BCI Competition of Finger Flexion Predictions Based on ECoG Recordings

Celine Lee[1] and Sabrina Weng[1]

[1]*University of Pennsylvania, Bioengineering*

***Abstract*—The project focuses on predicting finger flexion movements using intracranial EEG recordings. Leveraging a dataset from the 4th International Brain-Computer Interfaces Competition, we aim to achieve a high correlation between predicted finger flexion movements and the target leaderboard data. Our final algorithm consists of dataset analysis, pre-processing of input data, model selection, and post-processing of predictions. In the pre-processing step, we apply a filter to filter out noises, extract relevant features, and calculate the response matrix on the raw data. In model selection, several models are trained and tested, ultimately culminating in the discovery that the ensemble model of AdaBoost and XGBoost outperforms all the others. Last but not least, the post-processing techniques further improve correlation by filtering out noises and interpolating predictions to match input size. Finally, utilizing our final algorithm, we achieve a correlation of 0.4717 between our finger flexion predictions and the target leaderboard data.**

***Index Terms*—BCI, ECoG, finger flexion prediction, machine learning, neuro-signal processing**

I. Introduction

BRAIN-computer interfacing (BCI) is an innovative field facilitating direct communication between the brain and external devices, with promising applications in healthcare, rehabilitation, and human-computer interaction. Our project focuses on predicting finger flexion movements for each of the five fingers using intracranial EEG recordings from three subjects. The dataset and tasks originate from the 4th International Brain-Computer Interfaces Competition [3], providing a standardized platform for evaluating BCI algorithms.

Our project commences with a thorough analysis of the dataset, laying the foundation for subsequent methodologies. For data pre-processing, we employ filters to reduce dataset noise and extract pertinent features essential for further training. Subsequently, we implement our method to train and predict finger flexion from ECoG signals. Through rigorous training and testing, we evaluate the effectiveness of our approach in capturing intricate hand movements from neural activity. Our efforts culminate in comprehensive results, encompassing our method's performance metrics and the ranking on the project leaderboard. Furthermore, we reflect on alternative methodologies explored during our investigation, offering insights into their effectiveness and limitations.

II. Methods

Our final algorithm utilizes ensemble modeling of XGBoost and AdaBoost for regression to train the pre-processed ECoG signal. The flowchart is shown in Figure 1 below. The input data is partitioned into training and validation datasets. The ECoG training data undergoes pre-processing, including filtering, feature extraction, and R-matrix calculation. Both the training ECoG and training glove data are used to train AdaBoost and XGBoost models. Once trained, these models make predictions on the validation ECoG data. The ensemble prediction is computed as the mean of the two model predictions and then undergoes post-processing, including a weighted moving average filter and cubic spline interpolation. The final interpolated results are compared with validation glove data to calculate the average correlation.

This process encompasses the training and validation phases. Once the algorithm is validated, the models are employed to predict the leaderboard data. The predictions, after post-processing, are submitted for leaderboard correlation calculation. Our final algorithm achieves a leaderboard correlation of **0.4717**.

A. Dataset Analysis

The dataset comprises data from three epileptic patients who participated in the recording experiments during surgical planning. Each patient had electrode arrays placed subdurally on their brain's surface to identify epileptic focus, with consent given for participation. While performing a finger flexion task, simultaneous recordings of ECoG signals and finger flexion time courses were conducted. Each subject in the raw training dataset includes hand movement data and ECoG signal data. The data is structured as a dictionary with keys $'train\_ecog'$ and $'train\_dg'$, containing lists of numpy arrays corresponding to each subject's data. The number of samples (m) in this training dataset consists of 300,000 samples while the sampling rate is set to 1000 Hz.

- $'train\_ecog'$ dataset contains ECoG signals with shapes (m x 62), (m x 48), and (m x 64) for subjects 1, 2, and 3, respectively, representing different numbers of ECoG channels while m denotes the number of samples.
- $'train\_dg'$ dataset includes finger flexion signals with shapes (m x 5), where m denotes the number of samples, and 5 signifies the number of fingers.

The dataset used to test on the leaderboard contains the testing ECoG dataset, specifically structured as $'leaderboard\_ecog'$, also containing lists of numpy arrays corresponding to each subject's data. The testing dataset comprises 147,500 samples.

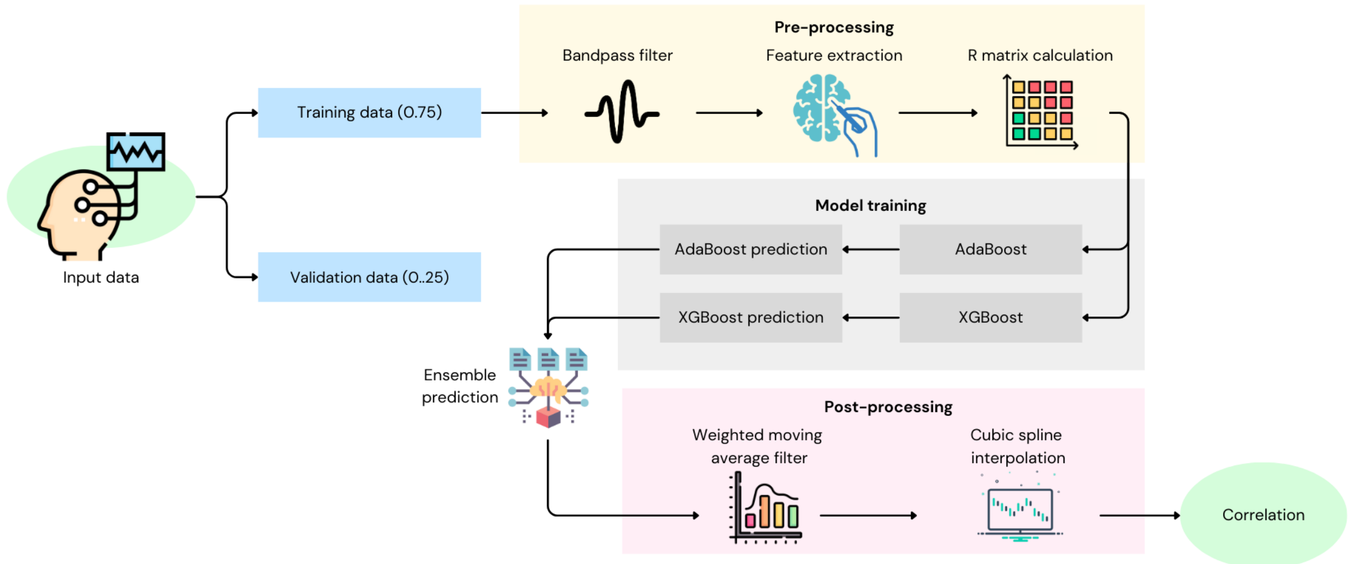
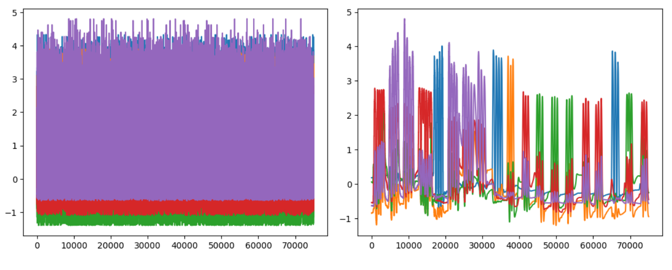


Fig. 1: The flowchart of the proposed pipeline.

- *'leaderboard_ecog'* dataset has shapes (m x 62), (m x 48), and (m x 64), representing ECoG signals from three subjects with 62, 48, and 64 channels, respectively.

B. Data Pre-processing

In our project, the model training process begins by partitioning our dataset into training and testing sets, using a 75:25 split and setting a random state of 42 for reproducibility. To maintain the integrity of the temporal sequence in our time series data, we opt to set shuffle to False. This decision ensures that the order of the samples remains unchanged when partitioning the data, which is critical for accurate modeling when it comes to time series data. The comparison of the training dataset before and after setting shuffle parameters are illustrated in Figure 2.



(a) Shuffle = True (b) Shuffle = False

Fig. 2: The visualization of the training dataset after splitting. (a) illustrates the potential noise introduced when training the dataset without shuffling, whereas (b) depicts the reduced noise observed when maintaining the original order of the data with shuffle set to False.

1) Data Filtering: In order to address the challenges of cleaning noisy signals while preserving essential information, we implement a filter function, aiming to select a filter type and parameters that achieve optimal results without compromising the integrity of the data. The optimal filter function utilizes a Butterworth bandpass filter with a pass band frequency ranging from 0.15 to 200 Hz frequency and an order of 4. We seek to effectively remove unwanted noise while retaining relevant signal components. We then employ forward-backward filtering ($filtfilt$ function from $scipy.signal$) to minimize phase distortion.

2) Feature Extraction: In handling high-dimensional ECoG data from numerous recording channels, relying solely on the model to sift through irrelevant features would complicate the learning process. To address this, we split the ECoG data into multiple windows with a window length of 100 ms and 50 ms window overlaps, and we prune nine different features using insights gained from data analysis, including average time-domain voltage, Hjorth activity, Hjorth mobility, Hjorth complexity, power spectrum with bandwidth range 12-30 Hz (Alpha), 30-70 Hz (Beta), 70-150 Hz (Gamma), zero-crossings, and absolute sum of differences.

Feature 1 calculates the mean voltage of the filtered ECoG signal across all channels in the given window. Feature 2-4 calculates the Hjorth Parameters, including activity, mobility, and complexity. The Hjorth activity represents the signal power, which measures the variance of the filtered ECoG signal across all channels. The Hjorth mobility represents the proportion of standard deviation of the power spectrum, which is defined as the square root of the variance of the first derivative of the signal divided by the variance of the signal. The Hjorth complexity represents the change in frequency and is measured by the ratio of the second standard deviation of the signal to the standard deviation of the first derivative. Feature 5-7 calculated the power spectral density (PSD) using Welch's method to estimate the power spectral density of each channel's signal. The bandwidth ranges are defined as 12-30 Hz (alpha), 30-70 Hz (beta), and 70-150 Hz (gamma). Feature 8 calculates the number of zero crossings in the filtered ECoG signal. It quantifies the number of times the signal changes its direction. Lastly, feature 9 computes the absolute sum of differences between consecutive samples of the filtered ECoG signal across all channels, capturing the overall change or movement in the signal.

The output of the feature calculation function will be a matrix of dimension $channels \times num\_features$, and the output of the windowed feature functions ($get\_windowed\_feats$), which calculates the features across all windows in the ECoG data, will be a matrix of dimension $num\_windows \times (channels \times num\_features)$.

3) R Matrix Calculation: Upon completion of training on the feature data, we encounter challenges in enhancing the correlation between predictions and the true target data. This difficulty is attributed to the intricacy of the ECoG data. We posit that constructing a response matrix and training the models utilizing the R matrix could offer insights into unraveling the complexity inherent in the input data. The R matrix function will take in a feature matrix and return a response matrix as an adaptation of the further training model. The response matrix R is defined as below (Warland et al., 199 [5]), while r_i^j is the feature value in channel j in time bin i:

$$\mathbf{R} = \begin{bmatrix} 1 & r_0^1 & r_1^1 & \cdots & r_{N-1}^1 & \cdots & r_0^\nu & \cdots & r_{N-1}^\nu \\ 1 & r_1^1 & r_2^1 & \cdots & r_N^1 & \cdots & r_1^\nu & \cdots & r_N^\nu \\ 1 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & r_{M-1}^1 & r_M^1 & \cdots & r_{N+M-2}^1 & \cdots & r_{M-1}^\nu & \cdots & r_{N+M-2}^\nu \end{bmatrix}$$

We also append a copy of the first N-1 rows of the feature matrix to the beginning of the feature matrix before calculating R due to the lack of feature data to populate the first N-1 rows of the R matrix, which will be used to predict the first N-1 finger angles. By incorporating features from multiple preceding time bins into each row vector, the R matrix enables the model to capture temporal dependencies and patterns in the data. Therefore, we anticipate that models trained on the R matrix should outperform those trained solely on the feature matrix, as the R matrix incorporates the temporal context of the data.

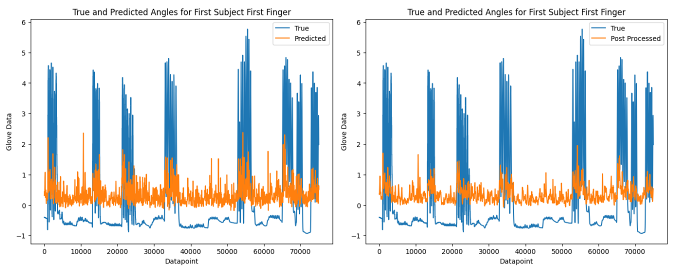
C. Proposed Model

Our final proposed model is an ensemble modeling of Extreme Gradient Boosting (XGBoost) [1] and Adaptive Boosting (AdaBoost) Regressor [4]. Ensemble modeling is a powerful technique that integrates predictions from multiple models to produce more accurate and robust results than individual models alone. XGBoost is an optimized gradient boosting library designed to be highly efficient, flexible, and portable, and it implements machine learning algorithms under a gradient boosting framework [7]. It is known for its efficiency and effectiveness in handling large datasets and complex relationships. AdaBoost (Adaptive Boosting) Regressor, on the other hand, is a boosting algorithm that focuses on minimizing errors by assigning weights to each data point and adjusting them iteratively based on the performance of preceding models. Combining XGBoost and AdaBoost Regressor significantly enhances the predictive performance while XGBoost is capable of capturing complex patterns and AdaBoost is capable of adapting to misclassified instances. Our final algorithm uses default parameters of default parameters for both models as hyperparameter tuning does not improve correlation further.

In ensemble modeling, we use the trained models to predict the validation dataset separately. Then, we take the average of the two predicted outcomes. Subsequently, the ensemble predictions (averaged predictions) undergo post-processing before correlation calculation. After verifying the performance of the models using the validation dataset, we proceed to prepare the leaderboard submission by predicting the leaderboard data using the validated models.

D. Post-processing

When plotting the true target data along with the interpolated predictions that were not post-processed, we observe significant noise in the interpolated predictions as shown in Figure 3a. To improve the quality of glove data predictions, we aim to apply post-processing techniques to the predicted data before interpolation. The final post-processing technique we choose consists of two steps: a weighted moving average filter and cubic spline interpolation.



(a) Without post-processing (b) With post-processing

Fig. 3: Comparison of target data and interpolated predictions with and without post-processing.

The weighted average is a variation of the standard moving average. In standard moving average techniques, each data point within a fixed window contributes equally to the average, resulting in uniform smoothing across the dataset. In contrast, a weighted moving average allows you to assign custom weights to each data point within the moving window. This means some data points in the window can have more influence on the calculated average than others, depending on the assigned weights. By adjusting the weights assigned to each data point, you can emphasize certain aspects of the data, filter out noise more effectively, or capture specific patterns or trends. We utilize a weights matrix of $[0.2,\ 0.3,\ 0.5]$, which places greater emphasis on the most recent data and less emphasis on the oldest data points in the window.

After passing predictions through the weighted moving average filter, we conduct a cubic spline interpolation to ensure the predictions match the size of the input data. The cubic spline interpolates between data points by constructing piecewise cubic polynomials that smoothly join together the data points. As depicted in Figure 3, we observe the presence of numerous noises in the interpolated results of predictions directly from the models. However, after post-processing, as illustrated in Figure 3b, many of these noises are filtered out, leaving behind signals that exhibit a higher correlation with the true target data.

III. Alternative Attempted Methods

A. LGBM Model

We come across the LightGBM (LGBM) model, abbreviated for Light Gradient Boosting Machine. It's built on a decision tree algorithm and serves various purposes including ranking, classification, and other machine learning tasks [6]. This model is tailored for efficiency, scalability, and high performance, particularly when handling large datasets. Given the substantial size of our input data, the LGBM model stands out for its optimized speed and efficiency. It employs a histogram-based algorithm to bin continuous features, effectively reducing the time and memory demands during training.

We make several attempts using the LGBM model. Our initial try employs default parameters, resulting in a leaderboard submission correlation of 0.3084. Recognizing the model's sensitivity to hyperparameters, we proceed with hyperparameter tuning, aiming to identify the most effective combinations for this specific dataset. Subsequent attempts focus on hyperparameter adjustments, such as varying the number of estimators, incorporating L1 regularization (alpha), and modifying the number of leaves. The final LGBM attempt yields a leaderboard correlation of 0.3467, surpassing checkpoint 1. It is fine-tuned with 100 estimators, an alpha regularization parameter of 0.1 to counter overfitting, and a reduced number of leaves set at 20. Alongside hyperparameter tuning, we explore different feature combinations, including the addition of the Hjorth parameter complexity.

However, despite the LGBM's efficiency in training, we encounter challenges in further improving the correlation beyond checkpoint 1. Therefore, we explore alternative models or variations of gradient boosting models, such as AdaBoost and XGBoost, which yield higher correlations on the leaderboard data.

B. LSTM Model

We design a basic RNN model comprising four LSTM layers, each followed by batch normalization and dropout. Batch normalization aids in stabilizing and expediting training by mitigating internal covariate shifts and enhancing model convergence. Dropout serves to mitigate overfitting by randomly deactivating a portion of input units during training, encouraging the model to learn more resilient features. In this instance, a dropout rate of 0.2 is employed. The final layer consists of a Dense layer with five neurons. However, the LSTM model's performance falls short, requiring considerable computational resources. It exhibits significant overfitting, evidenced by a 0.0068 correlation score on the leaderboard. Even though we increase the dropout rate, the model's performance does not improve significantly. This suggests that the issues may stem from the LSTM architecture itself, as indicated in the reference paper [2]. Despite efforts to mitigate overfitting through dropout, the model continues to exhibit low accuracy.

IV. Evaluation and Analysis

Throughout the course of the project, we experiment with various combinations of algorithms, such as employing different combinations of features in the feature matrix calculation function, different passbands in the filter function, different models and combinations of models, with and without post-processing techniques, etc. Our initial focus is to identify the optimal combination of the filter and feature extraction function. Table I illustrates that the highest validation correlation is achieved with filter function 2, featuring a passband of 0.15 to 200 Hz, and the feature extraction function with 9 features (as discussed in the previous section).

Validation Correlation	Filter function 1	Filter function 2
Feature function 1 (6 features)	0.1139	0.1863
Feature function 2 (9 features)	0.1029	0.1905

TABLE I: Correlation results on the validation set for different combinations of filter functions and feature calculation functions.

We then proceed with the optimal combination of filter and feature functions to test our models. In our attempts to train the XGBoost model and AdaBoost Regressor separately, we achieve correlation scores of 0.4227 for XGBoost and 0.4105 for AdaBoost Regressor on the leaderboard test dataset. However, combining these models into an ensemble model yields a mean Pearson Coefficient of 0.2042 for the validation set and 0.4227 for the testing set on the leaderboard. These results underscore the substantial improvement achieved by the ensemble model compared to the individual models, indicating its efficacy in enhancing correlation.

Model	Validation Correlation	Test Correlation
AdaBoost Regressor	0.2025	0.4105
XGBoost	0.1838	0.4227
Ensemble Model	0.2042	0.4227

TABLE II: Correlation result on the validation set and test set (leaderboard data) for AdaBoost Regressor, XGBoost, and Ensemble Model.

The ensemble model provides the best validation and leaderboard correlation results, and we want to further improve the correlation results on the leaderboard test data. We decide to implement post-processing techniques as discussed earlier, and Table III shows that with post-processing techniques implemented on the ensemble model, the predictions on the leaderboard data achieve a much higher correlation of 0.4717, allowing us to reach checkpoint 2 (0.45) of this project.

Ensemble Model	Validation Correlation	Test Correlation
Without Post-processing	0.2042	0.4227
With Post-processing	0.2077	0.4717

TABLE III: Correlation results on validation set and test set (leaderboard data) for Ensemble Model with and without post-processing techniques.

V. Discussion

We further discover that the correlation between the fourth, third, and fifth fingers' flexion can be attributed to physiological and neural factors. Physiologically, the control of the fourth finger involves muscles shared with the third and fifth fingers, including long flexors in the forearm and lumbrical muscles in the hand. Unlike the thumb, index, and little fingers, which have independent extensors, the middle and fourth fingers lack independent flexors or extensors, as highlighted in previous research. Additionally, the neural connection between these fingers further reinforces their interdependence. The radial nerve connects the thumb, index finger, and one side of the middle finger, while the ulnar nerve connects the other side of the middle finger, the ring finger, and the little finger. The intertwining of nerves for the ring and little fingers, as well as between the ring and middle fingers, results in difficulty moving these fingers separately. Consequently, the three fingers exhibit a strong correlation in flexion due to their neural and physiological inter-connectedness.

VI. Conclusion

This second part of the competition project builds on the previous phase, allowing us to apply all the signal processing and machine learning techniques we learned throughout the course. However, it can be both disappointing and overwhelming due to the relatively quick turnaround of this competition project and the numerous tuning and modifications required based on validation or leaderboard results. It's particularly disheartening when improvements in validation results fail to translate to improvements in leaderboard results. Initially, our team lack an organized method for testing and verifying the best combinations of different algorithmic components. Instead, we resort to random testing, hoping for better results. However, after passing checkpoint 1, we gain an insight into how to leverage the leaderboard to fine-tune our models and algorithmic components effectively, allowing us to reach checkpoint 2 relatively quickly compared to the previous stage. While our final algorithm may not be the optimal model for predicting finger flexion angles from the recorded ECoG data, our team dedicates weeks to meticulously modifying and fine-tuning various algorithmic components to achieve the best performance possible for the given task. Through this competition project, we gain invaluable insights and experiences. Given additional time, we are confident that further improvements can be made to enhance the model's performance even more.

References

[1] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. https://arxiv.org/abs/1603.02754

[2] Han, D., Liu, P., Xie, K., Li, H., Xia, Q., Zhang, Y., & Xia, J. (2022). An attention-based LSTM model for long-term runoff forecasting and factor recognition. *Environmental Research Letters*. https://doi.org/10.1088/1748-9326/acaedd

[3] Miller, Kai & Schalk, Gerwin. (2008). Prediction of Finger Flexion 4th Brain-Computer Interface Data Competition. https://www.bbci.de/competition/iv/desc_4.pdf

[4] Solomatine, D. P., & Shrestha, D. L. (2004). AdaBoost.RT: a boosting algorithm for regression problems. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, Budapest, Hungary (pp. 1163-1168 vol.2). https://doi.org/10.1109/IJCNN.2004.1380102

[5] Warland, D. K., Reinagel, P., & Meister, M. (1997). Decoding Visual Information From a Population of Retinal Ganglion Cells. *Journal of Neurophysiology*, 78(5), 2336–2350. https://doi.org/10.1152/jn.1997.78.5.2336

[6] Welcome to LIGHTGBM's documentation!. *Welcome to LightGBM's documentation! - LightGBM 4.0.0 documentation*. Retrieved from https://lightgbm.readthedocs.io/en/stable/

[7] XGBoost documentation. *XGBoost Documentation - xgboost 2.0.3 documentation*. Retrieved from https://xgboost.readthedocs.io/en/stable/

VII. Appendix

Final algorithm can be accessed here: VanLeeuwen