EEG signal quality analysis by beacon biosignals

Jean Velluet, Julien Enguehard

May 15, 2025

Contents

1 Introduction

This challenge aims to use data recorded by the Dreem Headband to perform an automated analysis of Electroencephalography (EEG) signal quality. the Dreem Headband is an FDA-approved device designed for use in home-based clinical trials. These trials are conducted on a large scale, collecting longitudinal EEG data from a large cohort of participants. However, due to the unsupervised nature of these at-home trials, it is crucial to assess the quality of the acquired EEG data before conducting any analysis. Given the substantial volume of data generated, manual review is impractical, necessitating the development of an efficient algorithmic solution to evaluate the data automatically. The algorithm to be developed will generate quality labels for each two-second segment of data acquired on the five EEG channels of the device

2 Data and labeling

The Dreem Headband records EEG data on five channels sampled at 250 Hz. Although the headband also includes a three-axis accelerometer, accelerometer data is excluded from this challenge for simplicity.

To train the algorithm, we will have access to data scored by human experts. The human experts manually annotated Dreem data and selected spans of bad-quality signals for each channel. The following definition was used for good and bad quality. **Good Quality**: Sleep and wake-related eeg patterns are visible on the channel. They are not created by other noise sources (ECG/Respiration/...

or non-physiological noise). Perturbation of these patterns over less than one second or by artifacts that appear irregularly at a rate less than 1hz is not considered as hindering interpretability. **Bad Quality**: Sleep patterns are not visible because of artifacts that make their reading and interpretation impossible, OR sleep-like patterns are created by non-EEG/EOG sources and confuse the interpretation of the EEG.

The scorers selected the start and end of each bad-quality event; these labels were then transformed into a segmentation mask, where each label maps to 2 seconds of data. If the majority of two seconds overlaps with a bad-quality event, then this span is considered bad quality, and otherwise, it is considered good quality.

This means that the first label corresponds to the first two seconds of data, the 2nd label to the span of EEG from 2 seconds to 4 seconds, and so on. For each span of 2 seconds, five labels are provided, one for each EEG channel. We use the following encoding : 0 corresponds to bad quality over the majority of the two seconds for this channel while 1 corresponds to good quality over the majority of the two seconds for this channel

2.1 Data pre-processing

First, a butter band pass filter was used on the EEG signal in order to preserve frequencies in the 0.1–18 Hz range and removing noise outside this range, allowing better visualization as seen in Figure 2.1.

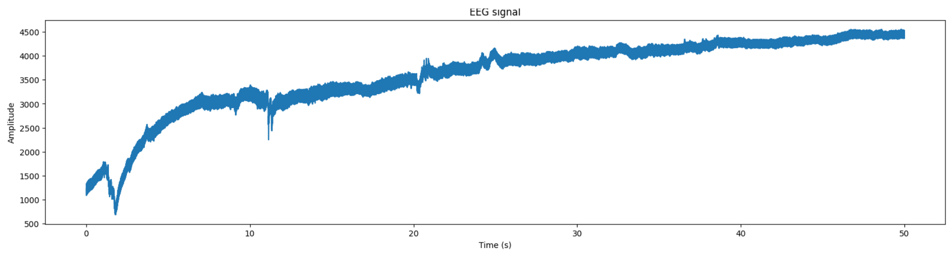

Figure 1: Raw EEG signal

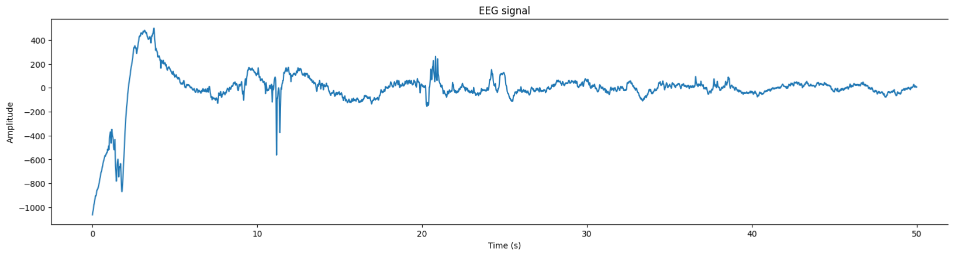

Figure 2: Filtered EEG signal after the butter band pass filter was applied.

Data was reshaped into 'windows', where 'window' contains the points corresponding to 2 seconds of data, using the `reshape_array_into_windows` function provided in the notebook. Then, the data was scaled to improve the performance and stability of the model using the `RobustScaler` scaler that centers and scales the data using the **median** and the **interquartile range (IQR)**, making it less sensitive to outliers compared to standard scaling methods. Each feature is transformed according to the formula:

$$X_{\text{scaled}} = \frac{X - \text{median}(X)}{\text{IQR}(X)}$$

3 Features extraction

Feature extraction naturally follows signal pre-processing and is a crucial step in EEG signal analysis. In the medical field, particularly with EEG signals, the trend toward big data is driven by multi-hour recordings and multi-channel acquisitions. Some of the most used features (found in [SK23] and [SI21]) were computed from the filtered signals. Then we plotted boxplot visualization (Figure 3) to assess the separability of ”good” and ”bad” EEG signal data based on each feature and visually identifies the features that are likely the most discriminative.

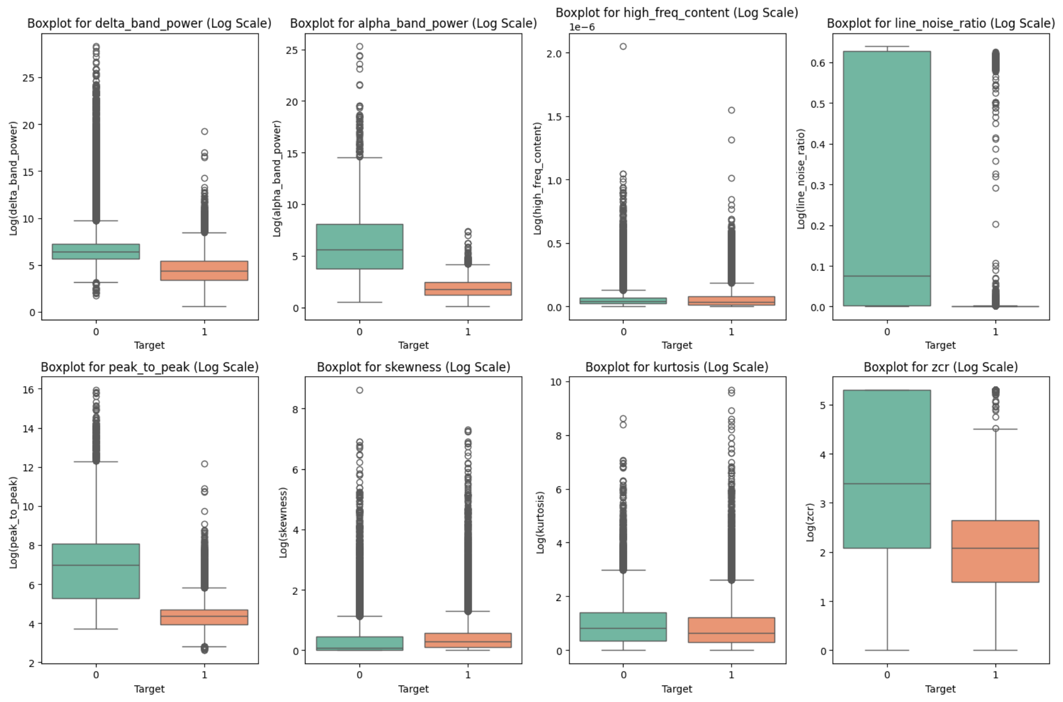

Figure 3: Box plots of several features, with the data labeled as 0 represented in green and the data labeled as 1 in orange.

There are some outliers are present in the data. To mitigate their effects, we applied Winsorization using a `FunctionTransformer` with the `apply_winsorization` function. Specifically, we capped the lowest 2% of values at the 2nd percentile and the highest 2% at the 98th percentile for each characteristic. This approach minimizes the influence of extreme values while preserving the overall structure and integrity of the data.

3.1 Feature selection

We first selected features that appeared to distinguish good quality signals from bad quality ones by visualizing boxplots for each feature. Then, we plotted the correlation matrix in figure 4 to visualize the correlation between each feature and detect groups of correlated features. When two features are highly correlated ($|r| > 0.8$, for example), this means that they contain similar information. Including them both in a model can lead to multicollinearity, which has a negative impact on model performance and interpretability. In this case, one feature from each highly correlated pair is selected, reducing redundancy while maintaining the predictive power of the dataset. As expected, there are some cluster of highly correlated features. For example, it’s not surprising that time-domain characteristics (amplitude, rms energy, energy, peak-to-peak) are strongly correlated, or that entropy measurements and fractal dimensions are also correlated.

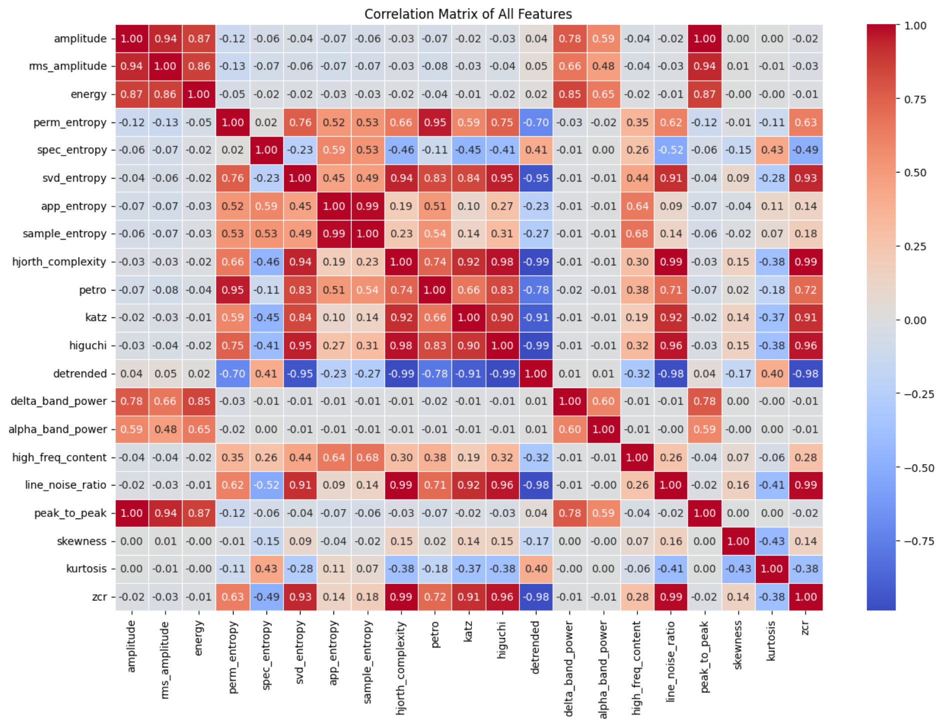

Figure 4: Correlation matrix for features that were computed

Finally, the features that have been selected are :

- **Amplitude:** Represents the magnitude of the signal.
- **Energy**
- **Skewness:** Quantifies the asymmetry of the distribution, indicating whether the data is predominantly spread out on one side
- **Kurtosis:** Provides a measure of the "peakedness" of a random signal.
- **Permutation Entropy:** Describes complexity of a time series or signal measured on a physical system through phase space reconstruction.
- **Spectral Entropy:** This measures the "forecastability" of a time series.
- **Sample Entropy:** Used for assessing the complexity of physiological and other time-series signals.
- **Katz Fractal Dimension:** A nonlinear dynamic metric that characterizes the complexity of time series by calculating the distance between two consecutive points.
- **Higuchi Fractal Dimension:** An approximate value for the box-counting dimension of the graph of a real-valued function or time series.
- **Detrended analysis :** Useful for analysing time series that appear to be long-memory processes
- **Delta band power:** Power within a specific frequency range (0.5–4 Hz), associated with specific brain states or activities.
- **Alpha band power:** Power within a specific frequency range (8–12 Hz), associated with specific brain states or activities.
- **High frequency content:** Used to characterize the amount of high-frequency content in the signal.

4 Model selection

Given the nature of the data (labeled), the task (binary classification), and the size of the dataset, we initially experimented with a k-Nearest Neighbors (k-NN) model, as suggested in the provided notebook. However, the performance of the k-NN model was insufficient for our requirements. So we considered two alternative approaches: Support Vector Machines (SVM) and boosting algorithms. Ultimately, the boosting algorithm was chosen due to its typically superior performance compared to SVMs, as well as its faster training times and better scalability with larger datasets.

The dataset was split into training and testing sets in a 70-30 ratio. A pipeline was constructed to preprocess the data as outlined above. The final step of the pipeline employed the `HistGradientBoostingClassifier` model, a gradient boosting algorithm particularly well-suited for structured data and efficient when dealing with large feature sets.

5 Theoretical Description of HistGradientBoostingClassifier

The `HistGradientBoostingClassifier` is a gradient boosting algorithm designed for classification tasks. It is an optimized implementation of gradient boosting that leverages histogram-based techniques for faster training and reduced memory usage, making it particularly effective for large datasets.

5.1 Gradient Boosting Overview

Gradient boosting is an ensemble learning technique where multiple weak learners (typically shallow decision trees) are combined to form a strong predictive model. The idea behind gradient boosting is to aggregate several weak learners (decision trees) and progressively, greater weight is given to individuals who were poorly classified in the previous step.

At each iteration, the model aims to minimize a predefined loss function $L(y, \hat{y})$, where y represents the true labels and $\hat{y}$ the predicted probability to be in the positive class. Formally, given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, the objective is to minimize:

$$\mathcal{L} = \sum_{i=1}^{n} L\left(y_i, \hat{F}(x_i)\right),$$

where $\hat{F}(x)$ is the ensemble model. The Gradient boosting binary classifier typically uses the `log loss` (binary cross-entropy), which is defined as:

$$L(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right],$$

The log loss penalizes incorrect predictions more severely when the predicted probability deviates significantly from the true class. The log loss function quantifies the accuracy of the predicted probabilities, with a lower value indicating better predictions.

Algorithm 1: Gradient Boosting

```
1 Initialize the model: with a constant value
      $$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$
2 for $m = 1$ to $M$ do
3     Compute pseudo-residuals:
          $$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n$$
4     Fit a base learner: Train a weak learner (decision tree) $h_m(x)$ using the training set
      $\{(x_i, r_{im})\}_{i=1}^{n}$
5     Compute the multiplier:
          $$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right)$$
6     Update the model:
          $$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$
7 Output: $F_M(x)$
```

By iteratively reducing the residual errors, gradient boosting minimizes the chosen loss function, leading to a robust and accurate model.

5.2 Histogram-Based Optimization in HistGradientBoostingClassifier

While traditional gradient boosting methods compute splits for each individual data point, the `HistGradientBoostingClassifier` introduces histogram-based techniques to improve computational efficiency and scalability, particularly for large datasets.

- **Binning of Features:** Continuous feature values are discretized into a fixed number of bins. This reduces the precision of the data, but it enables faster computation by reducing the number of comparisons needed to find the best splits.
- **Efficient Split Finding:** Rather than evaluating splits over every individual data point, the histogram approach finds splits by considering the bins, significantly reducing computational cost, especially for datasets with many features.
- **Memory Efficiency:** By working with binned data, the `HistGradientBoostingClassifier` reduces memory usage, allowing it to scale to larger datasets that might otherwise be too memory-intensive for traditional gradient boosting implementations.

These optimizations make the `HistGradientBoostingClassifier` highly efficient when dealing with large datasets, while still retaining the benefits of gradient boosting in terms of model accuracy.

6 Hyperparameter tuning

The performance and behavior of the `HistGradientBoostingClassifier` are controlled by several key hyperparameters:

- `learning_rate`: Controls the contribution of each weak learner to the final model. A smaller learning rate results in more iterations for convergence.
- `max_iter`: The maximum number of boosting iterations (trees to be added).

- `max_depth`: The maximum depth of the individual decision trees. Shallower trees can lead to faster training but may underfit.
- `min_samples_leaf`: The minimum number of samples required to be at a leaf node. This parameter controls the complexity of the trees and can prevent overfitting.
- `l2_regularization`: A regularization term that helps prevent overfitting by penalizing large model coefficients.

Due to the high dimensionality of the hyperparameter space, an exhaustive search was computationally prohibitive (however, for illustration purposes, we plotted in Figure 5 an example of the cross-validation error only for the l^2 regularization parameter on a grid). Therefore, hyperparameter tuning was performed using `RandomizedSearchCV`, which efficiently explores a predefined search space by sampling hyperparameter values randomly. This approach was applied to the boosting model, considering key parameters such as the learning rate, the number of iterations, tree depth, and regularization strength. The model was optimized based on accuracy through 20 iterations of randomized search with 4-fold cross-validation. Once trained, the best model was identified and evaluated on the test set. Metrics such as test accuracy and Cohen's Kappa Score were computed to assess the model's performance, demonstrating its classification capabilities on unseen data.

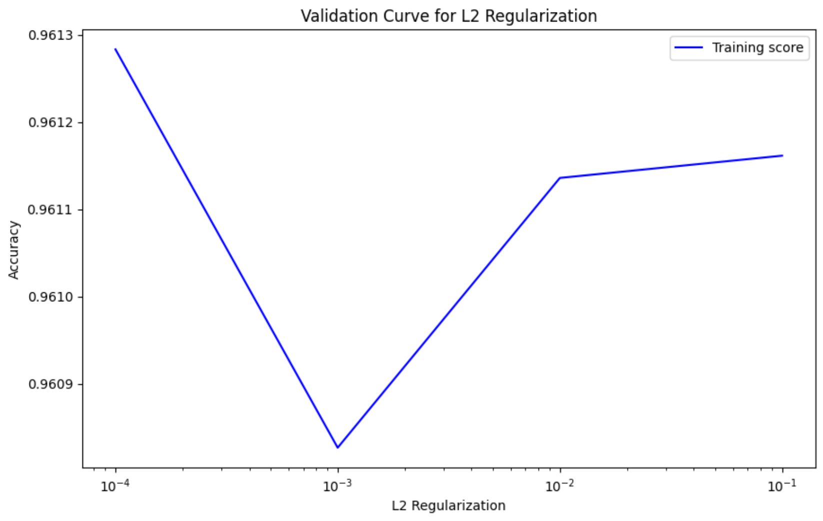

Figure 5: Cross-validation error as a function of the regularization parameter (log scale).

We identified the following model as the best after performing the cross-validation procedure:

```
Hist_GB = HistGradientBoostingClassifier(max_iter=500, max_depth=10, learning_rate=0.05,
    l2_regularization=0.01)
```

7 Conclusion

7.1 Model performances

We then computed the prediction of our best model on the test set to evaluate the model's performance. The values of several performance measure are summed up in table 6 and we also plotted the ROC curve in figure 7.

Performance Measure	**Value**
Accuracy	0.94
Precision (Positive Class)	0.95
F1 Score	0.95
True Positive Rate (TPR)	0.94
False Positive Rate (FPR)	0.04
AUC (Area Under Curve)	0.99
Cohen kappa score	0.89

Figure 6: Performance Measures of the Best Model

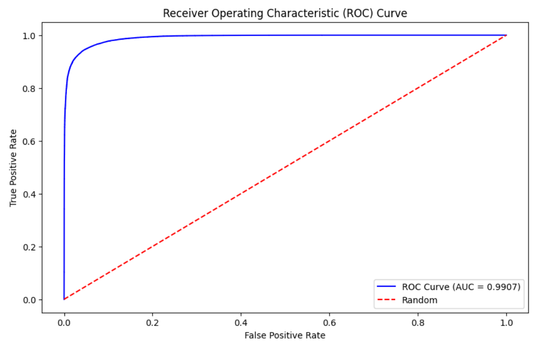

Figure 7: ROC curve

The model performs very well on the test data set with satisfactory performance measures.

7.2 Discussion

However, on the data provided for the challenge, the Cohen-kappa score is significantly worse than on the test set, at just 0.73. I tried to manually increase the regularization parameter to lower the variance and to allow a better generalization, but it did not significantly improve the score. Possible ways of improving this score could be to add new features or performing exhaustive search on the hyper-parameter space on a more powerful computer.

References

[SI21] Jovic A Stancin I, Cifrek M. A review of eeg signal features and their application in driver drowsiness detection systems. *Sensors*, page 3786, 2021.

[SK23] Anupreet Kaur Singh and Sridhar Krishnan. Trends in eeg signal feature extraction applications. *Frontiers in Artificial Intelligence*, 2023.