BE 5210 Final Project Report: Team Budget Neuralink

Milan Filo (mfilo@seas.upenn.edu)
Matt Timmons-Brown (tbmatt@seas.upenn.edu)
Uday Malhotra (udaym@seas.upenn.edu)

April 7, 2025

1 Introduction

In this work we present an algorithm and data-processing pipeline that, given a subject's electrocorticography (ECoG) data, can be trained to predict their finger flexion angles. Our algorithm is composed of a 1) data pipeline, 2) feature selection and 3) a Machine Learning model. In 1) we have a 5th order Butterworth filter that bandpass filters the raw ECoG data. In 2) we generate 12 features from the data, including features from previous homeworks such as line length, area, energy, as well as spectral power in certain frequency bands. We then use a window length of 100 ms, a displacement of 50 ms and a window lookback of 10 to create an R matrix. Finally, in 3) we train a gradient-boosted tree regression model (Light-GBM) to predict finger flexions. We performed hyper-parameter tuning and tried several models, and our final model resulted in strong 0.5286 performance on the leaderboard.

2 Final Algorithm

2.1 Pre-processing and Loading Data

Firstly we ingest the data using scipy, storing it in variables `train_dg` and `train_ecog`. Using scipy's `train_test_split` we split each of the 3 subject's data into training and validation, with shuffling set to False. In our testing we used a standard train/test split of 80:20. This allowed us to evaluate our model's performance on a held-out unseen validation test set before trying it on the leaderboard, increasing our iteration speed when developing.

2.2 Filtering

The electrocorticography (ECoG) data underwent a filtering process using the `filter_data` function, accomplished by using a Butterworth bandpass filter. The filter settings include a lower cutoff frequency of 1 Hz and an upper cutoff frequency of 150 Hz, which effectively isolates the frequency range most relevant for our signals by removing both high-frequency noise and low-frequency drift. The filter's order is set to 5, which provides a reasonable compromise between removing unwanted frequencies and preserving the shape of the waveform in the passband. The filtering process employs the `signal.filtfilt` method, which applies the filter in both forward and reverse directions. This zero-phase filtering approach ensures that no phase shift is introduced to the data, preserving the temporal relationships within the ECoG signals.

2.3 Window Calculation

The `get_windowed_feats` function segments filtered ECoG signals into overlapping windows for feature extraction - a crucial step for subsequent analysis such as decoding individual finger movements. This method follows recommendations from [4], which suggests using a window length of 100 milliseconds and an overlap of 50 milliseconds. In this implementation, the sampling rate (fs) is set to 1000 Hz, translating the window length to 100 samples and

the overlap to 50 samples. The step size, which is the difference between the window length and overlap, determines how much the window moves forward on each iteration, resulting in 50 samples. This overlapping window technique allows for a richer and more continuous representation of the ECoG data by ensuring that each signal segment is analyzed multiple times in slightly shifted contexts, thus increasing the resolution of temporal dynamics and improving the accuracy of finger movement decoding.

2.4 Features

The features extracted from the ECoG data to decode finger movements encompass a range of time-domain, frequency-domain, and complexity measures, each selected for their potential to reveal different aspects of the neurological signals involved in motor control. The two prominent features that were extracted from [4] were Spectral Power and Local Motor Potential (LMP). Using Welch's method, Spectral Power quantifies the power within a specific frequency band, capturing how the power of the ECoG signal is distributed across different frequencies. We used bands identical to the ones used in the Kubanek study (8–12 Hz, 18–24 Hz, 75–115 Hz, 125–159 Hz, 160–175 Hz). Similarly important to decoding finger flexions are LMPs, which are thought to represent the preparation and intention of movement - these are just calculated as the mean of a channel's values within a filtered window. In addition to those used in Kubanek's paper, we included the following features to be included in our feature extraction (many from the previous homeworks): Line Length, Area, Energy, Zero Crossings, Signal Entropy, and Hjorth Parameters (Activity, Mobility, Complexity).

2.5 R Matrix and Windowed Features

Building on the paradigm for predicting finger angles from ECoG data as described in Warland et al.[6], we use the `create_R_matrix` function that builds a regression matrix R from the extracted features. We calculated features from all available ECoG channels over M total time windows using the `get_windowed_feats` function that processes raw ECoG data through filtering, windowing, and feature calculation. In constructing the R matrix, the approach closely follows the methodology outlined by Warland et al [6]. For each time bin, a row vector is compiled which encompasses features from all ECoG channels over the preceding N time bins (see our tuning discussion below, for us, $N = 10$). This results in a design matrix where each row vector exhibits significant redundancy, stemming from the overlap of feature sets across successive time bins.

2.6 LightGBM

LightGBM (Light Gradient Boosting Machine) is a gradient boosting framework that utilizes tree-based learning algorithms. Given that our features are calculated over multiple time windows for each subject dataset, traditional tree-based methods like Random Forests or XGBoost were computationally prohibitive for our task. In contrast, LightGBM employs a histogram-based algorithm which not only reduces memory usage but also accelerates computation without compromising the accuracy typically achieved by similar tree-based models. Unlike conventional gradient boosting methods that grow trees level-wise, LightGBM grows trees leaf-wise, potentially reducing loss. This characteristic makes it exceptionally suitable for managing the large and complex datasets typical in our analysis. The model's efficiency in handling high-dimensional features and its scalability made it an ideal choice for our task.

2.6.1 Training and Evaluation

For the training and evaluation of our LightGBM, we used our 80-20 split training and validation sets. We wanted to maximize the amount of data available for training while still retaining a separate, unbiased subset for validation purposes, to judge the generalizability of our model rather than directly submitting and testing against the leaderboard dataset. To measure the performance of our model, we calculated the average correlation score across all fingers for each subject. In addition to correlation, we also monitored the mean squared error (MSE) on the validation set during training. This served as another indicator of model performance to keep an eye on as we

tried to improve the average correlations. This aim of our training and evaluation methodology was to ensure that our model is robust and capable of performing well on new unseen data as well, reflecting real-world variability and ensuring the models' applicability in practical scenarios.

2.7 Flow Chart of Algorithm

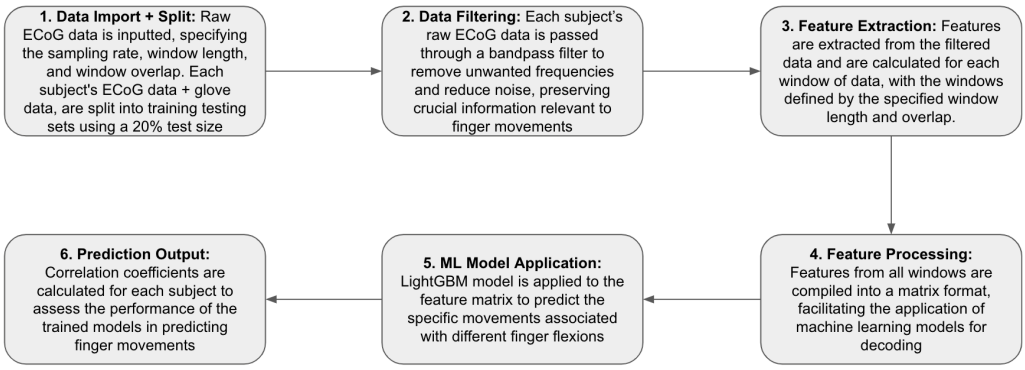

Figure 1: Flow chart of algorithm

3 Hyperparameter Tuning

3.1 N_wind

The `N_wind` parameter is responsible for the number of preceding windows used to calculate features in the R matrix. Each window in the model not only includes data from the current time window, but also from the previous `N_wind` windows as well. This approach is designed to capture temporal dependencies for making predictions. Theoretically, a too-small `N_wind` might not capture sufficient historical information for accurate predictions, and a too-large `N_wind` may introduce noise or irrelevant information if it effectively goes too far back in time.

We tested a range of values for `N_wind`: $[2, 4, 6, 8, 10]$, and evaluated the impact on the average correlation across all subjects on our local test set. Our findings are visible in Figure 2.

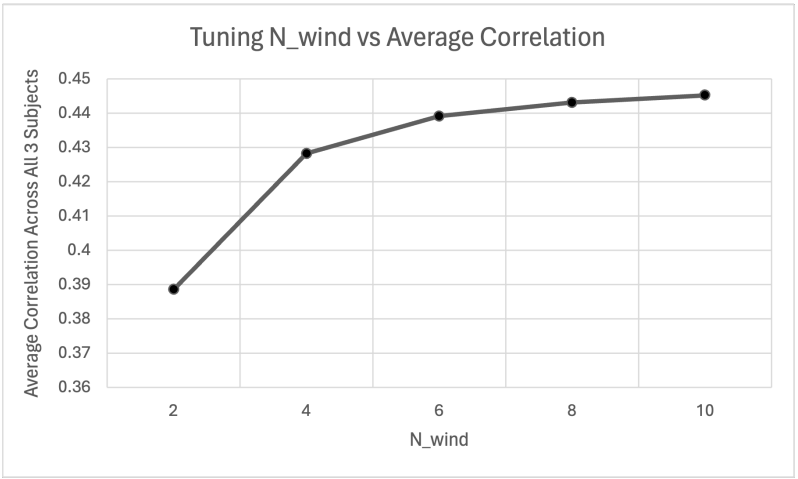

Figure 2: Tuning of the `N_wind` hyperparameter

As can be seen from these results, higher values of `N_wind` increased the average correlation across all 3 subjects.

This makes sense as we are increasingly using the past to predict the future - but we also see a diminishing effect as `N_wind` increases. While testing more values could've been interesting, each re-windowing of our data, followed by re-training, took over 1 hour - meaning that this hyperparameter tuning took over 10 hours! We settled on `N_wind` $= 10$).

3.2 num_leaves

After optimizing `N_wind`, we turned to the `num_leaves` parameter, which controls the complexity of the LGBM model by setting the maximum number of leaves in one tree, see [3] for further details. Theoretically, while a larger number of leaves in one tree may be able to capture more complex decision boundaries in the training data, it may lead to overfitting, so it is important to tune.

A range of values for `num_leaves` was tested, specifically $[10, 20, 30, 40, 50]$, to evaluate their impact on the average correlation across all subjects. This approach allowed us to observe the trade-off between model complexity and performance, helping us to identify the optimal setting that maximizes predictive accuracy without overfitting. We especially wanted to be mindful of overfitting to our training data as we use a large amount of features as well, and want to make sure that the model maintains good generalizability. We again evaluated performance based on the average correlation across all 3 subjects, given a model fit with a particular value of `num_leaves`, see Figure 3.

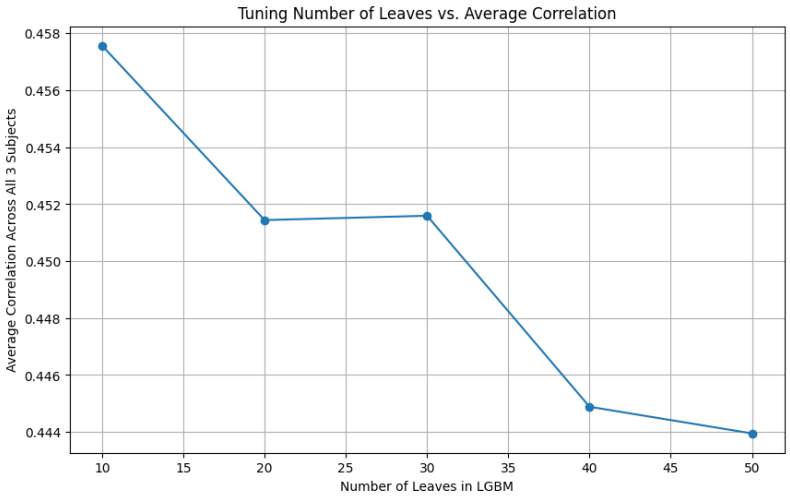

Figure 3: Tuning of LGBM's number of leaves hyperparameter

Based on these results, we decided to set `num_leaves` to 10 in our final model. Although the average correlation stays about the same (about 0.44 - 0.46) for all values tested, we decide to keep model complexity to a minimum given our vast feature set without compromising on performance.

4 Discussion

4.1 Optimization Strategies - What didn't work?

During the development of our final model, we embarked on a three-pronged optimization strategy focusing on filtering, feature selection, and model selection/tuning, all of which presented unique challenges and learnings.

Filtering: Initially, we experimented with various cutoff frequencies for signal filtering to isolate meaningful data while reducing noise. Some of these included reducing the order of our Butterworth Bandpass Filter, aggressive low and high cutoffs like 2Hz/100Hz, 5Hz/75Hz, etc.. Identifying an optimal cutoff was particularly challenging as it variably affected the signal integrity across different subjects, leading to inconsistent results. Some subjects tended to fare better, while the other would suffer a big drop in performance. Ideally, this would involve tuning all

hyperparameters simultaneously in a large N-dimensional grid search. However, due to computational limitations in our Google Colab work environment, such extensive optimization was not feasible.

Feature Selection: Our focus on feature engineering prompted us to explore several features, several of which did not make the cut in our final model. Another standout feature that we thought showed great promise was the Higuichi Fractal Dimension [5]. Despite its theoretical promise, the Higuichi Fractal Dimension proved computationally expensive and did not significantly correlate with our target variables, thus slowing down our development without substantial performance gains. We also considered dropping basic statistical features such as line length, area, energy, and zero-crossings to reduce overfitting, and reduce our total overall features. However, their removal led to a detrimental impact on validation performance, which was too significant to ignore.

Model Selection and Tuning: We initially employed a simple linear model, which quickly proved inadequate due to the non-linear nature of ECoG data. After optimizing basic feature selection, we started noticing that our baseline Machine Learning Model was already performing significantly better than the linear filter without any tuning. Subsequent trials with Support Vector Machines (SVM) and Convolutional Neural Networks (CNN) also fell short of expectations. We believe that the SVM, even after kernel tuning, struggled due to the high dimensionality and variability of the data. The CNN, which is meant to convolve over spatially/temporally-related features, did not align with our approach of using handcrafted features, making its use counterintuitive as we later realized. After several iterations, these experiences did not yield the desired improvements.

After a comprehensive series of modifications and optimizations that did not yield the anticipated improvements, and encountering challenges along the way, we successfully converged on our tuned LightGBM model.

4.2 Finger Flexion Correlation

We believe that the observed high correlations between the flexion of the fourth finger (ring) and those of the third (middle) and fifth (little) fingers can be attributed to both the anatomical and neurological characteristics of the human hand. The tendons controlling these fingers are interconnected, often sharing common muscle groups and tendon sheaths, which promotes movement synergy [1]. Furthermore, the neurology of these fingers provides additional insight into these observed correlations. The fourth and fifth fingers share innervation by the ulnar nerve, which coordinates their movements, while the third and fourth fingers are influenced by the radial nerve, leading to synchronized activity patterns between them [2]. This common innervation underpins the shared mechanical actions and reflexive responses that frequently occur during hand movements, thus reinforcing the interconnected functionality observed in our results.

During many routine tasks, such as grasping or holding objects, these fingers often move together or in coordination, further reinforced by their shared muscle groups and nerve supply. This bio-mechanical and neurological synergy could potentially facilitate efficient hand coordination, crucial for performing complex manual tasks. This synergy is directly reflected in the high inter-correlation observed in our study, underlining the practical implications of our findings for understanding hand movement patterns.

5 Conclusion

In conclusion, we feel that the project went well and our team created a strong algorithm. Each of us learned and applied multiple concepts from the class, ranging from signal filtering to neurological understanding and hyperparameter tuning. We tackled overfitting and getting insights from the literature. While working with a large amount of data proved time-consuming and computationally challenging, we overcame this by working together and trying out different models and approaches in parallel.

References

[1] Biomechanics OF THE HAND. https://ouhsc.edu/bserdac/dthompso/web/namics/hand.htm. [Online; accessed 2024-04-21].

[2] Nerves of the arm. https://myhealth.alberta.ca/Health/Pages/conditions.aspx?hwid=ax1000lang=en-ca. [Online; accessed 2024-04-21].

[3] Kaggle BEXGBOOST. Lgbm optuna hyperparameter tuning w. understanding. https://www.kaggle.com/code/bextuychiev/lgbm-optuna-hyperparameter-tuning-w-understanding. (Accessed on 04/21/2024).

[4] J Kubanek. Decoding flexion of individual fingers using electrocorticographic signals in humans. *Journal of Neural Engineering*. (Accessed on 04/21/2024).

[5] Anupreet Kaur Singh and Sridhar Krishnan. Trends in EEG signal feature extraction applications. *Frontiers in Artificial Intelligence*, 5. (Accessed on 04/21/2024).

[6] David K Warland, Pamela Reinagel, and Markus Meister. Decoding visual information from a population of retinal ganglion cells. *Journal of Neurophysiology*, 78. (Accessed on 04/21/2024).

A Algorithm