ÉCOLE POLYTECHNIQUE DE LOUVAIN

# EEG classification Eyes open and closed

*LGBIO2020 – Bioinstrumentation – Group 14*

*Project done by Tomás dos Santos Talento and Marc Phillipe Marburger*

## 1. Introduction

Classifying EEG signals into "eyes open" and "eyes closed" states is a good introduction to EEG processing. In this project, we received two datasets with different lengths one recorded during eyes open and one recorded during eyes closed. In this report we describe the development and testing of several ML models including the basic data preprocessing, to have the best possible predictions.

## 2. Decisions

Here we describe the decisions of the project for each part of the machine learning pipeline. The data processing is handled separately for the two datasets (classes) until segmenting them into epochs of 500ms. These epochs of equal length are used as samples for the classification. We binarized our epoch data so that label “1” means eyes closed and that label “0” means eyes open.

### 2.1 Preprocessing

The raw signal preprocessing involves noise removal through filtering, narrowing the frequency spectrum to the desired range. The MNE Python package plots the spectrum of raw EEG data, showing influence from powerline noise around 50Hz seen in figure 1. EEG signals typically range from 0.5 to 100Hz, corresponding to delta, theta, alpha, beta, and gamma brain wave bands. We use a bandpass filter from 0.5Hz to 40Hz to focus on these bands, disregarding frequencies above 40Hz, which includes part of the gamma band (not keeping the gamma band in its entirety, in this case, is acceptable since it won’t be a very discriminative feature between the eyes open and eyes closed EEGs). A 4th-order Butterworth IIR filter is chosen to avoid ringing near window edges, though with a wider transition region. The filter is applied bidirectionally to prevent phase shifts. Figure 2 in the appendix displays the filtered signals, appearing more harmonious with decreased amplitude in higher frequencies due to chosen cutoffs. The bandpass filtering removed the low-frequency noise and high-frequency noise from the data, but it still contains a 50 Hz peak due to the power line noise also seen in the appendix. This is removed by applying a notch filter with a cut-off at 50Hz and the higher harmonic at 100hz. The result of this operation is shown in figure 3 in the appendix.

### 2.2 Z-score normalization

Z-score normalization, also known as standardization, is used to rescale the dataset so that it has a mean of zero and a standard deviation of one. By subtracting each channel by its mean, and then dividing by the standard deviation of that channel, the distribution of the dataset is centered around zero and has a spread that is measured in units of standard deviations. EEG signals recorded from different channels may have different scales and variances due to factors such as electrode placement and contact quality. This is easily shown on figure 4 in the appendix. Normalizing each channel independently ensures that all channels contribute equally to the analysis, regardless of their individual characteristics.

### 2.3 Segmentation

We decided to have a 50% overlap between segments of length 500ms. Each segment is considered an epoch. As a first argument, having overlap allows for more samples, which is an advantage when using machine learning, since the more samples, the less overfitting there will be. With this approach, the same part of the signal is used multiple times in different segments, which can lead to more robust

results, especially when the amount of data is limited, which is our case in this project with EEG time-series of around 120 seconds. Thus, introducing overlap helps maintain temporal continuity while also reducing edge effects. The overlapping can help capture the temporal dependencies between successive segments.

## 2.4 Feature space

Features are extracted from segmented EEG data across different channels. The process begins by iterating through each channel in the dataset. For each channel, statistical features such as mean, standard deviation, kurtosis, and peak-to-peak amplitude are calculated across each segment of the EEG signal. These features provide insights into the statistical properties and amplitude characteristics of the signal. Additionally, the Power Spectral Density (PSD) is computed using the getBandPower() function provided. Based on our definitions of frequency bands (Delta, Theta, Alpha, Beta, Gamma) the power is calculated for each segment. These can be seen in figure 5 in the appendix. Ratios between these band powers are also calculated to capture the relative power distribution across frequency bands. The ratios and their distribution can be seen in the appendix in figure 6. In total, per each of the 64 channels, 14 features are computed, amounting to a total of 896 features.

The computed features, along with the corresponding labels, are organized into a DataFrame. Each row of the DataFrame represents a segment of the EEG signal, while columns represent different features extracted from the signal. The final dataset consists of 1030 segments of 512ms length (rows) and 896 features (columns).

## 2.5 Principal Component Analysis

Having 897 features is a rich set of information, which can be used to train machine learning models to classify or predict various conditions or outcomes based on the EEG signals. However, it's also essential to consider the dimensionality of the dataset and the potential for overfitting when working with many features, on top of the extra training time. Therefore, we decided to use PCA (Principal Component Analysis). PCA is a variance-maximizing method, it projects the original data onto directions that maximize the variance. If any feature of our feature space has a broad range of values, the variance will be dominated by these features (scale sensitive). Therefore, the range of all features should be normalized so that each feature contributes equally to the final variance. Z-score normalization is done before applying PCA (subtracting the mean and dividing by the standard deviation). When we used Principal Component Analysis (PCA) on the data, we found that 198 components were needed to explain 95% of the variance in the data. This means that even though the data is complex (a lot of features), we can understand most of its distribution by looking at just these 198 components. So, by using these components, we can simplify the data while keeping most of the important information. The graph for the cumulative explained variance ratio used to determine the number of components explaining 95% of the variance can be seen in the appendix in figure 7. To get a grasp of the distribution of the data, we decided to plot it into its first three principal components which is seen in appendix figure 8. Even though this does not paint the full picture of the data distribution, it still allows us to draw some conclusions on what machine learning models should work well with the data in hand. For instance, by looking at the graph we can see that globally the two classes have different distributions. However, they do not appear to be linearly separable (there is some overlap), or at least not in the first three principal components, which makes it unlikely that it is in the rest of the components since these two already explain about 35.5% of the variance in the data. This allows us to conclude that linear discriminant models will not provide good results and we should thus use non-linear models.

## 2.5 Model Selection

We want to evaluate both advanced and more simple models for this classification task, to benchmark algorithm types. In this section, the hyperparameter search and the various models are explained. Hyperparameter search was done by manual grid searching for each model. This was an acceptable approach since all the models were so quick to train, even the neural network. Even with slightly randomly chosen parameters, the accuracies were above 70%.

**Neural network using TensorFlow Keras**

We built a sequential Neural network with 67457 trainable parameters, that has 256 units in the first dense layer, 64 units in the second, and 1 unit in the output dense layer. The model is trained using a learning rate scheduler. It decreases the learning rate exponentially over time at a rate of 0.2 within 40 epochs (one epoch equals one full pass on the training set) of training. This allows the model to gradually refine its parameters as training progresses and converge to a better solution while converging faster in the beginning of training. Having a decaying learning rate allowed us to have fewer training epochs for the same performance which helped speed up training. Model fitting was done with the Adam optimizer, with Binary Cross-Entropy as the loss function. The loss function and optimizer were chosen based on intuition and what we previously found working well. The number of units in each dense layer, the learning rate, and the number of epochs were done iteratively. The batch size was left at the default 32 samples, and the kernels and biases were initialized using a random normal distribution.

**Gradient boosting ensemble classifier**

This was our second most advanced model, where we found the best combination to be a learning rate of 0.1, with a max depth of 6 for each estimator. The number of max features was found to be best when using the square root of the number of features. The minimum samples per leaf was found to be best at 4, and the minimum samples for splitting was 3. Because of these two previous values, we introduce pruning to the trees, avoiding overfitting when training as the trees would keep splitting in branches until the train accuracy was perfect. The number of estimators in the ensemble was found to have good performance with 500 estimators, while still training relatively fast.

**K-Nearest Neighbor classifier**

KNN is often used as a benchmark in classification tasks. Therefore, we decided to test it as well in our data. We found that using the ‘Euclidean’ metric for distance computation with 20 neighbors and setting the weight function to ‘distance’, gave the best performance. This approach weights samples by the inverse of their distance, meaning that closer neighbors will have a greater influence than neighbors far away. Regarding overfitting, we used a relatively large number of neighbors to try to avoid it.

**Decision Tree**

The best decision tree utilized the criterion 'log_loss', which is the same as cross-entropy loss, common for binary classification models. The best tree had minimum samples for splitting set to 4, with a minimum samples per leaf to 1, with a max depth of 16. Having the minimum samples to split at 4 introduced pruning, which combats overfitting.

**Support Vector Machine**

As we have mentioned, linear discriminant models may not be suitable for this task as the data does not seem to be easily linearly separable. SVMs make use of Kernels to extend linear discriminant

models to have a non-linear mapping of the feature space. Setting the regularization parameter C to 0.2 gave good results. We found the best kernel to be Radial Basis Function (RBF). It non-linearly transforms the data into a higher dimensional space where it is easier to find a hyperplane that better separates the classes. Gamma is the parameter for the kernel which is inversely proportional to the spread of the kernel, here we set it to 'scale', which showed the best performance. Both the gamma and the C parameters are introduced to avoid overfitting: the lower the value of C the more regularization, and the lower the gamma the higher the spread of the RBF kernel, both decreasing overfitting.

## 2.6 Results on validation set

We initially split the dataset into two parts using an 80/20 ratio, where 80% of the full dataset is designated as the training set and the remaining 20% as the test set. This approach allows us to train the model on a larger portion of the data while still reserving a separate portion for evaluation to assess the model's generalization performance. Within the training set, we further applied K-fold cross-validation with K=10. This involves dividing the training set into K subsets or folds and iteratively training the model K times. During each iteration, the model is trained on K-1 folds and evaluated on the remaining fold. By averaging the performance metrics, such as accuracy, over all K iterations, we obtain a more reliable estimate of the model's performance on unseen data.

After completing K-fold cross-validation and training the model on the entire training set, we evaluate the model's performance on the separate test set. This test set was not used during model training or cross-validation, serving as a completely unseen dataset for evaluating the model's performance. We expect the estimated accuracy from K-fold cross-validation on the training set to be very similar to the accuracy of the model evaluated on the test set. This similarity indicates that the model has not overfit the training data and generalizes well to unseen data. Any significant inconsistency between the estimated accuracy and test accuracy may suggest issues such as overfitting. The results are shown and compared in Table 1 below.

*Table 1: Model accuracies during both K-Fold training and testing on the validation set.*

| Model | Estimated Accuracy (training K-fold) | Test accuracy (test set) |
|---|---|---|
| Neural Network | 0.9769 | 0.9951 |
| K-Nearest Neighbor | 0.9017 | 0.9515 |
| Support Vector Machine | 0.9090 | 0.9272 |
| Gradient Boost | 0.9223 | 0.9563 |
| Decision Tree | 0.8240 | 0.8544 |

As seen in the table 1 above, the estimated accuracy is identical to the accuracy obtained on the test set. However, a small difference can be noted, with the test accuracy always exceeding the estimated k-fold cross-validation. This may be because the test accuracy refers to the model trained on the 80%, whereas the estimated accuracy refers to the model trained with 90% out of the 80% training data. This means that the neural network is trained on fewer samples for the latter, maybe impacting the accuracy of the model. One other factor worth mentioning is that models with accuracy close to 100%, when doing k-Fold cross-validation accuracy can only go up to 100%. Given the small size of a single fold as the test set, missing just one sample, will already decrease the accuracy by 1.2%.

Overall, the accuracies are very good, implying good data preprocessing and feature generation, as well as evidencing the success of the model in explaining this data.

# 3. Conclusion

Based on the validation results, it is evident that the estimated accuracies from the K-fold cross-validation closely align with the actual test accuracies on the independent test set, meaning that we didn't introduce overfitting of our models. Among the models tested, the neural network exhibited the highest estimated and test accuracies. It is worth noting less complex models like the gradient boost and k-nearest neighbor models demonstrated very competitive performance, although slightly lower. On the other hand, decision tree models displayed comparatively lower accuracies, indicating potential limitations in capturing the underlying patterns within the dataset. Overall, these results provide valuable insights into the relative strengths of the chosen machine learning models for the given task and dataset, showing that the faster and smaller models are almost as good for eyes-open and eyes-closed classification tasks as a neural network.

# 4. Limitations and future improvements

In order to enhance the development of better eyes-open and eyes-closed classifiers in the future, we have looked into the following future improvements.

## 4.1 EEG channel selection

Two channels (T7 and P1) show flatline across most of the recording. Between 32 and 35 seconds in eyes open, the voltage jumps to very high amplitudes, disproportionally to the rest of the channels, like an electrode pop. These channels should have been dropped, which is easily seen in the appendix in figure 9. When removing contaminated channels, it might also be worth removing channels with high power in high frequencies (50 to 150 Hz), since this activity is mostly of muscular origin.

Furthermore, some channels are more relevant for this classification task, considering the brain areas involved in visual processing. Indeed, visual processing involves several brain regions, including for example the occipital lobe, which is primarily responsible for visual perception. Channels located over the occipital region may capture neural activity that is relevant to the task of distinguishing between the eyes open and closed states. Additionally, EEG channels positioned near regions involved in attention and arousal, such as the parietal and frontal lobes, may also contribute valuable information.

## 4.2 No muscle artefact detection and removal

We did not introduce methods to remove other types of artefacts than noise filtering. Artefacts in the dataset might have been introduced due to the subject either speaking or laughing during the recording. The possible activation of facial muscles or the movement of the EEG-cap happens about 38 seconds into the eyes closed dataset, where huge fluctuations are seen across the different channels. These abnormalities do not reflect the patterns seen throughout the rest of the eyes-closed dataset. The clear interrupted signal can be seen in the screenshot in figure 10 in the appendix.

Artefact elimination and removal could be done using independent component analysis to separate EEG signals from artifacts, and then do interpolation, to replace the artifact-contaminated segments with estimates based on neighboring data. In cases where artifacts affect entire segments of data, one should consider excluding those segments from the analysis altogether.

## 4.3 Eye blink artefacts

It might be relevant to look for eye blinks as a feature for the eyes open dataset. It could be a smart feature, as the subject won't blink in eyes closed state. Eye blinks should produce distinct muscle patterns in the EEG signals due to the electrical activity associated with blinking muscles followed by some neural responses in the visual cortex, where the eyes have "no" input for some milliseconds. This feature might have an impact on classification accuracies if implemented correctly.

# Appendix

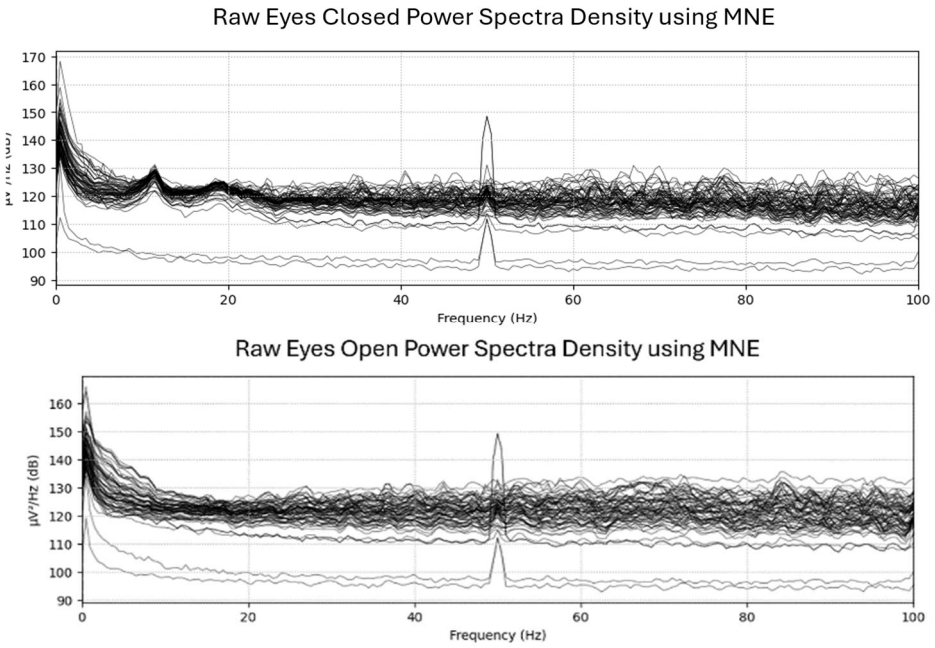


*Figure 1 Power Spectra Density plots using MNE for both datasets. Clearly visible powerline noice at 50Hz.*

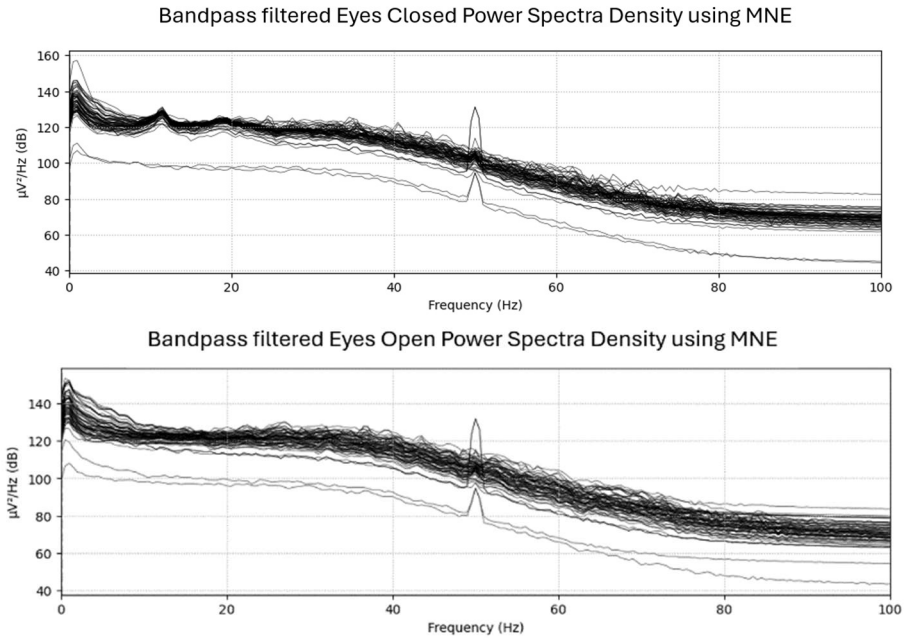


*Figure 2 Power Spectra Density plots using MNE for both datasets after bandpass filtering.*

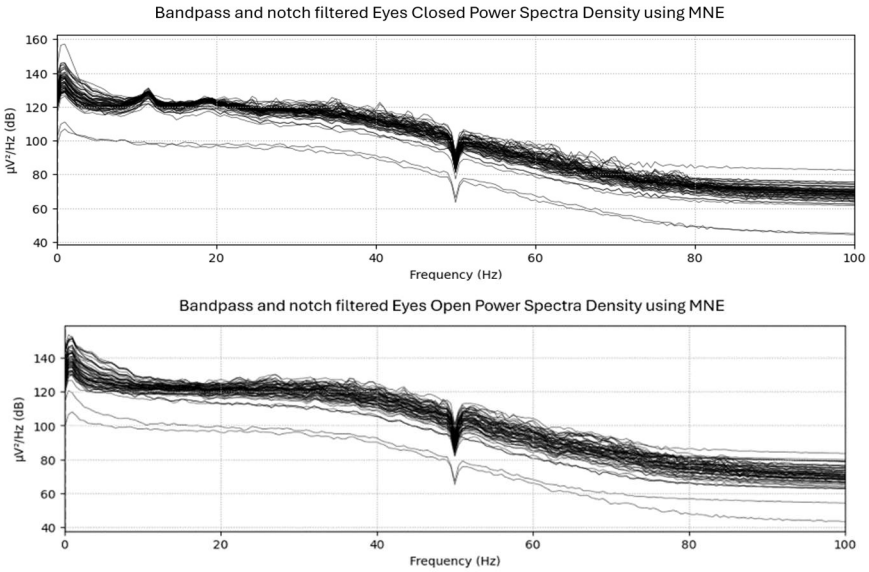


*Figure 3 Power Spectra Density plots using MNE for both datasets after notch and bandpass filtering. The notch width is set to 0.5, filter length set to 'auto', phase mode as 'zero', fir window using 'hann', and fir design employing 'firwin'.*

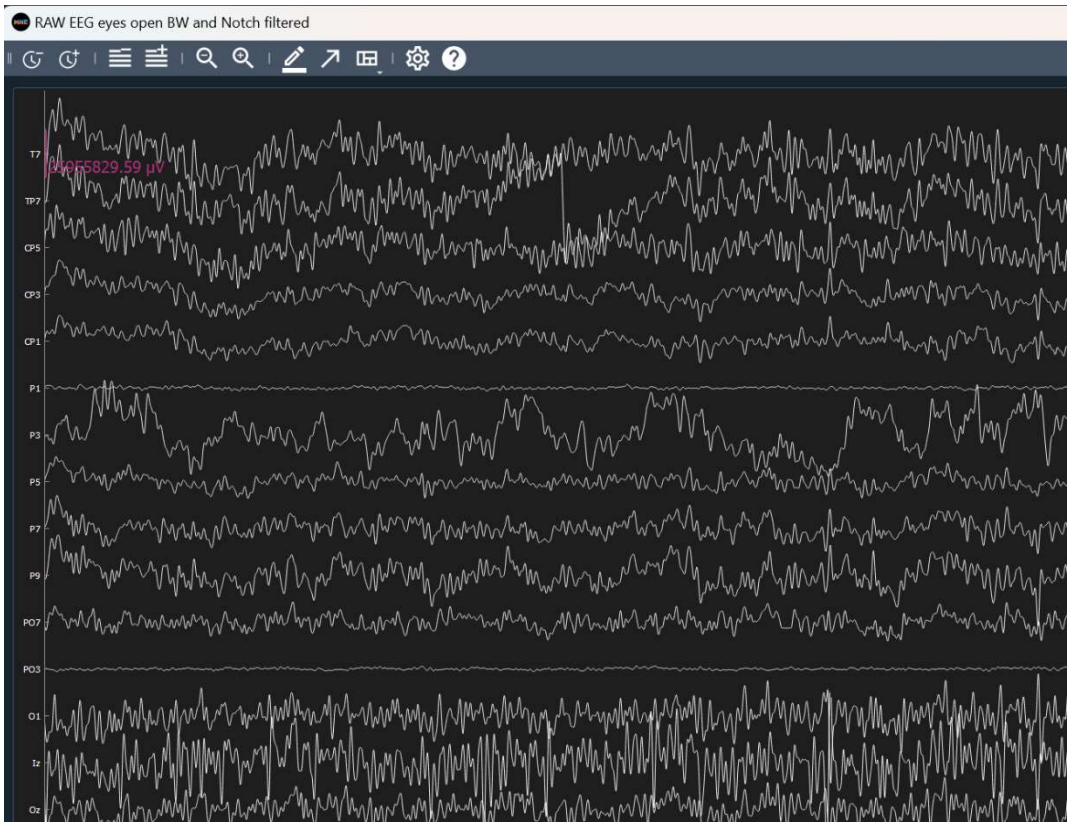


*Figure 4 Screenshot of MNE Python library showing two odd electrode channels P1 and PO3. This proves the normalization step necessary, as these channels seems dead compared to the rest. But it is simply due to a magnitude difference.*

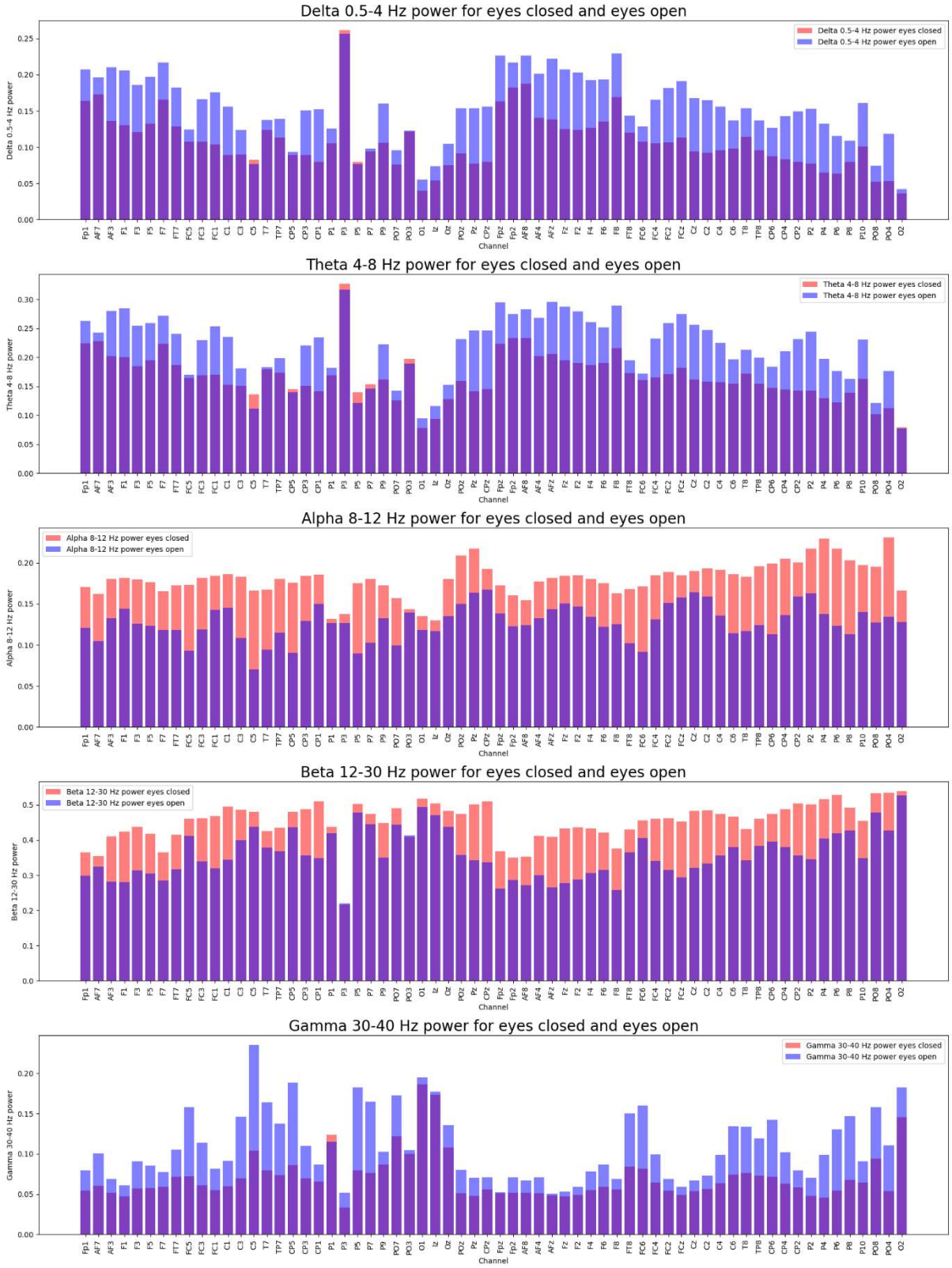


*Figure 5 Band power across each brain wave frequency band for all 64 channels for both eyes closed and eyes open. Overlapping bars are shown in purple, as red + blue gives purple.*

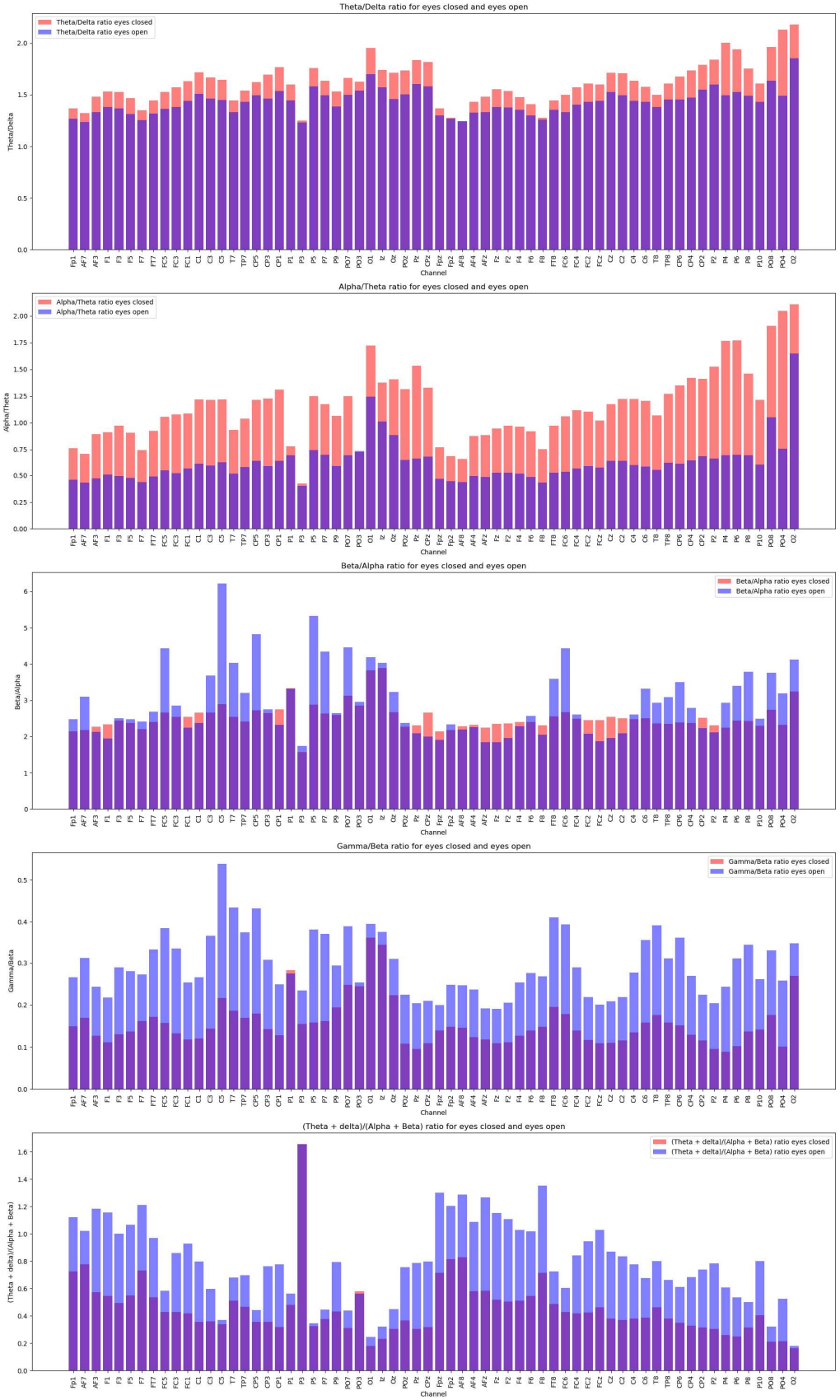


*Figure 6 Ratios between band powers across brain wave frequency bands. These features are inspired by Danish lector Kaare Mikkelsen, Institute of Electrical and Computer Engineering, Aarhus university.*

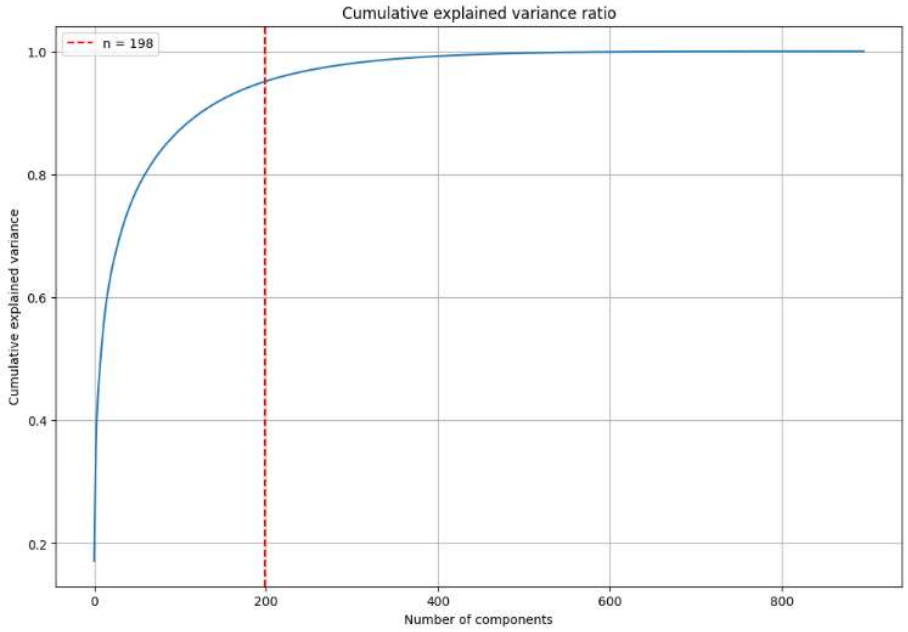


*Figure 7 Cumulative explained variance ratio used to determine the number of components explaining 95% of the variance which amounts to 198 features/components.*

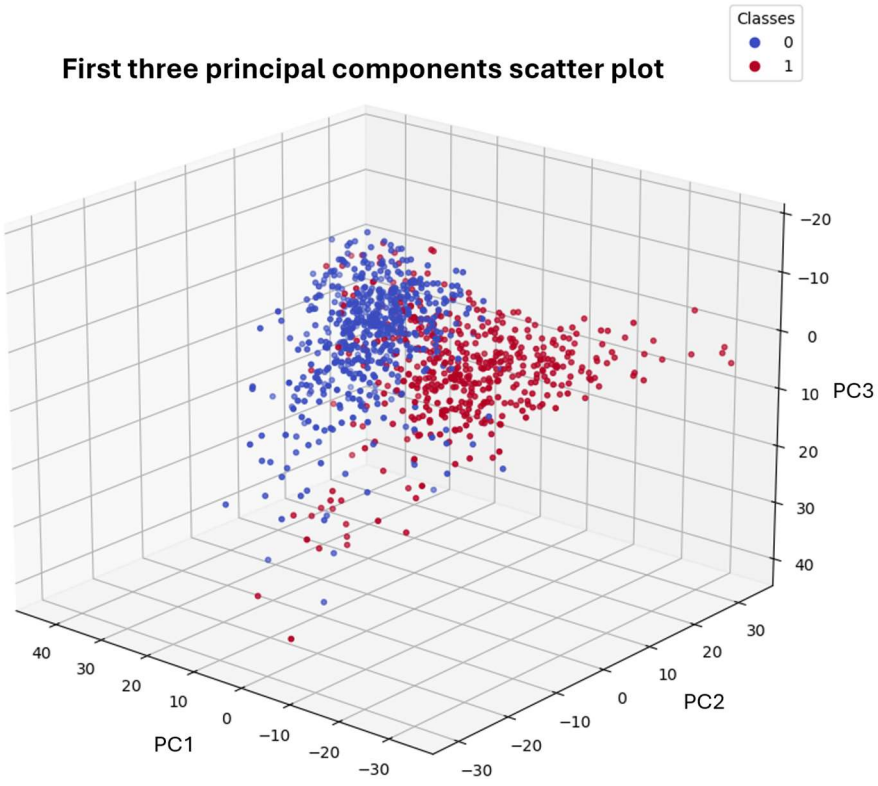


*Figure 8 Scatterplot in 3D with PCA transformed features with three principal components.*

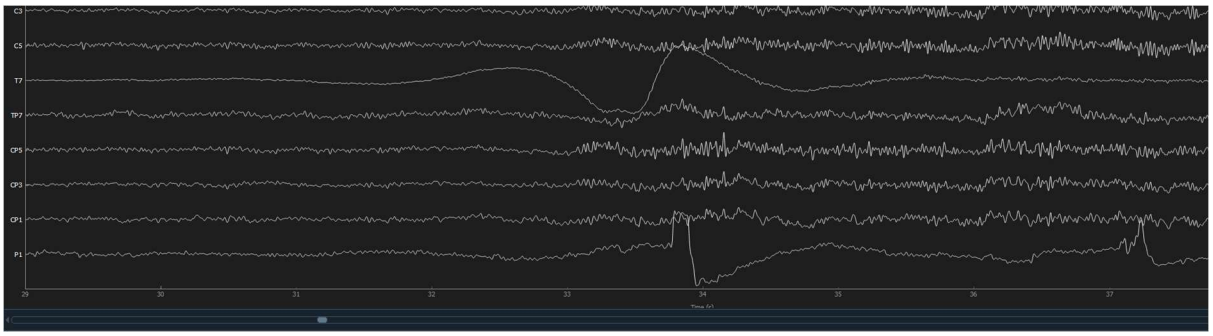


*Figure 9 Screenshot from the MNE Python library where it is seen that electrode channel T7 and P1 have very flat tendencies until they have large fluctuations. Since all channels have a standard deviation of 1 (after normalization), these channels must have a high amplitude fluctuating part - to be mostly flat elsewhere. These channels should have been dropped as they don't produce valuable information for the classification task.*

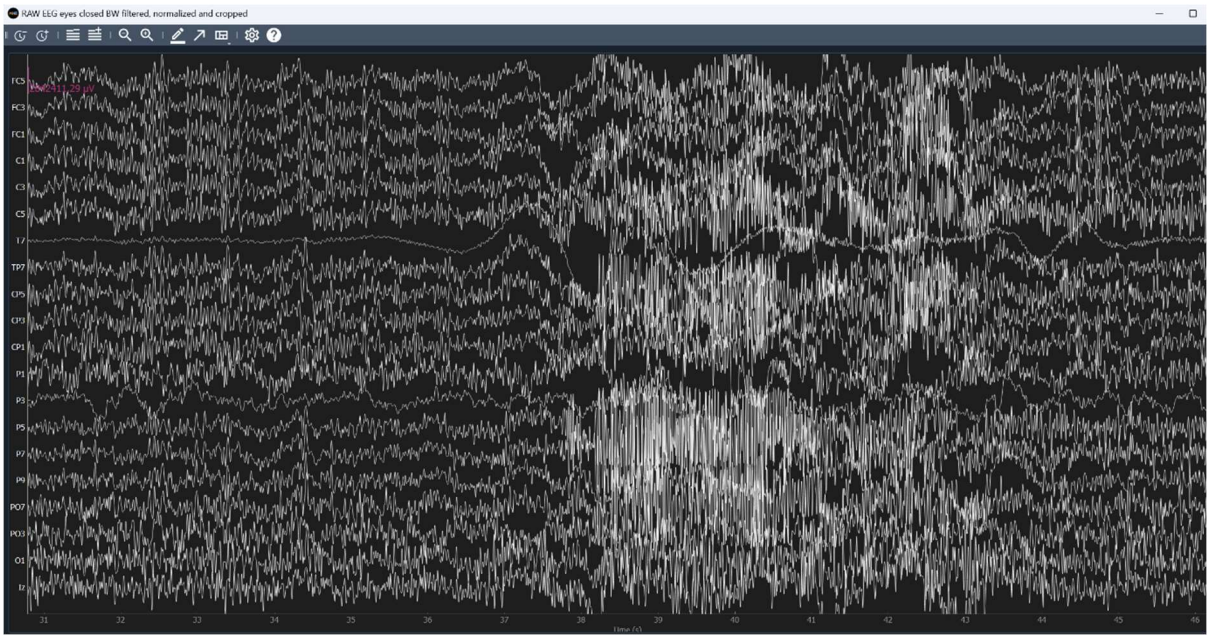


*Figure 10 Screenshot from the MNE Python library where it is seen that there are huge fluctuations*