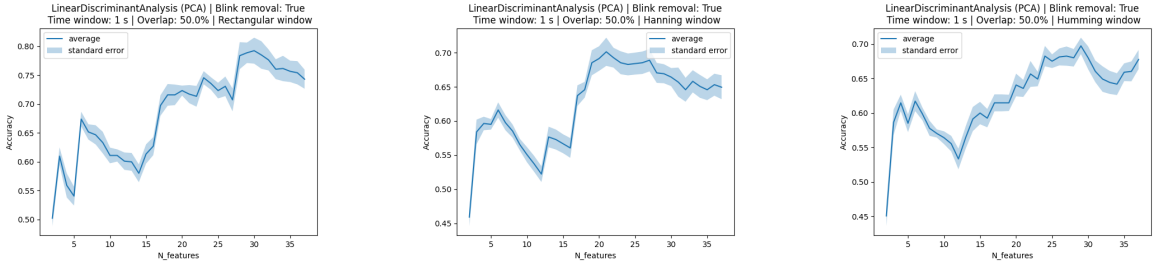


(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.27: Accuracy for different number of components used for PCA, on a time window of 0.5 seconds

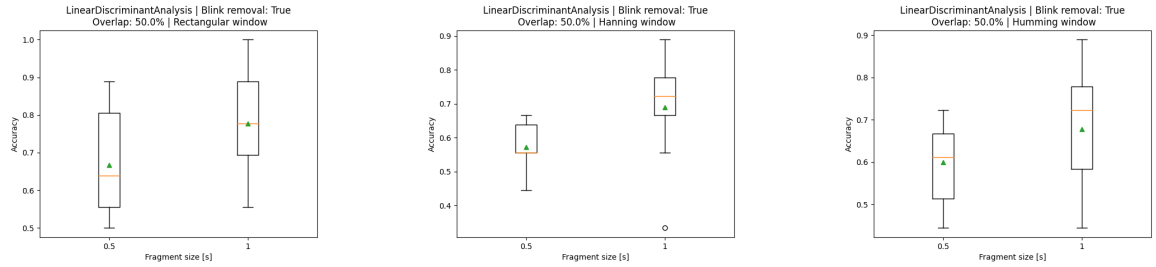
Table A.18: Performance measures for a window size of 1 second

	Rectangular Window	Hanning Window	Hamming Window
Accuracy	0.7778 ± 0.1405	0.6889 ± 0.1474	0.6778 ± 0.1444
Balanced Accuracy	0.7583 ± 0.1685	0.7250 ± 0.1346	0.7083 ± 0.1502
Kappa	0.4988 ± 0.3341	0.3695 ± 0.2705	0.3695 ± 0.2705
F1-Score	0.7085 ± 0.2810	0.6405 ± 0.1828	0.6662 ± 0.1354
Matthew Correlation Coef	0.5194 ± 0.3480	0.3847 ± 0.2667	0.4040 ± 0.2847



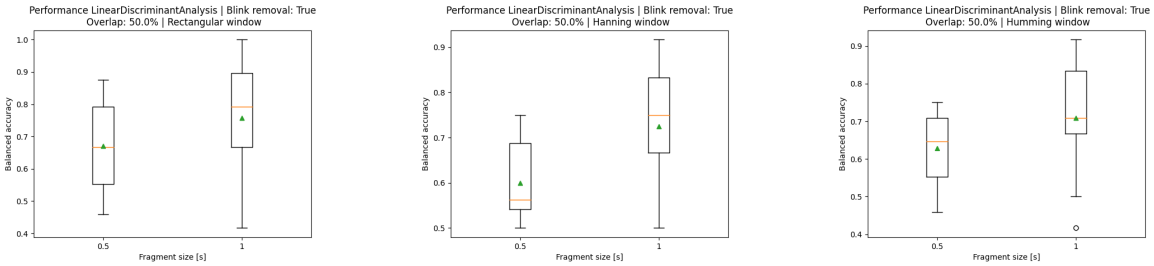
(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.28: Accuracy for different number of components used for PCA, on a time window of 1 second



(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.29: Accuracy for different window sizes



(a) Rectangular window **(b)** Hanning window **(c)** Hamming window

Figure A.30: Balanced accuracy for different window sizes

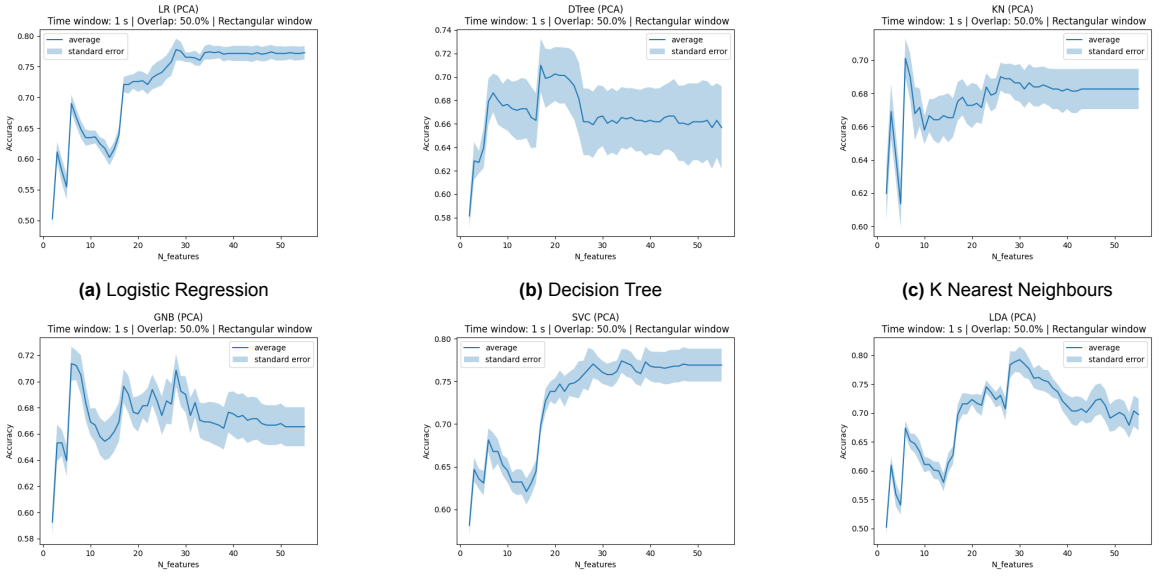
A.2.4. Comparing classifiers

Table A.19: Performance measures for a window size of 1 seconds with different classifiers

	Accuracy	Balanced Accuracy	Kappa	F1-Score	Matthew Correlation Coef
LR	0.7444 ± 0.1575	0.7500 ± 0.1394	0.4839 ± 0.2791	0.7344 ± 0.1682	0.5205 ± 0.2762
DTree	0.6444 ± 0.2037	0.6917 ± 0.1828	0.3491 ± 0.3517	0.5873 ± 0.2500	0.3959 ± 0.3465
KN	0.5556 ± 0.1648	0.5833 ± 0.1394	0.1548 ± 0.2593	0.5495 ± 0.1833	0.1668 ± 0.2730
GNB	0.6556 ± 0.1528	0.6833 ± 0.1572	0.3217 ± 0.2944	0.6552 ± 0.1276	0.3567 ± 0.3339
SVC	0.7333 ± 0.2000	0.7000 ± 0.2179	0.3980 ± 0.4426	0.7372 ± 0.2107	0.4052 ± 0.4570
LDA	0.7778 ± 0.1405	0.7583 ± 0.1685	0.4988 ± 0.3341	0.7085 ± 0.2810	0.5194 ± 0.3480

Table A.20: PCA Components and Classifier Parameters for Different Models

Model	PCA N components	Parameters classifier
LR	28	{'max_iter': 10000, 'penalty': None, 'solver': 'saga'}
DTree	17	{'max_depth': 1, 'min_samples_leaf': 1}
KN	6	{'n_neighbors': 5, 'weights': 'uniform'}
GNB	6	{'var_smoothing': 1e-06}
SVC	34	{'C': 1.0, 'degree': 1, 'kernel': 'linear'}
LDA	30	{'solver': 'svd'}



(a) Logistic Regression **(b)** Decision Tree **(c)** K Nearest Neighbours

(d) Gaussian Naive Bayes **(e)** Support Vector Machine **(f)** Linear Discriminant Analysis

Figure A.31: Accuracy for different number of components used for PCA, for different classifiers

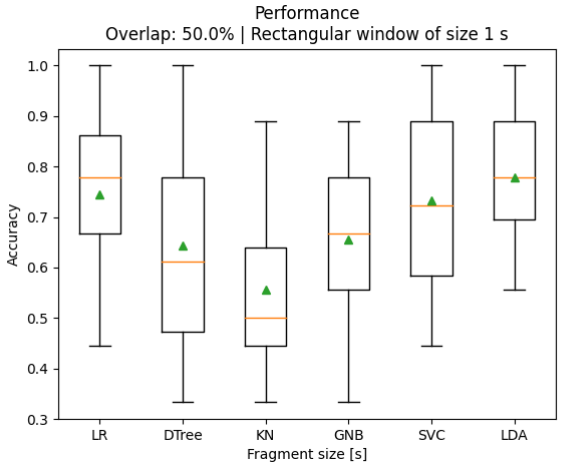


Figure A.32: Accuracy for different classifiers

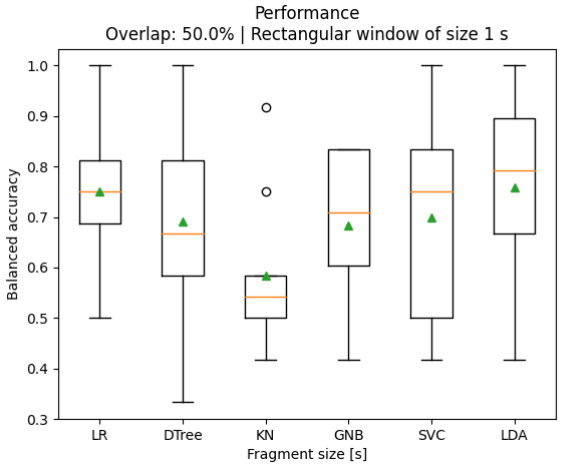


Figure A.33: Balanced accuracy for different classifiers

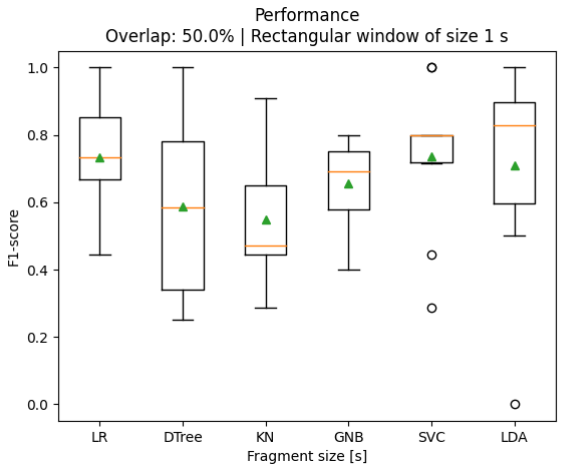
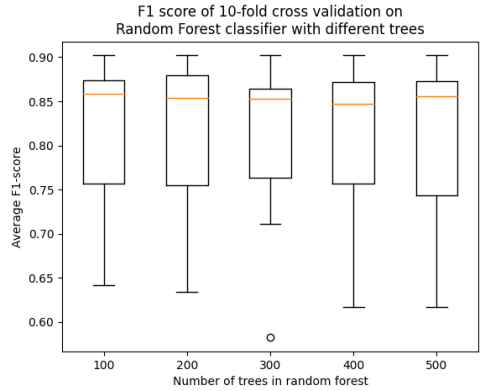
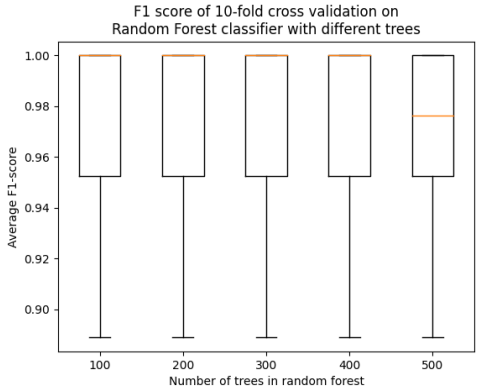


Figure A.34: F1-score for different classifiers

A.3. Different trees relevance in for random forest



(a) EEG-IO average F-1 score in 10 fold cross validation for different trees in a window size of 250 samples

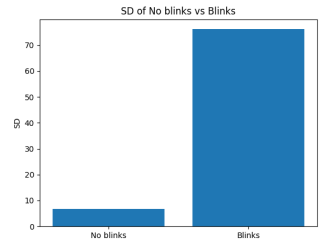


(b) Own dataset B average F-1 score in 10 fold cross validation for different trees in a window size of 250 samples

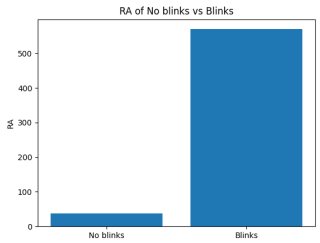
Figure A.35: Comparison different trees for F1-score

A.4. Feature relevance for random forest

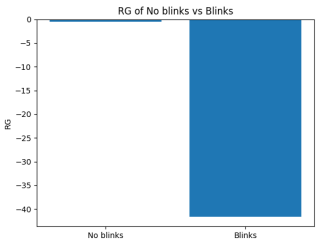
In Figure A.36 the average value for each feature is shown in a bar plot for blinking and no blinking with Own dataset B. The features are averaged over 100 trials of EEG with blinks and 100 trials of EEG with no blinks. A clear difference can be seen between the value of the feature of no blinking and blinking. This shows that these features are relevant for classification with random forest.



(a) Bar plot of average standard deviation between EEG with no blinks and blinks



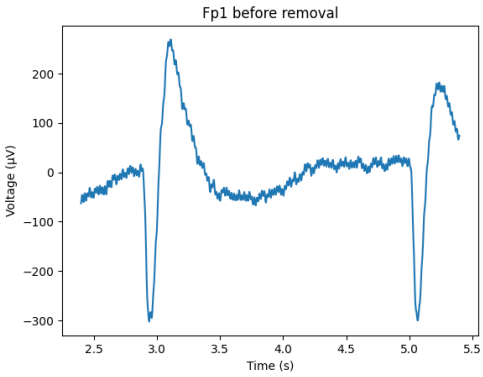
(b) Bar plot of average range of amplitude between EEG with no blinks and blinks



(c) Bar plot of average range of grade between EEG with no blinks and blinks

Figure A.36: Bar plots of the average value of each feature for EEG with no blinks and blinks.

A.5. Additional Figures blink removals



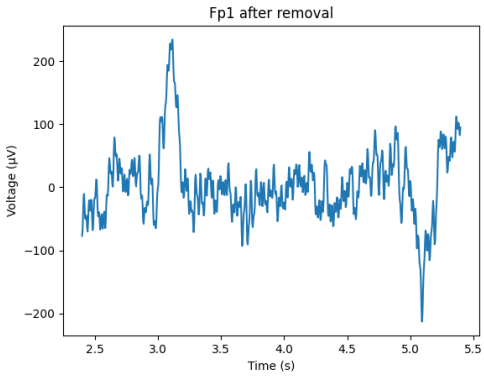


Figure A.37: Blink removal

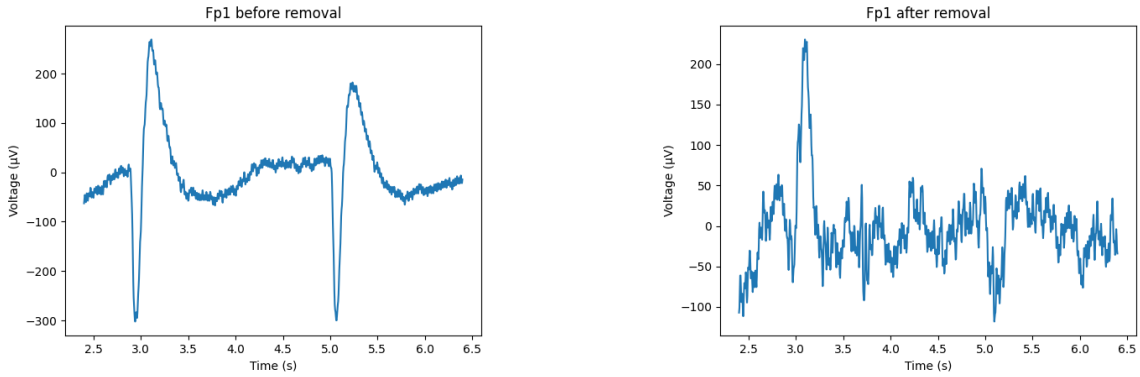


Figure A.38: Blink removal

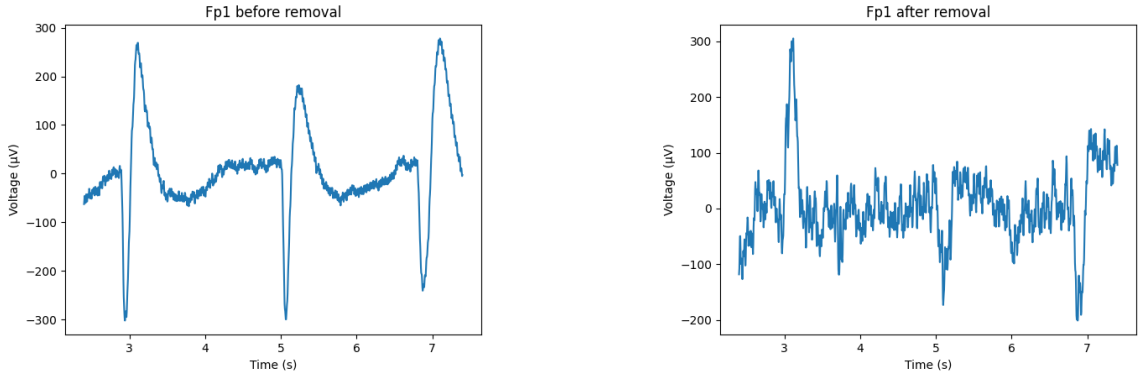


Figure A.39: Blink removal

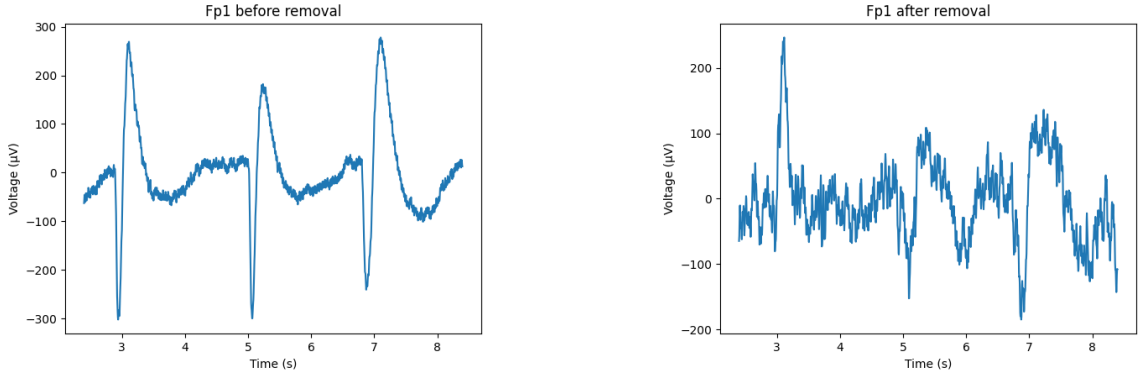


Figure A.40: Blink removal

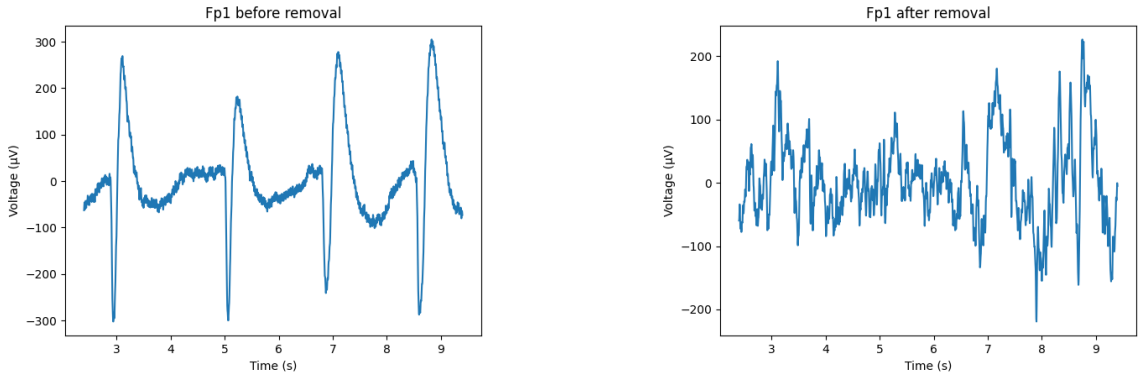


Figure A.41: Blink removal

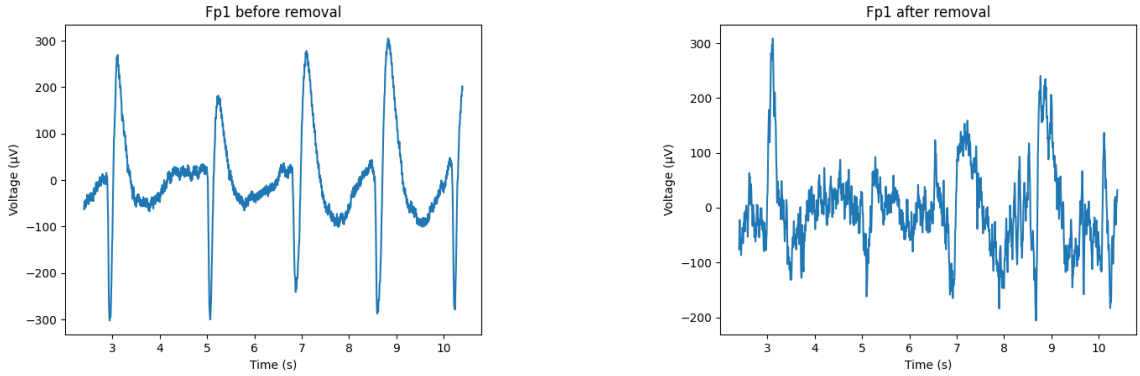


Figure A.42: Blink removal