Investigating Neuronal Cell Classes and their Role in Cognition

by

Emily Huang

B.S. Computation and Cognition, MIT, 2022

Submitted to the
Department of Electrical Engineering and Computer Science, Brain and Cognitive Sciences
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN COMPUTATION AND COGNITION

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2024

Authored by:	Emily Huang Department of Brain and Cognitive Sciences February 7, 2024
Certified by:	Earl Miller Picower Professor of Neuroscience, Thesis Supervisor
Accepted by:	Katrina LaCurts Chair Master of Engineering Thesis Committee

Investigating Neuronal Cell Classes and their Role in Cognition

by

Emily Huang

Submitted to the
Department of Electrical Engineering and Computer Science, Brain and Cognitive Sciences
on February 7, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN COMPUTATION AND COGNITION

ABSTRACT

Classifying neurons into different cell classes is both an idea that has existed since the origins of neuroscience, and one that is essential to understanding the complex interactions of the brain. While there has been a substantial effort to categorize neurons morphologically, molecularly and physiologically in *in vitro* studies, there is a gap in experiments performed on awake and behaving animals. Using data collected from macaque monkeys performing a working memory task, and employing an unsupervised Gaussian mixture model (GMM) clustering algorithm, a number of different cell classes and their defining features were distinguished in area 7A, the lateral intraparietal area (LIP), the dorsolateral and ventrolateral prefrontal cortex (PFC) and the extrastriate visual area (V4). While the number of cell classes found across areas differed, there were several classes across areas that appeared to be correlates. Classes in each area also showed functional differences in information encoding during predictable trials and distributional differences in depth. This signifies both the potential of functionally distinct cell classes involved in prediction, as well as the existence of universal cell classes across different areas.

Thesis supervisor: Earl Miller
Title: Picower Professor of Neuroscience

Acknowledgments

I would like to express my gratitude to Dr. Earl Miller for supervising my work and for welcoming me into his lab, despite the newness of this program. I would also like to express my gratitude to Scott Brincat for mentoring me on this project, and for always being so patient and open. His insight was crucial in propelling this project forward.

A big thank you to my previous research mentors, Leo Kozachkov, Katya Tsimring and Michael Liu-Happ for starting me out on this path, and giving me guidance in not just research, but in all areas of life. Thank you for being someone for me to look up to when I was still a bright-eyed, bushy-tailed undergrad.

Another big thank you to Dr. Karchmer for taking me in as a TA for 6.1210 and giving me what was genuinely one of the best experiences of my life, as well as allowing me to fund this pursuit. And to all my students, thank you for being the reason I woke up in the morning. You made my weeks brighter.

Lastly, I would like to deeply thank my family and my close friends for being a source of unwavering support and love throughout both my time in college and this masters program. Even through the ups and downs of this process, I was always able to turn to someone. The only reason I'm able to keep beating against the current is because of you – thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The identification and classification of neurons into different neuronal cell types is essential for understanding the complex behaviors and the local and distal circuit dynamics of the brain. While the foundations of neuroscience had its origins in attempting this classification [1], fascinated by all the diversity the brain has to offer, attempts were often hindered by the lack of computational methods available at the time, resulting in a laborious, and often biased approach to the problem [2]. Focus in the field shifted towards more hypothesis driven studies. However, with the recent development of reliable technological methods that significantly reduce human bias and offer the ability to perform powerful computations on large datasets, there has been a resurgence of this effort to identify different neuronal sub-classes, their properties, and their functional roles [3], [4].

There are several different approaches to the classification of cell types, generally categorized by morphological, molecular or physiological features. Morphological methods aim to classify cells based on their structure, such as their dendritic and axonal shapes, and their branching patterns, while molecular methods separate classes through properties such as protein and mRNA composition [2]. Physiological methods, on the other hand, distinguish neurons through properties such as firing rate, inter-spike intervals, and waveform shape. Although a large number of different cell classes have been identified using these methods

in *in vitro* studies, until recently, there had been a lack of classes identified using data collected from awake animals [3], as well as limited studies done with data from the primate pre-frontal cortex, critical for higher level cognition and decision making [5]. It is essential to observe the function of different cell types within a natural framework, with an animal engaging directly in tasks. Only then, can the full function of a cell class, as well as its interactive dynamics, both local and distal, be understood.

Classification from extracellular recordings has shown a bimodal distribution of spike-waveform widths, with each peak characterizing "broad-spiking and "narrow-spiking" cells. The broad-spiking cells are thought to be a correlate to excitatory pyramidal cells, while the narrow-spiking cells are thought to be GABAergic, inhibitory cells [6]. However, more recent studies have diversified and broken down the two categories even more, suggesting that there are more features than just spike waveform width that can be used to identify cell classes [3]–[5], [7]–[9]. One such study found four clusters, two of which mirrored the traditional broad and narrow-spiking cells, and the other two of which mirrored potential morphologically identified classes, non-fast spiking inter-neurons, and intrinsically bursting neurons [4]. Others identified many more cell types - up to seven or eight, with even more diversity beyond broad and narrow spiking [3], [5], [8]. These studies included more than just waveform features in their clustering analysis; they introduced other features of the neuron, such as firing rate features, and inter-spike interval features, and it made clear that there was more to cell classification than just the shape of the waveform itself; a neuron had other intrinsic properties that allowed it to be differentiated.

Once these features of interest are identified, a computational approach to grouping different neurons based on these features needs to be chosen. Since the problem of classification is an inherently unlabeled problem, an unsupervised clustering algorithm such as K-means clustering [3], [5], [8], [9] or a Gaussian mixture model [4] is chosen. Both models have advantages in different situations. K-means clustering uses a measure of squared euclidean distance and attempts to minimize the sum of the averages of each cluster [10]. On the other

hand, a Gaussian mixture model uses a model that maximizes expectation, and data points are assumed to have a normal distribution. This often makes it more sophisticated than K-means, and more able to handle non-linear data [11].

This project isolated five features of interest – trough to peak time and repolarization time, two markers of waveform width, the mean firing rate of the neuron, and its coefficient of variance (CV) and local variance (LV), two measures of how regular the neuron's firing patterns are. A Gaussian Mixture Model was chosen over a k-means model due to its lack of bias in the perceived shape of the clusters, since there was no prior that suggested what geometry the data would be.

Chapter 2

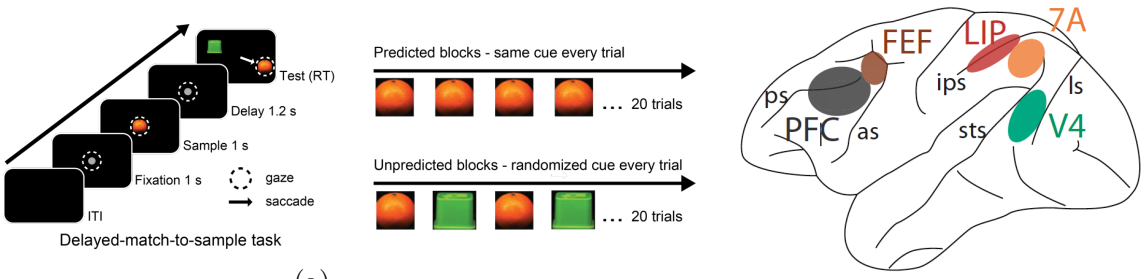
Methods

2.1 Datasets

Two datasets collected in the Miller Lab were used for this project – the laminarPharm dataset, collected by Alex Major, and the wmPredict dataset, collected André Bastos. The data was collected from two macaque monkeys, Selma and Lucky. Data was formatted into the standard Miller Lab data format. Preliminary clustering and plotting was done using only the laminarPharm dataset; the wmPredict dataset was then added so that the clustering algorithm could have more data points to produce a more accurate clustering. Since both datasets were formatted using the standard Miller lab format, most functions were easily extended – any discrepancies were dealt with manually in the code.

2.1.1 Task Design

Both datasets contain data from working memory tasks, with manipulations on sample predictability and laminar recording. The laminarPharm dataset contained additional data with manipulation of laminar pharmacology, though that data was not used for this project. The task was a delayed match to sample task (DMTS), which consisted of two different trial blocks – blocks in which the sample was predictable, and blocks in which the sample was



(a) (b)

Figure 2.1: (a) DMTS task setup; (b) Recording areas illustrated.
NOTE: FEF is labelled, but was not used in any analysis.
Both figures taken from laminarPharm documentation.

randomly selected from a set of three images (unpredictable). Samples were chosen from a set of 12 used across all sessions. In the wmPredict dataset, blocks were 50 trials long. In the laminarPharm dataset, they were 20 trials long.

Each trial consisted of a fixation period (1 second), followed by a sample image (1 second), a delay period (variable) and finally, a test array containing 1-2 distractor objects. Subjects had to saccade to the image matching the sample shown previously. The task is illustrated in Fig. 2.1a.

Data in the laminarPharm datasets was recorded from the dorsolateral and ventrolateral prefrontal cortex (dlPFC and vlPFC respectively), area 7a of the posterior parietal cortex (7A) and the extrastriate visual area (V4). In addition to those areas, data in the wmPredict dataset recorded from the lateral intraparietal area of posterior parietal cortex (LIP) was also used. Data from dlPFC and vlPFC was combined for the analysis, and compiled into datasets labeled 'PFC'. Areas are highlighted in 2.1b.

2.2 Spynal Library

The analysis in this project was heavily supported by Spynal, a neural data analysis library coded by Scott Brincat [12]. Various functions from Spynal were used, mainly in preprocessing data. These functions included loading data, extracting spike rates, inter-spike

intervals, waveform statistics, concatenating arrays, calculating percent explained variance, and interpolating waveform data. Additionally, plotting functions from Spynal was used to create the raster plots shown later on.

2.3 Data Pre-Processing

2.3.1 Spike and Waveform Data

Data pre-processing consisted of several steps: (1) Loading and reshaping the data, (2) Area selection, (3) Trial Selection, (4) Unit selection, (5) Wave Alignment and (6) Concatenation. Code for all pre-processing, feature extraction, clustering and plotting can be found in Appendix A.

Loading Data

All relevant data for each session was loaded using the matIO module from Spynal. Spike and waveform data was reshaped such that all individual trains were at least 1-dimensional arrays, and all individual waveforms were at least 2-dimensional. The data for the proper areas was selected by isolating the corresponding indices for the area from the 'unitInfo' dataframe. Separation by area was chosen so that area would not be a potential hidden parameter used in clustering. Additionally, it's possible that not every area has the same distribution or prevalence of varying cell-types, so the data from different areas was separated to avoid inaccurate grouping.

Trial Selection

Valid trials were defined as trials that were (1) correct and (2) ≤ 5 trials before drug onset (in the 'laminarPharm' dataset). Trials after drug onset were chosen to not be used in analysis due to previous analysis suggesting an almost complete suppression of activity after drug onset, and thus would not provide much use for clustering.

Unit Selection

Valid units were defined as units that (1) had a mean spike rate of ≥ 1 spike/ second and (2) $< 0.1\%$ of inter-spike intervals that were within 1 millisecond. These criteria weeded out silent cells and poorly isolated cells respectively. Additionally, a few outlier units with inverted waveforms due to an error in the data recording process were also not used for the analysis.

Wave Alignment

Wave alignment served to align all waveforms to the mean trough of all the waveforms in a session. Firstly, waves were interpolated by 10x using the `interpolate` Spynal function. Then, the mean waveform over all units was calculated. The trough of that waveform was extracted to be the mean trough that all the waveforms would be aligned to. All waveforms were aligned to the trough by shifting the waveform either left or right; the ends were padded with NaN values as necessary. Zero-padding was avoided, so that zero values would not affect any means taken over the waveforms. Then, the mean aligned waveform was found by taking the mean over all of the aligned waveforms, ignoring NaN values.

Concatenation

Data from across all sessions was concatenated into large dataframes. Only sessions with more than two valid trials and two valid units were used – otherwise, they were not concatenated. All dataframes were then saved for ease of future access, so that they would not have to be recalculated every time the clustering algorithm ran.

Merging vlPFC and dlPFC

In order to create the PFC dataset, data for vlPFC and dlPFC were pre-processed, concatenated and saved separately, and then manually merged. dlPFC data was always concatenated to the vlPFC data, for consistency.

2.3.2 LFP Data

LFP data was not heavily processed due to the data having already been pre-processed into the Miller lab data format. Similar to spike and waveform data, the units from the area of interest were isolated in the LFP data. Then, LFP data was pooled into two groups – data recorded from electrodes above to Layer 4, and data recorded from electrodes deeper than Layer 4. Depth information was taken from the corresponding area electrodes in the `electrodeInfo` dataframe. Depths were recorded based on the estimated relative depth of the electrode to layer 4, such that a label of 0 corresponded to layer 4, a label of > 0 corresponded to deep layers, and a label of < 0 corresponded to superficial layers.

2.4 Feature Extraction

Five features of interest were identified. Two were spike waveform features – the time in milliseconds from a waveform's trough to its peak, and repolarization time. Two were spike rate features – firing rate, coefficient of variation (CV). The last feature was a spike timing (ISI) feature, local variation (LV). The five features of each unit was recorded in a Pandas dataframe.

2.4.1 Spike Waveform Feature Extraction

Spike waveform features for each unit were extracted from the mean aligned waveform for that unit, which was found through the wave alignment section of pre-processing. Both trough to peak time and repolarization time were calculated using the `waveform_stats` Spynal function. An array of the trough to peak time and repolarization time for each unit was returned, and added as columns in the feature dataframe.

2.4.2 Spike Rate Feature Extraction

The mean firing rate for each unit was taken from the firing rates found using the `rate` Spynal function. Firing rate was averaged over all time points, as well as over all trials for a unit. An Anscombe transform was then performed on the mean rates, so that the data would be normalized.

The coefficient of variation for each unit was calculated using the `rate_stats` Spynal function, which took the mean firing rate of each trial per unit as input, and returned $\frac{\text{SD}(rates)}{\text{mean}(rates)}$ for each unit.

Arrays with the mean firing rate and CV were added as columns in the feature dataframe.

2.4.3 Spike Timing Feature Extraction

The local variation of each unit was found by calculating the LV of each trial of the unit using the `isi_stats` Spynal function. If the trial had ≤ 1 ISI, meaning there were no intervals to compare against each other, the LV was recorded as NaN. After all the LV of all trials was calculated, the mean LV was found by taking a mean over all LVs, ignoring NaN values. The final array of the mean LV per unit was added as a column to the feature dataframe.

2.5 Clustering

The optimal number of clusters for each area was found by performing Gaussian Mixture Model (GMM) clustering on the given feature dataframe. Before feeding the data into the GMM, the data was standardized by using sklearn's `StandardScaler` and `fit_transform` functions, which produces a scaler and then fits the scaler to the data and transforms it, respectively. GMM clustering was performed using sklearn's `GaussianMixture` function from the `mixture` package.

2.5.1 Optimal Cluster and Model Selection

500 repetitions of the GMM algorithm was performed, each repetition fitting the model on component sizes in the range 2-27. For each repetition, every time a model was found for a certain component size, the model was then fit to the features, and cluster labels for each unit (which cluster each unit belonged to based on its five features) were predicted.

The Bayesian information criterion (BIC) and Akaike information criterion (AIC) values were then calculated by using sklearn's `bic` and `aic` functions from the `mixture` package. The BIC value, AIC value, the current model, and labels were then recorded. The best model per repetition was determined by finding the model with the minimum BIC value; the best component size per repetition then recorded as the component size of the model that produced that minimum. AIC value was not used to find the best fitting model since AIC doesn't penalize for larger component sizes. BIC penalizes for larger component sizes, so that overfitting doesn't occur.

The best component size (c) over all repetitions was determined by calculating the mode of the array of best component sizes per repetition. The best c-component model was then found by taking the c-component model with the lowest BIC value. The model itself, the labels produced by that model, and c was returned for each area. Cluster labels were saved for ease of access when performing analyses.

2.6 Depth Distribution

In order to investigate the distribution of each cluster across different depths, depth data was isolated from the `unit_info` dataframes from both datasets. Both datasets took a measure of depth relative to Layer 4 that was estimated by the crossing point of relative power in beta and gamma bands. In the wmPredict dataset, this was called `betaGammaDepth`, and in the laminarPharm dataset, this was called `laminarDepth`. In both datasets, negative numbers meant more superficial depths, while positive numbers meant deeper depths. Zeros

were considered to be within layer 4.

In the wmPredict dataset, depths that could not be reliably estimated were labelled as NaN values. These values were converted to a value of -3, which was lower than the minimum depth value recorded, to allow the investigation of whether certain clusters had more depth information missing than others.

In addition to the original depth data, which was discrete, a "jittered" version of the depth data was generated to aid in plotting the distributions. Jittered values helped prevent the points from being plotted on top of each other due to the confines of the figure. The array of jittered values was created by randomly selecting from a uniform distribution in the range of [-0.02, 0.02] using numpy's random package, `np.random.uniform`. The random values were then added to the original depth values.

2.7 Information Encoding

The information encoding of each cluster was measured by finding the percent explained variance of each unit's firing rate over time by the variation in sample (three groups) or block predictability (two groups). The percent explained variance was calculated using the Spynal function `neural_info`.

In addition to using the full sample data, a truncated version of the sample labels was created by filtering out the trials that occurred in predictable blocks, due to the presence of units that would indiscriminately fire during those predictable blocks. The percent explained variance calculated using the truncated sample data was also calculated using the same Spynal function.

2.8 Spike Field Coupling

Spike field coupling/ synchrony was calculated using the Spynal function `spike_field_coupling`, which uses the default method of measuring the magnitude of pairwise phase consistency

(PPC). The LFP data was truncated to the three seconds surrounding sample onset ([-1, 2]) so that its time dimensions would match the spike time dimensions used for previous analyses. In each area, the unique probes used for recording in that area were identified. Only linear probes were used – single electrodes (only present in the wmPredict dataset) were excluded from this analysis, due to the lack of comparable depth data. To reduce processing burden, the lfp data for a specific probe was grouped into "deep" lfp data, measured with the "deep" electrodes (relative to Layer 4), and "superficial" lfp data, measured with the "superficial" electrodes. The spike data for that probe was then isolated by referencing the unique ID of the probe.

Spike times were converted to boolean spike trains using the Spynal function `times_to_bool`. Then, for every unit, the PPC between the spikes recorded by that unit, and the deep and superficial lfp data was calculated, yielding two datasets. The data was then visualized using the Spynal `plot_spectrogram` function.

Chapter 3

Results

3.1 Preliminary Clustering

The preliminary clustering done with only the laminarPharm dataset, and for only datasets recorded from 7A, PFC, and V4, revealed an optimal component size of 4, 3, and 5 for 7A (n_pts = 354), PFC (n_pts = 121), and V4 (n_pts = 261) respectively. Table 3.1 illustrates this data. The wmPredict dataset was added after preliminary results to increase the number of data points for the model to be able to predict on, as the original dataset was small, especially for the PFC.

In the preliminary clustering, some clusters had high variability in waveform shape (Figure 3.1). Most notably, the last cluster of area 7A has an increased number of outlier waveforms [3.1a], categorized by those with repolarization times or trough to peak times that were more than 2.5 standard deviations outside of the mean waveform, shown in black. Other clusters, such as the last two clusters in V4[3.1b], and the last cluster of PFC[3.1c], appear to have two different waveform shapes – one with a narrower trough to peak, and one with a broader trough to peak.

Table 3.1: Clustering Information, laminarPharm Only

	7A	PFC	V4
Number Datapoints	354	121	261
Optimal Component Size	4	3	5

Table 3.2: Clustering Information, Full Dataset

	LIP	7A	PFC	V4
Number Datapoints	689	765	1083	1663
Optimal Component Size	7	5	8	8

3.2 Cell Classes

The minimum mean BIC value for each component size revealed an optimal component size of 7, 5, 8, and 8 for areas LIP (n_pts = 689), 7A (n_pts = 765), PFC (n_pts = 1083), and V4 (n_pts = 1663) respectively. Table 3.2 illustrates this data.

All four areas showed a distinct "elbow" in the mean BIC curve for component sizes, indicating a clear optimum in choosing component size [3.2]. On the other hand, the mean AIC curves don't show a distinct elbow for any areas [3.3] – this is due to AIC allowing for overfitting by not punishing larger component sizes, which is why only the BIC values were used to determine the optimal component size.

The size of the data appeared to have no correlation with the optimal number of clusters found (i.e. more data points fed into the model did not imply a larger component size). This can most notably be seen in LIP having an optimal component size of 7, while having the smallest number of data points out of all the areas (689), while 7A, having the second smallest number of data points out of the areas (765), and closer in size to the LIP dataset than the other two areas, had an optimal component size of 5, smaller than that found for LIP.

3.2.1 Class Features

The mean value of each parameter for each cluster is shown in Figure 3.4. While each area had a differing number of cell classes, there were some trends in parameter values that seemed to remain constant across areas. Each area had a cell class characterized by a broad waveform (TTP $\approx$ .0065 ; RPT $\approx$ 0.0030), a lower mean firing rate ($3.5 < \text{MR} < 5$), and a semi-regular firing pattern ($0.5 < \text{CV} < 1$) and medium "burstiness" (LV $\approx$ 1.0), such as cluster 0 in area 7A [3.4b], cluster 7 in area PFC [3.4c], and cluster 7 in area V4 [3.4d]. Clusters 0 and 4 in area LIP seem to have similar properties, only being split up by cluster 0 having a smaller repolarization time, a smaller mean rate, as well as a less regular firing pattern [3.4a]. For convenience, this cluster will be referred to as B1. Inspection of the marginal distribution of each feature for each cluster in Figure 3.5 suggests that B1 is actually a group of neurons with highly variable trough to peak and repolarization times, which is why their mean waveform trough to peak and repolarization times are larger than other clusters.

The trough to peak and repolarization marginal distributions of each area clearly shows a bimodal distribution of waveform width, which would be the distinction between broad-spiking and narrow-spiking neurons. While there are some clusters that fit into either side of the bimodal distribution, there are others, like cluster 3 in 7A which has peaks on both sides. This suggests that what characterizes the cell class is not the waveform shape, but rather the rate or ISI features [3.5b].

In addition to the broad-spiking B1 class, each area also appears to have a narrow waveform cluster ($0.0002 < \text{TTP} < 0.0003$; $0.00005 < \text{RPT}\ 0.00015$) with a very high mean rate ($\text{MR} > 6$), and low variability in both pattern and burstiness ($\text{CV} < 0.6$; $\text{LV} < 0.95$). This pattern corresponds to cluster 3 in area LIP, cluster 0 in area PFC, and cluster 0 in area V4. The cluster in 7A that seems to correspond to this pattern (cluster 3) seems to have a slightly waveform than the others (TTP $\approx$ 0.00045 ; RPT $\approx$ 0.00018). For convenience, this

cluster will be referred to as N1.

Besides clusters that can be further categorized within the broad/ narrow spectrum, there appear to be clusters defined by very high/ low variability, or a high/ low mean rate. Cluster 2 in LIP, cluster 2 in PFC and cluster 6 in area V4 can all be characterized by a very high CV, implying a much more irregular firing pattern than the other cell classes in the area. This cell class is also characterized by a low mean rate ($3 < \text{MR} < 4$), a narrower waveform ($0.0003 < \text{TTP} < 0.0005$; $\text{RPT} \approxeq 0.00015$), and are not particularly bursty ($0.85 < \text{LV} < 1.0$). Area 7A doesn't appear to have a direct correlate, although cluster 2 exhibits a similar pattern, with a higher burstiness ($\text{LV} \approxeq 1.15$). For convenience, this cluster will be referred to as C1.

Similarly, cluster 6 in LIP, cluster 5 in PFC, and cluster 4 in V4 seem to be characterized by a high LV, or being more bursty neurons. It also appears to be characterized by a more regular firing pattern, a lower mean rate, and narrow to medium sized waveforms. For convenience, this cluster will be called L1.

3.2.2 Average Waveforms

Cell classes that demonstrated similar parameter values, especially those strongly characterized by their waveform features, appeared to have similar average waveform shapes, and little outliers [3.6]. The broad spiking cluster, B1, had larger trough to peak durations ($\tilde{1}50$ timepoints) compared to the narrow spiking cluster, N1, which had a trough to peak duration of less than 100 timepoints, as expected. This can also be seen in other broad spiking clusters, such as cluster 4 of LIP and cluster 3 of V4 [3.6a, 3.6d].

Notably, C1, which was characterized by a high CV, appeared to have two distinct waveform groupings, as can be seen by the average waveform plots for cluster 2 in LIP, cluster 2 in PFC, and cluster 6 in V4 [3.6a, 3.6c, 3.6d]. Similarly, L1, which was characterized by a high LV, shows the same waveform groupings in the average wave form plots for cluster 6 in LIP, cluster 5 in PFC and cluster 4 in V4 [3.6a, 3.6c, 3.6d].

3.2.3 Cluster Distribution

Figure 3.7 shows the percentage of data which fell into each cluster in each area. The distribution of cells into clusters appears to differ by area – for group B1, it comprises 11.0% of units in 7A, 15.3% of units in LIP, 7.7% of units in PFC, and 7.0% of units in V4. N1 comprised of much more units in 7A (29.5%) than the other areas (11.3, 15.6 and 11.8%). C1 comprised of more units in V4 (23.8%) than other areas (12.0, 6.8, 5.4%). L1 seemed to have the largest overall proportion of units, with 22.1% in LIP, 18.4% in PFC, and 16.4% in V4.

3.2.4 Depth Distribution

The distribution of units in each cluster was visually represented in swarmplots, shown in Figure 3.8. Due to lack of data, as well as missing depth data, the shapes of the distribution appears unclear, especially for clusters that had proportionally fewer units in them. Most clusters are missing a proportional number of depth information – the bigger they are, the more unlabeled clusters (plotted at $d = -3$) there are.

From the distributions with more units, clusters 1 and 4 in LIP, clusters 3 and 4 in 7A, clusters 0 and 1 in PFC, and clusters 0 and 5 in V4 appear "pinched" around layer 4 ($d = 4$), and bottom heavy (with a higher distribution of neurons in deep layers). Conversely, cluster 3 in LIP and cluster 3 in PFC are also pinched around layer 4, but appear top heavy (higher distribution in superficial layers). Cluster 5 in PFC and clusters 1 and 2 in V4 appear fatter around layer 4.

3.3 Cell Class & Firing Rate Variation

The percent explained variance (PEV) of the predictability of the block on each cluster's firing rate can be seen in Figure 3.9. Areas LIP and 7A had little to no effect on PEV

between clusters, but PFC showed a small peak in PEV for cluster 6 at sample onset. V4 showed a peak as well, with cluster 0. Cluster 6 had unusually high PEV throughout the time period before and after sample onset. Cluster 0 in V4 was part of group N1, characterized by a narrow waveform, high mean firing rate, and low CV and LV [3.4d]. Cluster 6 in PFC showed similar patterns, albeit with a less narrow waveform [3.4c]. This implies that in these clusters, information about the predictability of the trials in a block is encoded through a high firing rate.

The PEV of the sample image on each cluster's firing rate was also calculated, but was found to have unusually high elevated PEV throughout the trial [3.10a]. Investigating the activity of individual neurons revealed that some neurons would fire indiscriminately during the entire period only in certain sample blocks [3.10b]. After removing trials from predictable blocks from the PEV calculation, the elevated activity disappeared [3.11].

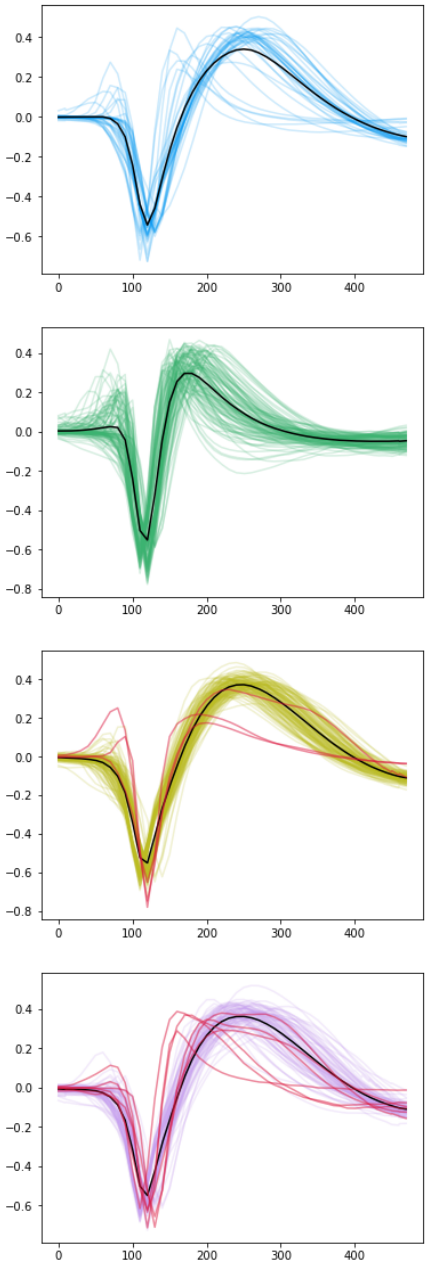
While clusters in 7A didn't have much of an effect on PEV, cluster 3 in LIP, clusters 0 and 6 in PFC, and clusters 0 and 2 in V4 all showed a higher PEV at sample onset. All clusters had a high firing rate, with half belonging to group N1, that similarly implies that sample information is encoded through firing rate in these clusters [3.4]. Notably, cluster 4 in PFC also high firing rate, but had a lower PEV than clusters 0 and 6 [3.4c]. This could be due to it having a higher CV, and thus having a less regular firing process.

3.4 Cell Class & Spike Field Coupling

Preliminary data from spike-field coupling shows synchronization occurring at different times in the trial in different clusters in PFC. The unit from cluster 0 (part of group N1) exhibits highest synchronization with superficial LFP signals throughout trial start, sample onset, and the delay period [3.12a]. The unit from cluster 5 in group L1 exhibits highest synchronization only *before* sample onset [3.12b]. The unit from cluster 7 in group B1 exhibits highest synchronization before and after sample onset, but not during [3.12c]. While this shows

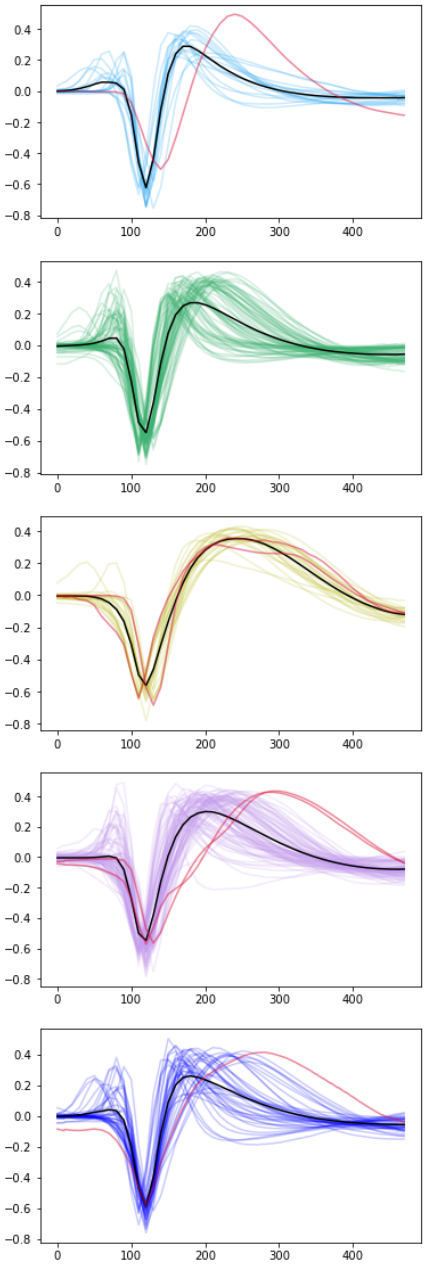
potential for different periods of synchrony in different clusters during prediction periods, the data is non-aggregate and isolated, so further analysis would need to be made.

7A cluster waveforms

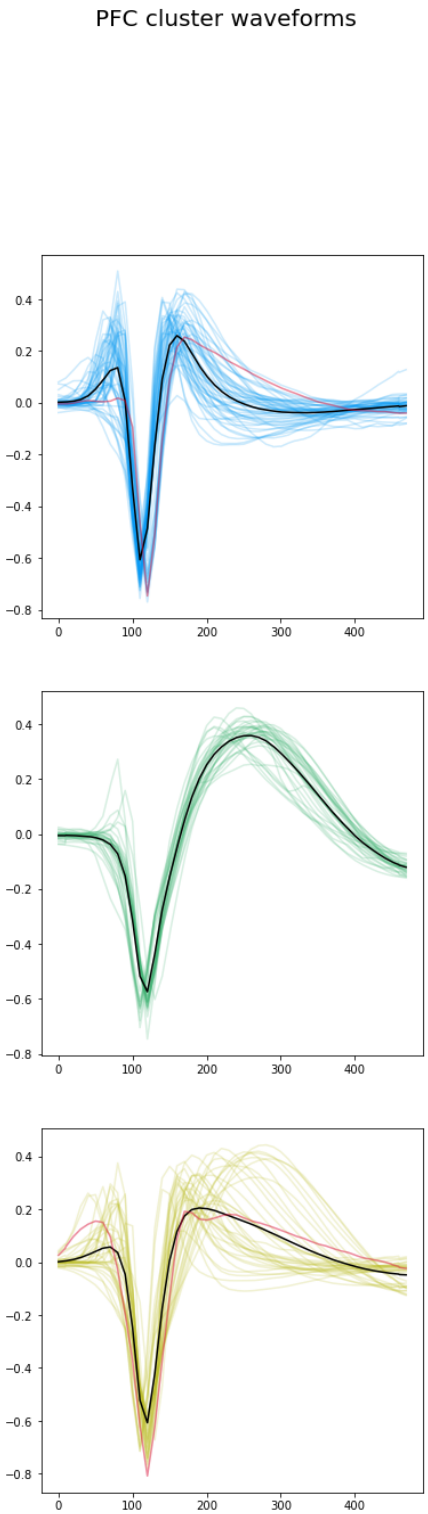


(a)

V4 cluster waveforms

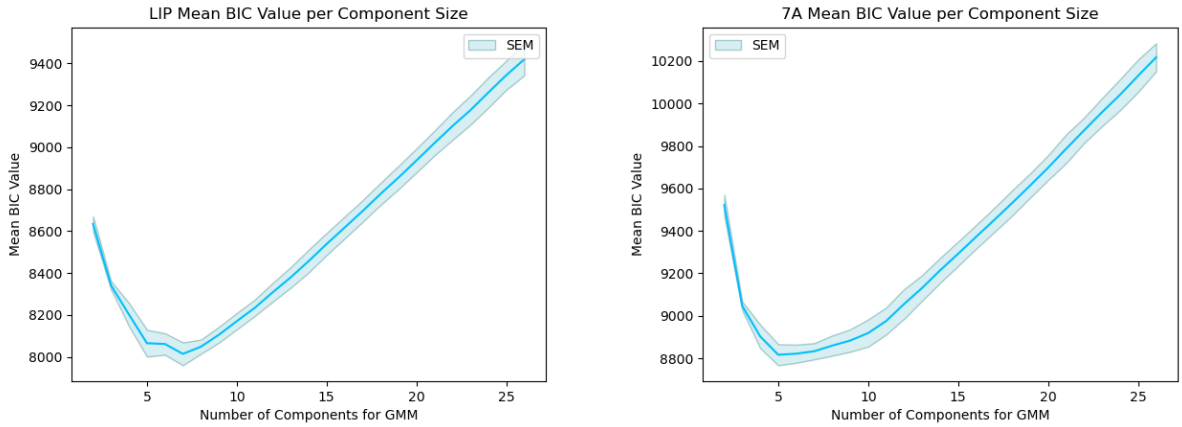


(b)



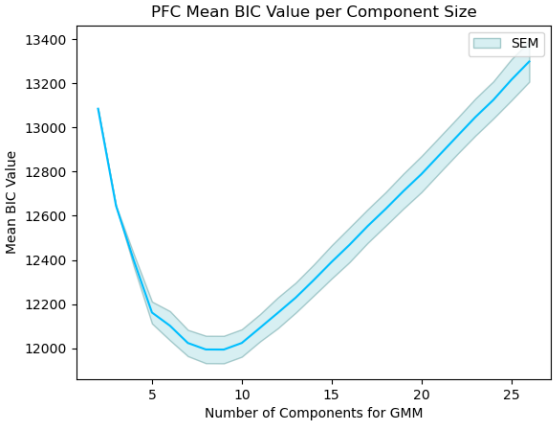
(c)

Figure 3.1: Average waveforms per preliminary clustering area. Outlier waveforms shown in red. Average waveform shown in black.
(a) 7A ; (b) PFC ; (c) V4

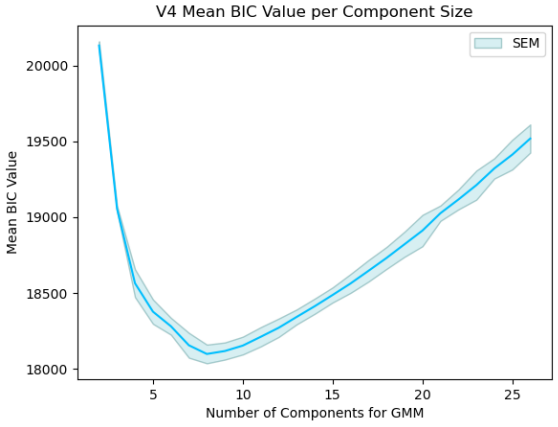


(a)

(b)



(c)



(d)

Figure 3.2: Mean BIC values and SEM values (shown as blue highlight) for each area (a) LIP ; (b) 7A ; (c) PFC ; (d) V4

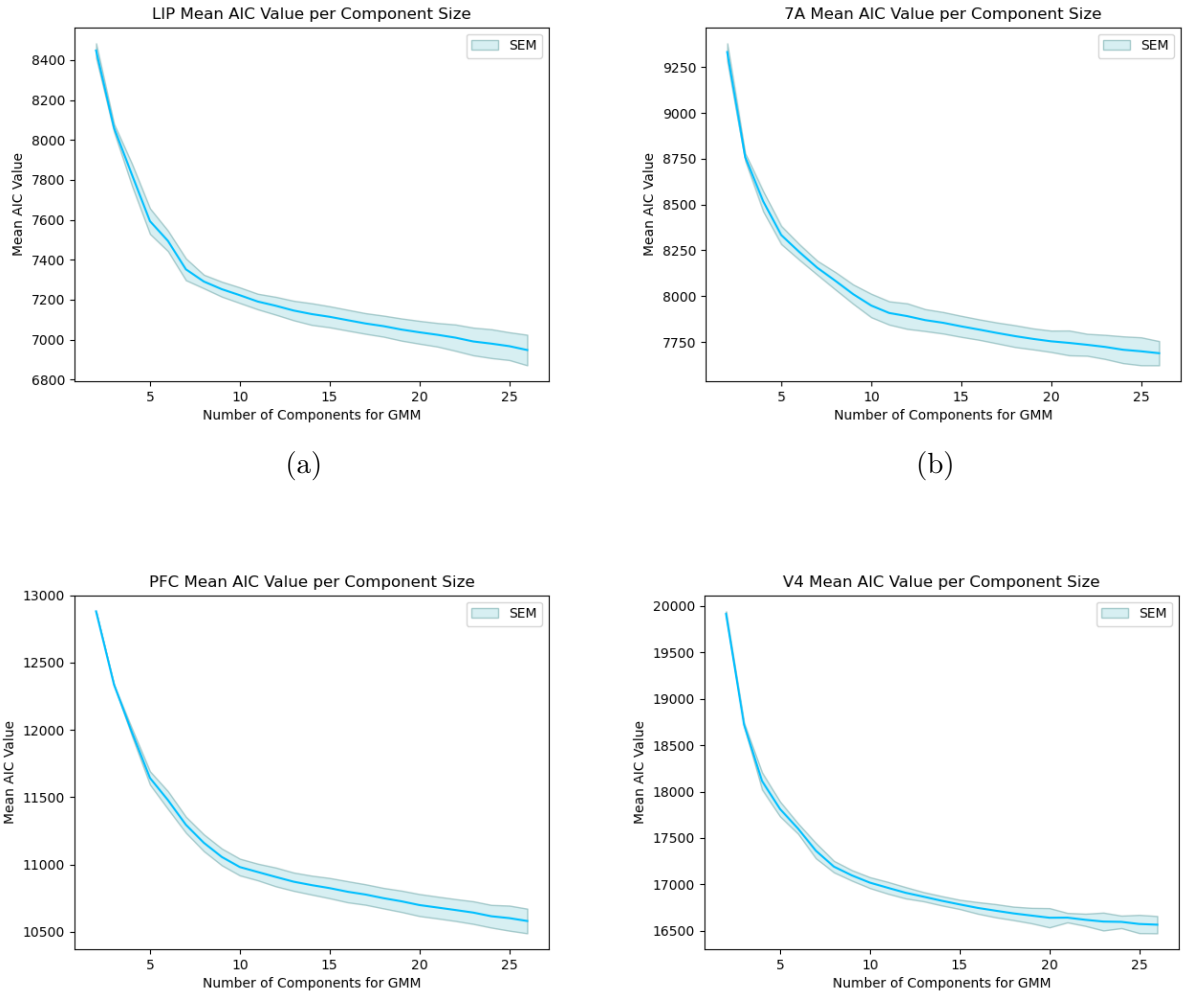
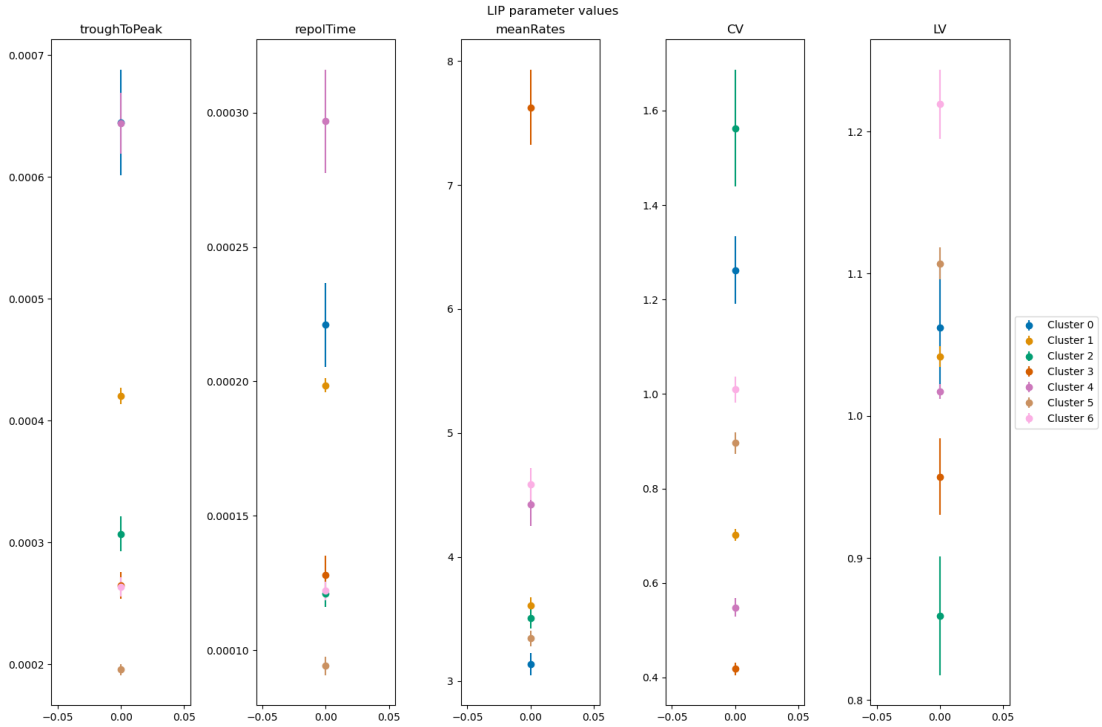
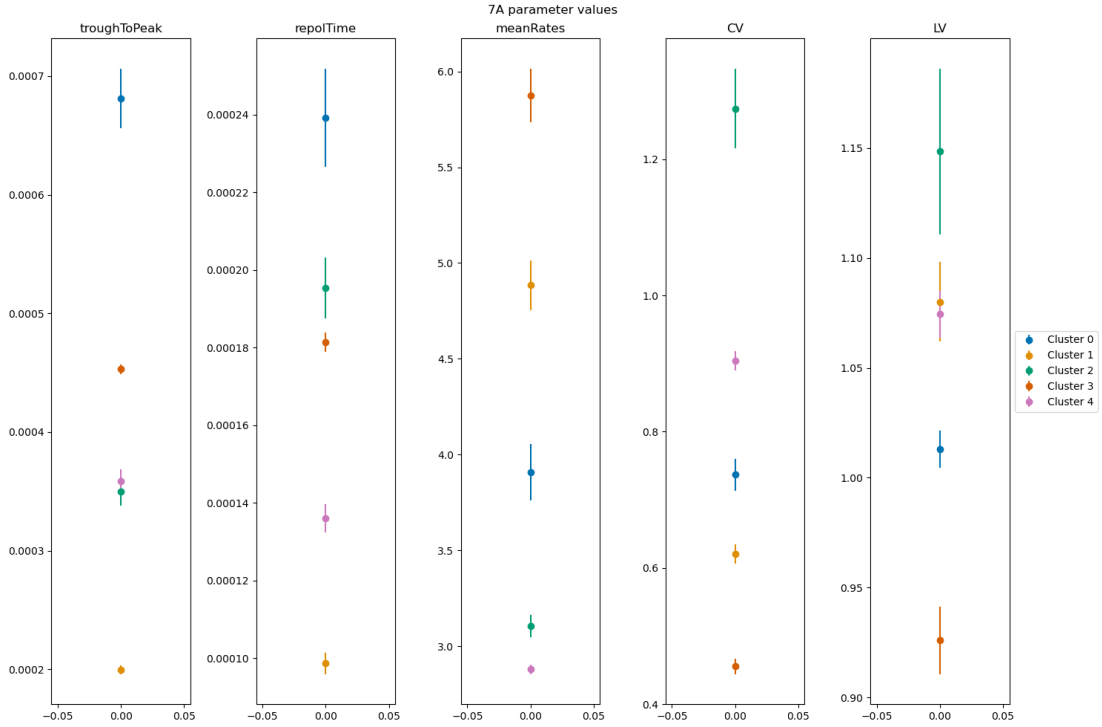


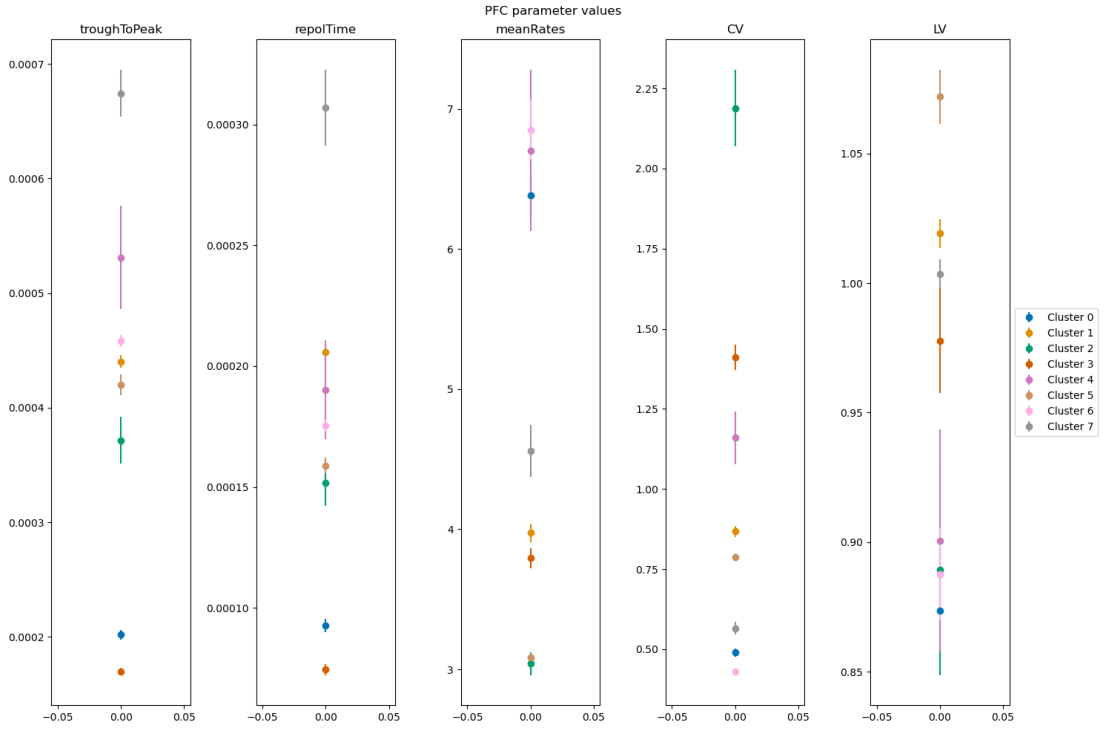
Figure 3.3: Mean AIC values and SEM values (shown as blue highlight) for each area (a) LIP ; (b) 7A ; (c) PFC ; (d) V4


LIP parameter values
troughToPeak
repolTime
meanRates
CV
LV
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6

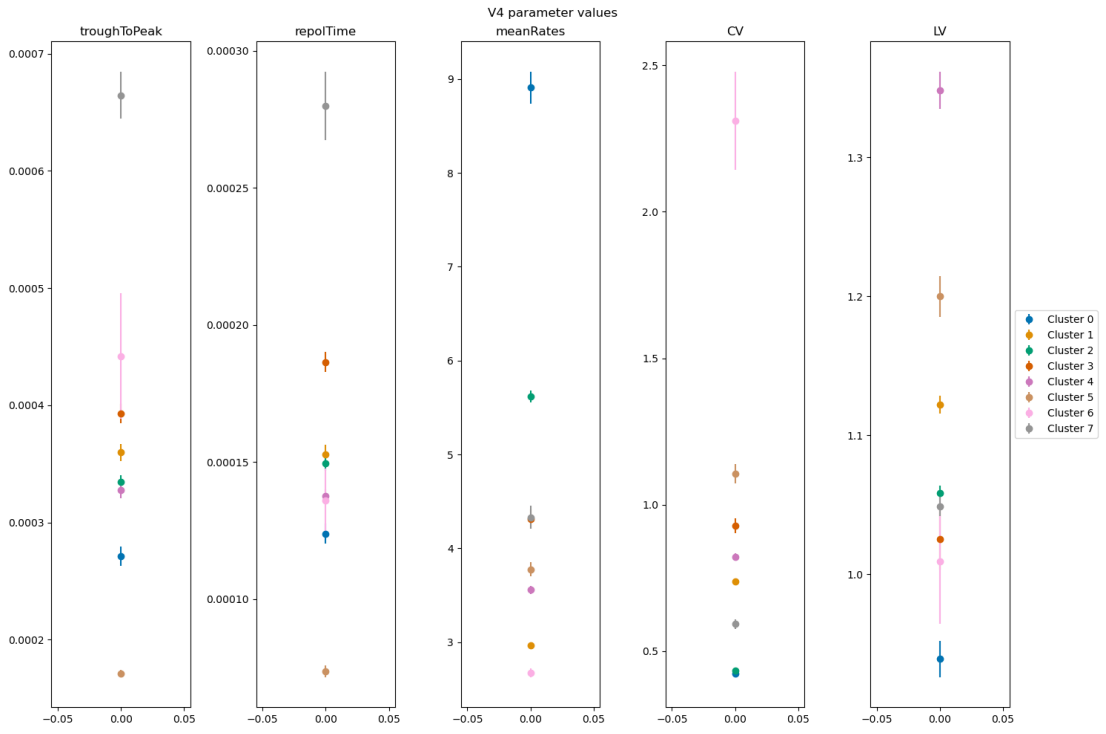
(a)


7A parameter values
troughToPeak
repolTime
meanRates
CV
LV
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4

(b)

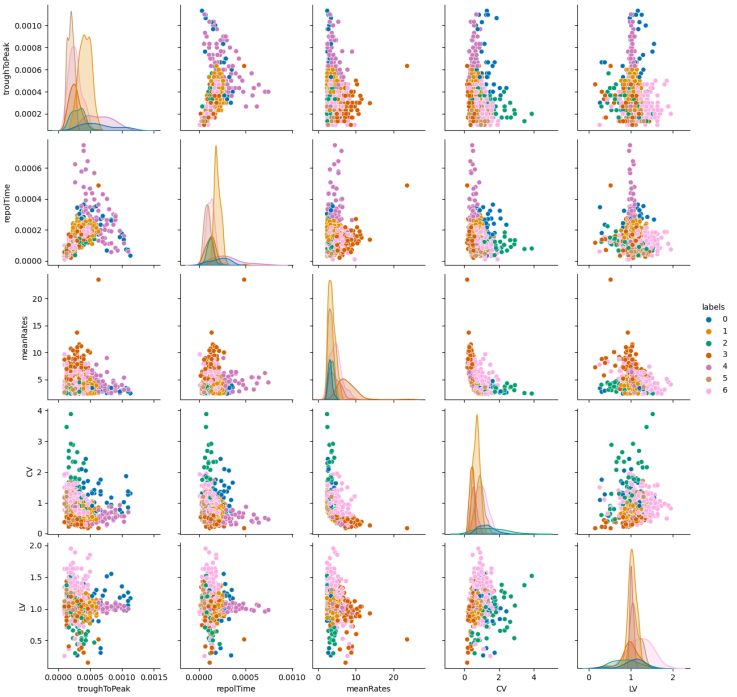


(c)

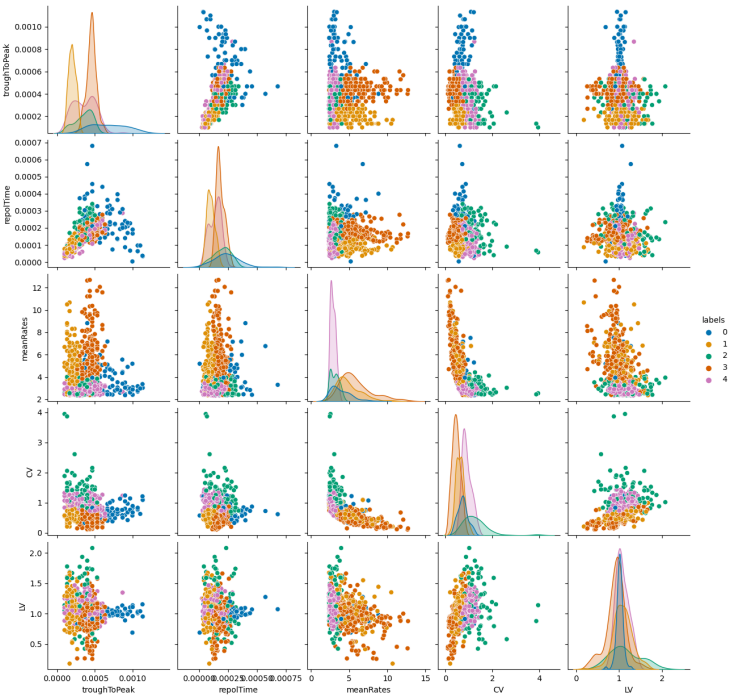


(d)

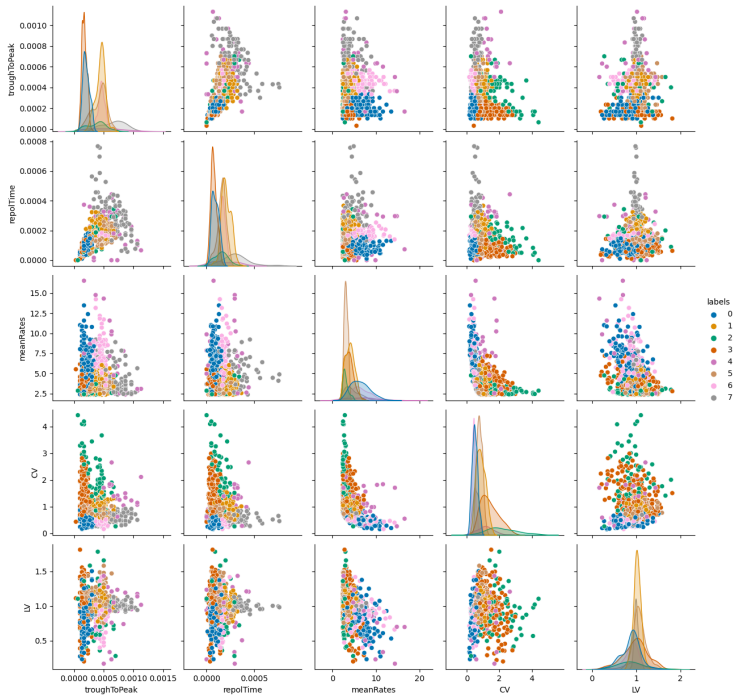
Figure 3.4: Five parameter values defining each cell class in areas (a) LIP ; (b) 7A ; (c) PFC ; (d) V4


troughToPeak
repolTime
meanRates
CV
LV
labels
0
1
2
3
4
5
6

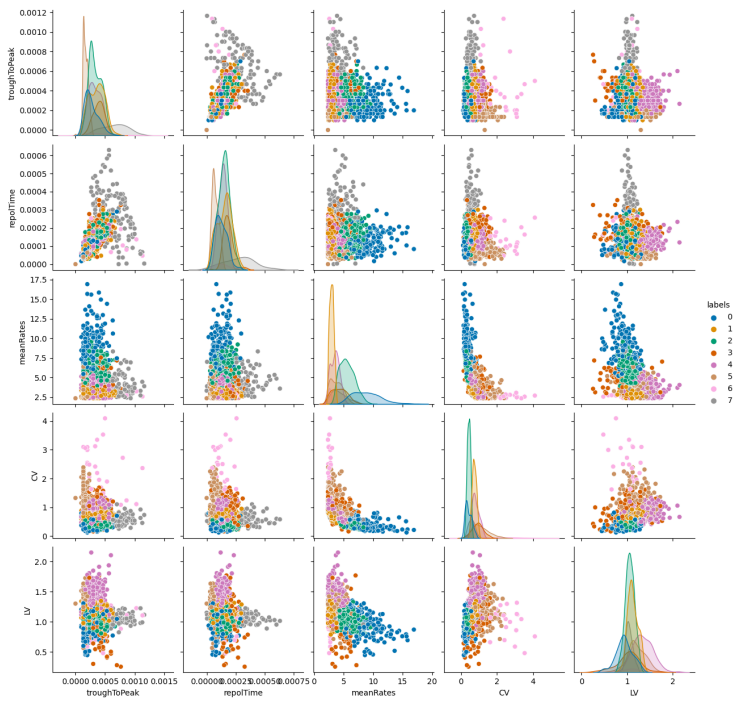
(a)


troughToPeak
repolTime
meanRates
CV
LV
labels
0
1
2
3
4

(b)

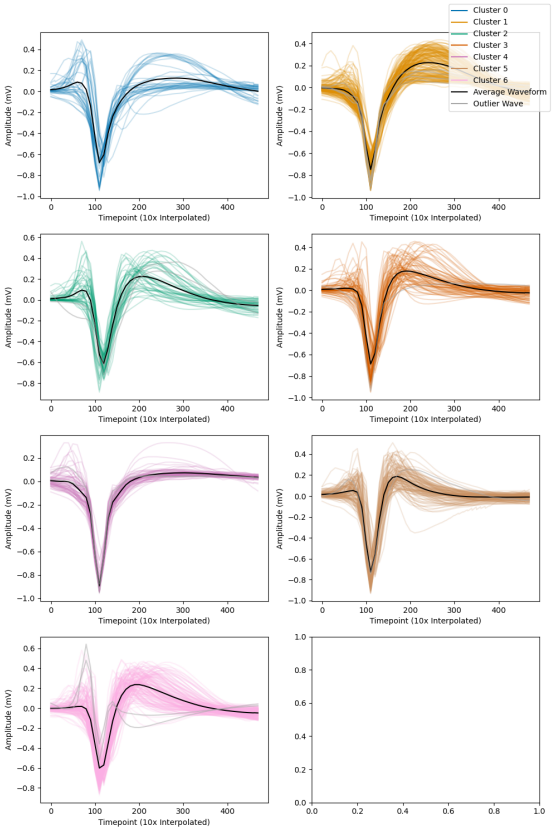


(c)

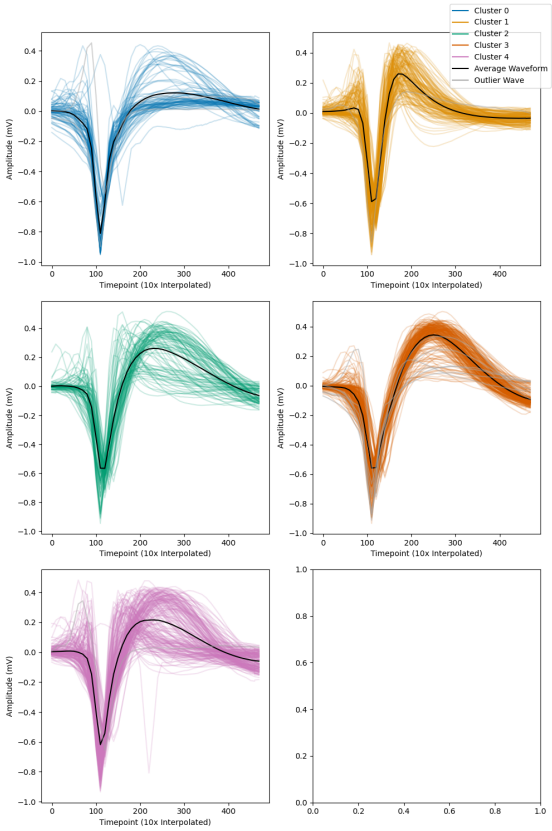


(d)

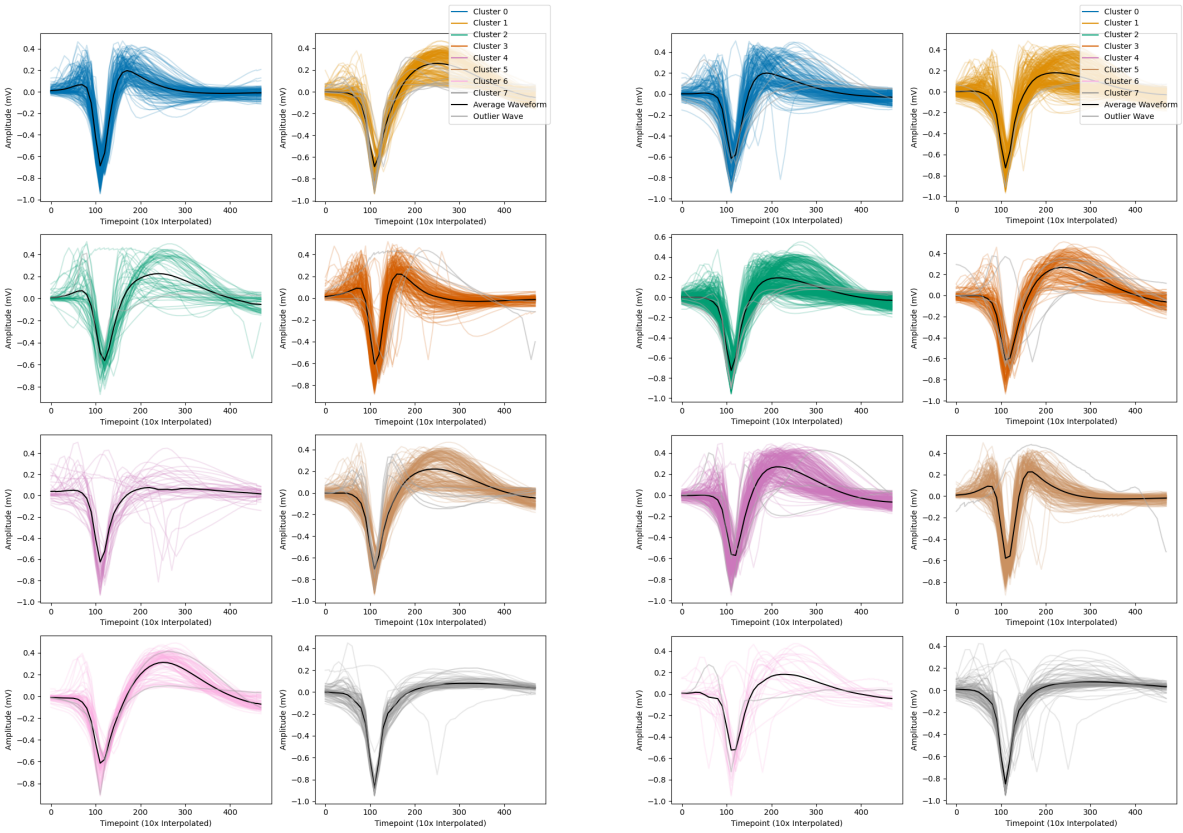
Figure 3.5: Visualization of clusters in the 2D space of each pair of features. Marginal distribution of each feature on the diagonal.
(a) LIP ; (b) 7A ; (c) PFC ; (d) V4


Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Cluster 6
Average Waveform
Outlier Wave
Amplitude (mV)
Timepoint (10x Interpolated)

(a)

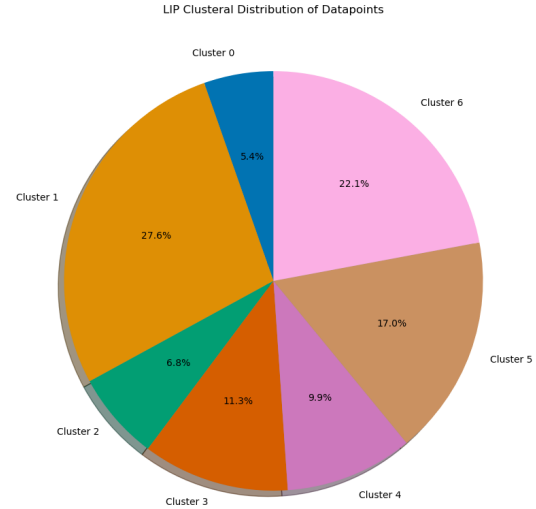

Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Average Waveform
Outlier Wave
Amplitude (mV)
Timepoint (10x Interpolated)

(b)

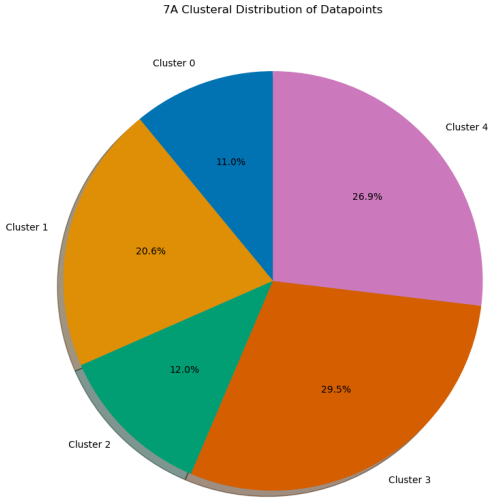


(c) (d)

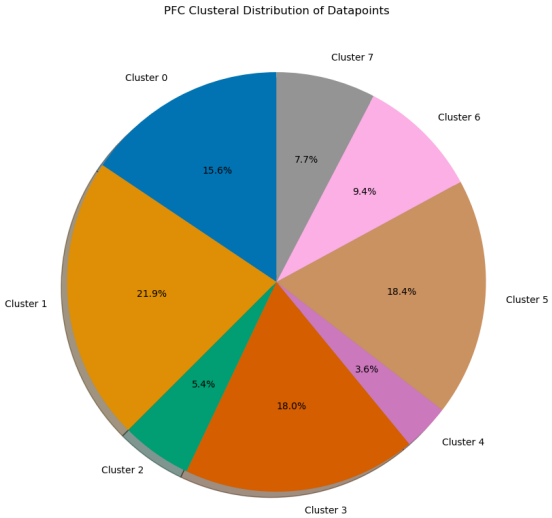
Figure 3.6: Average waveforms per preliminary clustering area. Outlier waveforms shown in light grey. Average waveform shown in black.
(a) LIP ; (b) 7A ; (c) PFC ; (d) V4



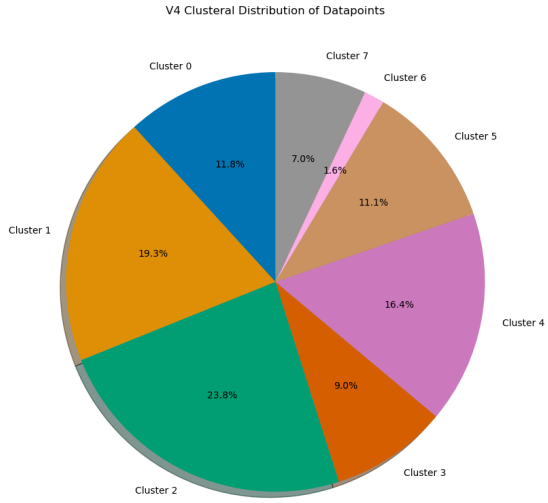
(a)



(b)

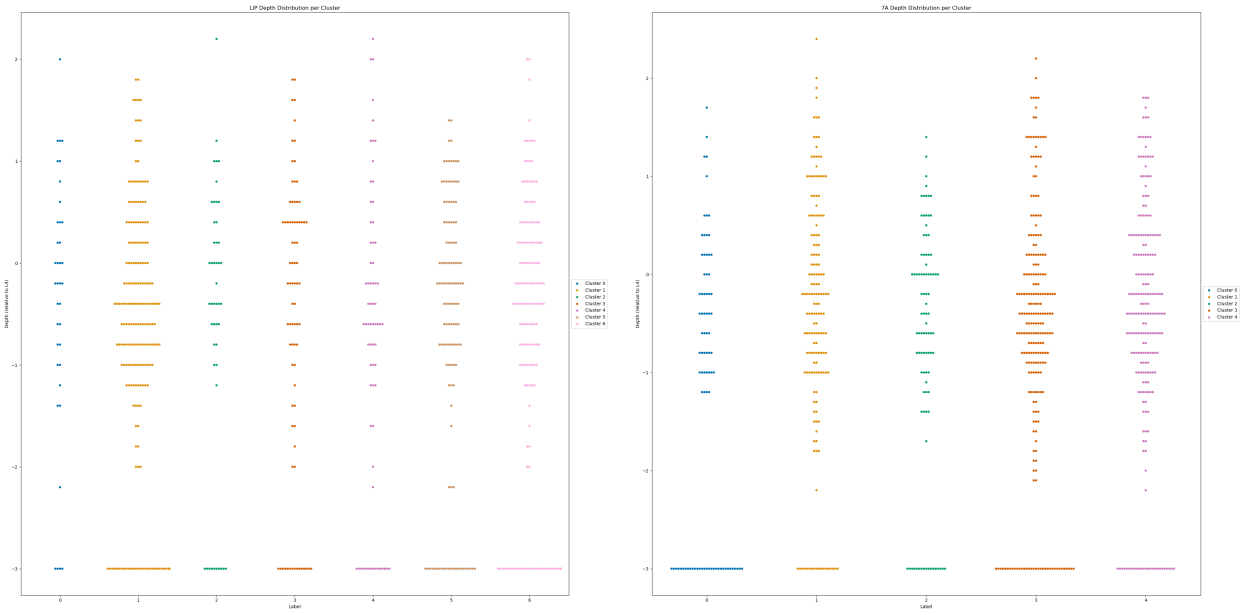


(c)

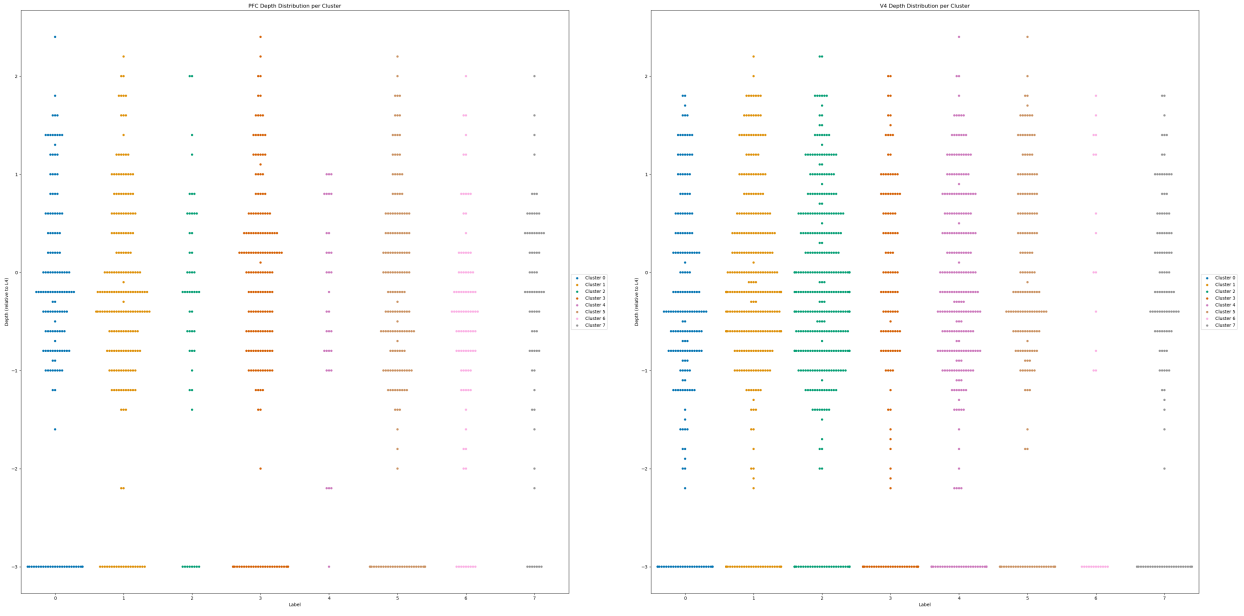


(d)

Figure 3.7: Percentage of data points in each cluster for areas (a) LIP ; (b) 7A ; (c) PFC ; (d) V4

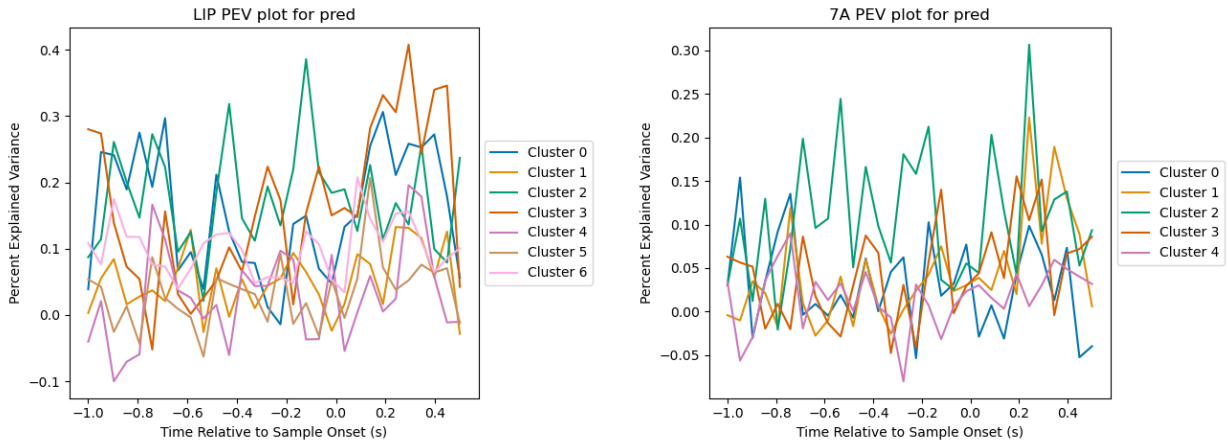


(a) (b)



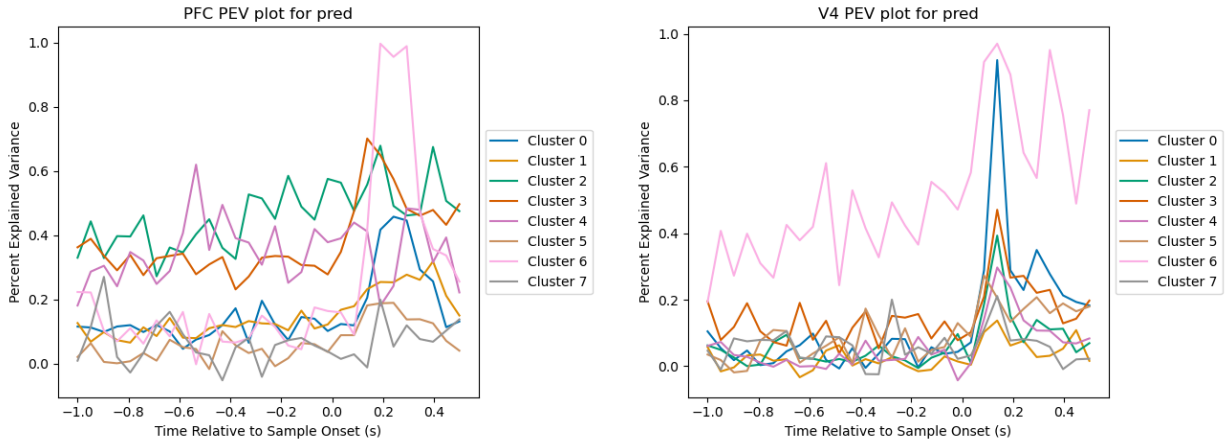
(c) (d)

Figure 3.8: Depth distribution for clusters in areas
(a) LIP ; (b) 7A ; (c) PFC ; (d) V4



(a)

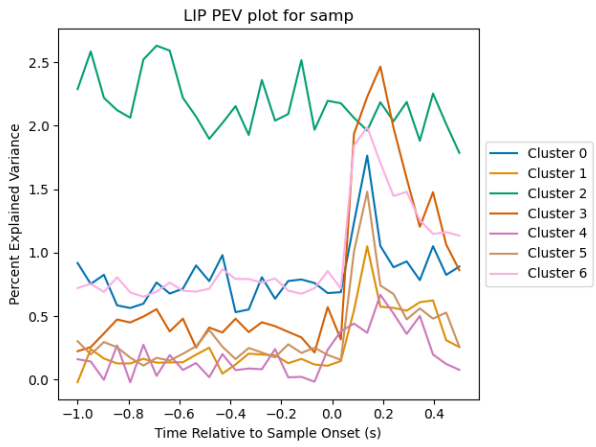
(b)



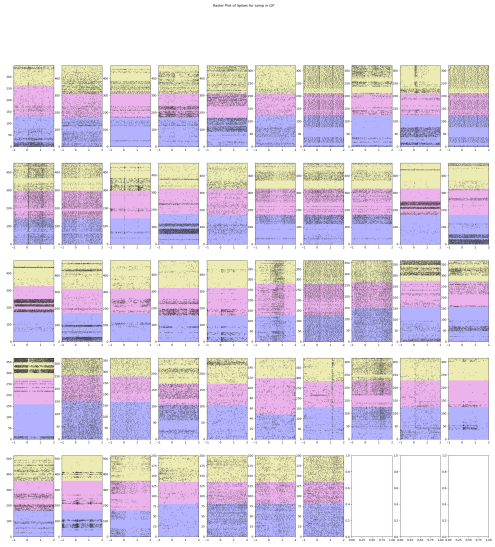
(c)

(d)

Figure 3.9: Percent explained variance of trial predictability on cluster firing rate (a) LIP ; (b) 7A ; (c) PFC ; (d) V4

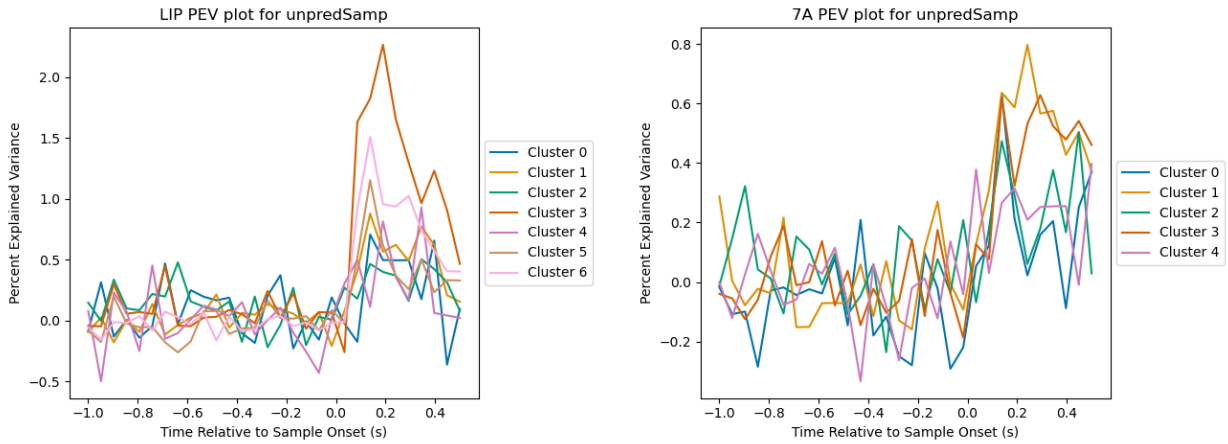


(a)



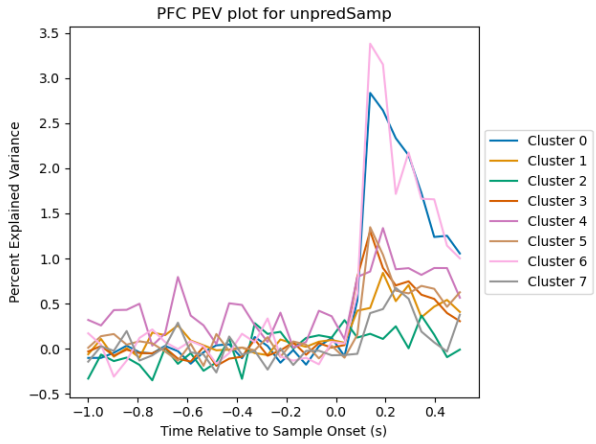
(b)

Figure 3.10: (a) PEV of each cluster on sample type in LIP ; (b) Raster plot of neuronal activity over different sample trials in LIP

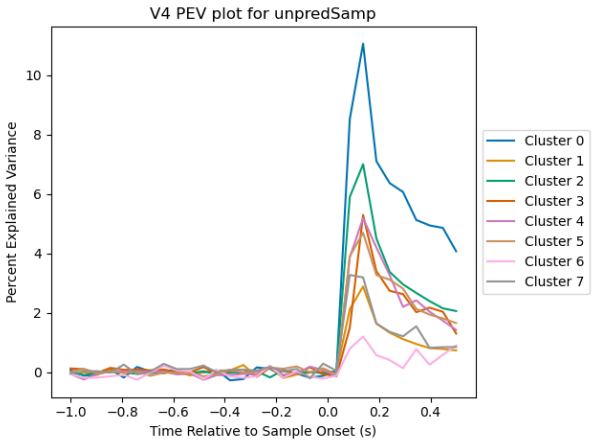


(a)

(b)

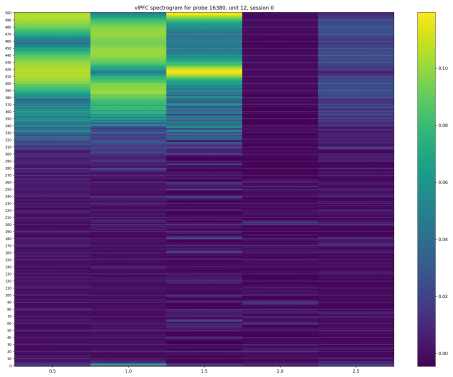


(c)

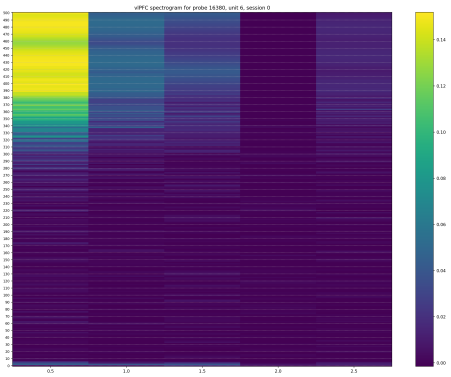


(d)

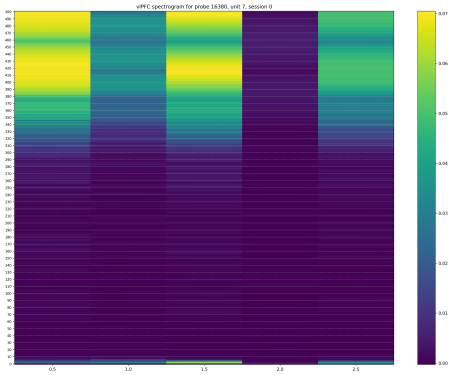
Figure 3.11: Percent explained variance of trial sample type on cluster firing rate, with predictable trial blocks removed
(a) LIP ; (b) 7A ; (c) PFC ; (d) V4



(a)



(b)



(c)

Figure 3.12: Spike-field coupling synchrony of select PFC units with superficial LFP signals (a) Cluster 0 unit (Group N1) ; (b) Cluster 5 unit (Group L1) ; (c) Cluster 7 unit (Group B1)

Chapter 4

Discussion

4.1 Distinct cell classes

Using data in awake macaque monkeys performing a working memory task, five distinct cell classes were found in area 7A, 7 in area LIP, and 8 in areas PFC and V4. These cluster sizes are similar to previous studies, though more varied due to separating the data by area. While the size of the number of cell classes differed between areas, there were certain classes that seemed to be correlates of each other across areas. Identified in this project were group B1 (broadest waveforms, low firing rates, and semi-regular spiking), group N1 (narrowest waveforms, high firing rates, and regular spiking), group C1 (narrower/variable waveforms, low firing rates, non-bursty, but most irregular spiking), and group L1 (variable waveforms, lower firing rates, regular spiking, high burstiness). While these were identified comparing trends in mean feature values across areas, a more robust, computational method is needed to see if these cell classes across areas are actually the same electrophysiological phenotype. This would also be beneficial for identifying the defining features of each class more thoroughly. A potential strategy would be to try validating the results with other unsupervised learning methods, such as the simpler K-means clustering. While GMM is good for handling complex, non-linear data, the results can be harder to interpret and parse.

Using a simpler algorithm to see the comparison in results could offer insight into the clusters given by the GMM.

4.2 Functional and distributional differences between cell classes

Different clusters were shown to be functionally distinct from another, through analysis of the differing firing rate for each cluster due to the difference in sample type, or predictability of the trial. Group N1 and clusters with similar attributes were able to explain variance in predictability in V4 and PFC, as well as sample type in LIP, PFC and V4. This implies that group N1 does a lot of the information coding during mentally demanding working memory tasks such as the DMTS task, in both aspects of differing sample and in prediction. Preliminary spike-field coupling data also showed promise for differing synchrony periods between different cell classes. More investigation into different measures of function would help distinguish the roles of the classes even further. While N1 seems to be important in information coding, it would be interesting to see what role the other classes play, if any at all. It's possible that there's not a lot of functional information in certain areas, such as 7A, which showed little to no effect of both sample type and predictability on any class' firing rate.

Clusters were also shown to be variable in both distribution across areas and depths. In some areas, certain groups would be much more prominent in ratio than others, such as N1 in 7A, or C1 in V4. Cluster depth distribution revealed patterns in certain clusters that were more prevalent in deep or superficial layers. The correlation between depth preference and any functional or informational differences between clusters should be investigated.

4.3 Correlates between classification methods

While it's hard to tell if all clusters have an identifiable morphological or molecular correlate, the characteristics of groups B1 and N1, which are probably the closest to the over-arching broad-spiking and narrow-spiking cell classes identified in previous studies, appear to be similar to regular-spiking pyramidal cells and fast-spiking GABAergic chandelier cells [13]. However, there are certain discrepancies that suggest a bigger picture – for example, pyramidal cells are known to make up around 70-85% of all neurons in the brain [14], but the percentages of datapoints the B1 cell classes made up out of all datapoints were all less than 15%. This might be due to the distribution of probes, and where they recorded from, but it seems suspect. As for the other cell classes, which are less easily separable, it may be to match it up with a previous cell class identified through morphological or molecular methods before more stringently pinning down each class' identifying features.

References

[1] S. R. Cajal, N. Swanson, and L. Swanson, *Histology of the nervous system, vol. 1*, 1995.

[2] H. Zeng and J. R. Sanes, "Neuronal cell-type classification: Challenges, opportunities and the path forward," *Nature Reviews Neuroscience*, vol. 18, no. 9, pp. 530–546, 2017.

[3] S. Ardid, M. Vinck, D. Kaping, S. Marquez, S. Everling, and T. Womelsdorf, "Mapping of functionally characterized cell classes onto canonical circuit operations in primate prefrontal cortex," *Journal of Neuroscience*, vol. 35, no. 7, pp. 2975–2991, 2015, ISSN: 0270-6474. [Online]. Available: https://www.jneurosci.org/content/35/7/2975.

[4] C. Trainito, C. von Nicolai, E. K. Miller, and M. Siegel, "Extracellular spike waveform dissociates four functionally distinct cell classes in primate cortex," *Current Biology*, vol. 29, no. 18, pp. 2973–2982, 2019.

[5] K. Banaie Boroujeni, P. Tiesinga, and T. Womelsdorf, "Interneuron-specific gamma synchronization indexes cue uncertainty and prediction errors in lateral prefrontal and anterior cingulate cortex," *eLife*, vol. 10, S. Haegens and M. J. Frank, Eds., e69111, Jun. 2021, ISSN: 2050-084X. [Online]. Available: https://doi.org/10.7554/eLife.69111.

[6] C. Constantinidis and P. S. Goldman-Rakic, "Correlated discharges among putative pyramidal neurons and interneurons in the primate prefrontal cortex," *Journal of neurophysiology*, vol. 88, no. 6, pp. 3487–3497, 2002.

[7] E. K. Lee, H. Balasubramanian, A. Tsolias, S. U. Anakwe, M. Medalla, K. V. Shenoy, and C. Chandrasekaran, "Non-linear dimensionality reduction on extracellular waveforms reveals cell type diversity in premotor cortex," *Elife*, vol. 10, e67490, 2021.

[8] M. Dasilva, C. Brandt, S. Gotthardt, M. A. Gieselmann, C. Distler, and A. Thiele, "Cell class-specific modulation of attentional signals by acetylcholine in macaque frontal eye field," *Proceedings of the National Academy of Sciences*, vol. 116, no. 40, pp. 20 180–20 189, 2019.

[9] K. Banaie Boroujeni, M. Oemisch, S. A. Hassani, and T. Womelsdorf, "Fast spiking interneuron activity in primate striatum tracks learning of attention cues," *Proceedings of the National Academy of Sciences*, vol. 117, no. 30, pp. 18 049–18 058, 2020.

[10] J. Gu, "Comparative analysis based on clustering algorithms," in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1994, 2021, p. 012 024.

[11] J. Jones, "Clustering: Out of the black box," *Medium*, 2021. [Online]. Available: https://towardsdatascience.com/clustering-out-of-the-black-box-5e8285220717.

[12] S. Brincat, *Spynal*, version 0.1.2, Sep. 2023. DOI: 10.5281/zenodo.8346152. [Online]. Available: https://github.com/sbrincat/spynal.

[13] L. S. Krimer, A. V. Zaitsev, G. Czanner, S. Kroner, G. González-Burgos, N. V. Povysheva, S. Iyengar, G. Barrionuevo, and D. A. Lewis, "Cluster analysis–based physiological classification and morphological properties of inhibitory neurons in layers 2–3 of monkey dorsolateral prefrontal cortex," *Journal of neurophysiology*, vol. 94, no. 5, pp. 3009–3022, 2005.

[14] Y. Wang, M. Ye, X. Kuang, Y. Li, and S. Hu, "A simplified morphological classification scheme for pyramidal cells in six layers of primary somatosensory cortex of juvenile rats," *IBRO reports*, vol. 5, pp. 74–90, 2018.

[15] E. Huang, *Clustering*, Feb. 2024. [Online]. Available: https://github.com/ehuas/clustering.

Appendix A

Code listing

Below is all code written for this project, separated by file/ function. Access to the repository containing this code (https://github.com/ehuas/clustering) can be given upon request [15]. Minor adjustments for clean up and documentation may have been made since the writing of this thesis.

A.1 Loading Data

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 30 13:25:19 2022

@author: huange
"""
from spynal.matIO import loadmat
import numpy as np
import pandas as pd
from analysis import *
from preprocessing import preProcessing, featExtract, coupling
import os

def select_area(unit_info, data, area_name):
    '''
    Isolates data recorded from a select area.

        Input: unit_info: dataframe containing unit recording
           information, such as area recorded
```

```
                spike_times (n_units, n_timepts): ndarray of spike
                    timestamps
                spike_waves (n_units, n_timepts): ndarray of spike
                    waveforms
                area_name: area of interest

        Output: spike times, waves of select area

    '''
    areas = unit_info['area'].to_numpy()
    area_idx = np.where(areas == area_name)[0]

    data = data[:, area_idx]

    return data, area_idx

def shape_data(spike_times, spike_waves = None):

    for i in range(spike_times.shape[0]):
       for j in range(spike_times.shape[1]):
           spike = spike_times[i,j]
           if spike_waves is not None:
                wave = spike_waves[i,j]
                if len(wave.shape) < 2:
                    wave = np.expand_dims(wave, axis = 1)
           if type(spike) != float:
               if np.size(spike) != 0:
                   trunc_spike = np.where((-1 < spike) & (spike < 2))
                       [0]

                   spike_times[i,j] = np.atleast_1d(spike[trunc_spike
                       ])

                   if spike_waves is not None:
                       trunc_wave = wave[:, trunc_spike]
                       spike_waves[i,j] = np.atleast_2d(trunc_wave)
               else:
                   spike_times[i, j] = []

                   if spike_waves is not None:
                       spike_waves[i,j] = np.empty((48,0))
           else:
               if -1 < spike < 2:
                   spike_times[i,j] = [spike]

                   if spike_waves is not None:
                       spike_waves[i,j] = np.expand_dims(wave, 1)
               else:
                   spike_times[i,j] = []

                   if spike_waves is not None:
                       spike_waves[i,j] = np.empty((48, 0))

```

```
69      return spike_times, spike_waves
70
71  def load_data(path):
72      spike_times, spike_times_schema, unit_info, trial_info,
           session_info, spike_waves, spike_waves_schema = \
73      loadmat(path,
74          variables=['spikeTimes','spikeTimesSchema','unitInfo','
               trialInfo', 'sessionInfo', 'spikeWaves', 'spikeWavesSchema
               '],
75          typemap={'unitInfo':'DataFrame', 'trialInfo':'DataFrame'})\
76
77      shape_spikes, shape_waves = shape_data(spike_times, spike_waves =
           spike_waves)
78
79      return shape_spikes, spike_times_schema, unit_info, trial_info,
           session_info, shape_waves, spike_waves_schema
80
81  def load_osc_data(path):
82      lfp, lfp_schema, electrode_info, spike_times, unit_info = \
83      loadmat(path,
84          variables=['lfp', 'lfpSchema', 'electrodeInfo', 'spikeTimes',
               'unitInfo'],
85          typemap={'electrodeInfo':'DataFrame', 'unitInfo':'DataFrame'
               })\
86
87      spike_times, _ = shape_data(spike_times)
88
89      return lfp, lfp_schema, electrode_info, spike_times, unit_info
90
91  def concat_sessions(paths, area):
92      comb = pd.DataFrame(columns=['meanRates', 'troughToPeak', '
           repolTime', 'CV', 'LV'])
93      PEV_samp_concat = np.empty((0, 30))
94      PEV_pred_concat = np.empty((0, 30))
95      align_waves_concat = np.empty((470, 0))
96      depths_concat = np.empty((0, ))
97      pred_concat = np.empty((0, ))
98      samp_concat = np.empty((0, ))
99      unit_count = 0
100
101     PEV_unpredSamp_concat = np.empty((0, 30))
102
103     for path in paths:
104         spike_times, _, unit_info, trial_info, session_info,
               spike_waves, spike_waves_schema = load_data(path)
105         area_spike_times, area_idx = select_area(unit_info,
               spike_times, area)
106         area_spike_waves = spike_waves[:, area_idx]
107
108         validTrials, validNeurons, meanRates, ISIs, meanAlignWaves,
               smpRate, rates, _, _, _, predInfo, sampInfo, depths, _, _
               = \
109             preProcessing(area_spike_times,
```

```
                            trial_info,
                            session_info,
                            area_spike_waves,
                            spike_waves_schema,
                            unit_info,
                            area,
                            unit_count) \

        if validTrials.size < 2 or len(validNeurons) < 2:
            pass
        else:
            unit_count += len(validNeurons)
            features = featExtract(meanRates, ISIs, meanAlignWaves,
               smpRate, rates)
            comb = pd.concat([comb, features], ignore_index=True)
            #pev_samp = pev_func(rates, sampInfo)
            pev_pred = pev_func(rates, predInfo)

            #PEV_samp_concat = np.concatenate((PEV_samp_concat, np.
               squeeze(pev_samp, axis=0)), axis = 0)
            PEV_pred_concat = np.concatenate((PEV_pred_concat, np.
               squeeze(pev_pred, axis=0)), axis = 0)

            # pev_unpredSamp = pev_func(rates[trial_trials, :, :],
               unpredSampInfo)
            # PEV_unpredSamp_concat = np.concatenate((
               PEV_unpredSamp_concat, np.squeeze(pev_unpredSamp, axis
               =0)), axis = 0)

            align_waves_concat = np.concatenate((align_waves_concat,
               meanAlignWaves), axis = 1)

            depths_concat = np.concatenate((depths_concat, depths),
               axis = 0)

            pred_concat = np.concatenate((pred_concat, predInfo.
               to_numpy()), axis = 0)
            samp_concat = np.concatenate((samp_concat, sampInfo.
               to_numpy()), axis = 0)

    return comb, PEV_pred_concat, align_waves_concat, depths_concat,
       pred_concat, samp_concat


def osc_concat(paths, areas):
    session = 0
    for path in paths:
        lfp, lfp_schema, electrode_info, spike_times, unit_info =
           load_osc_data(path)

        for area in areas:
            if unit_info['area'].isin([area]).any():
```

```
                area_lfp, area_idx = select_area(electrode_info, lfp,
                    area)
                lfp_trunc = area_lfp[1000:4001, :, :]
                area_spikes = spike_times[:, area_idx]

                smp_rate = lfp_schema['smpRate']

                if '/mnt/common/datasets/wmPredict/mat/mainTask' in
                   path:
                    depth_var = 'betaGammaDepth'
                else:
                    depth_var = 'laminarDepth'

                coupling(lfp_trunc, area_idx, depth_var,
                   electrode_info, unit_info, area_spikes, area,
                   session, smp_rate)

        session += 1

def main():
    directories = ['/mnt/common/datasets/wmPredict/mat/mainTask', '/
       mnt/common/scott/laminarPharm/mat']
    paths = []
    for directory in directories:
        for filename in os.listdir(directory):
            if filename == 'laminarPharm_databases.mat' or filename
               == 'spikesOnly' or filename == 'wmPredict_databases.
               mat':
                pass
            else:
                f = os.path.join(directory, filename)
                paths.append(f)

    areas = ['vlPFC', 'dlPFC', '7A', 'V4', 'LIP']

        #comb, PEV_pred, waves, depths, pred, samp = concat_sessions(
           paths, area)

    osc_concat(paths, areas)

        # unpredSamp_df = pd.DataFrame(PEV_samp)
        # unpredSamp_df.to_csv('/home/ehua/clustering/090623_data/{}
           _PEV_unpredSamp.csv'.format(area))

        # df = pd.DataFrame(comb)
        # df.to_csv('/home/ehua/clustering/090623_data/{}_df.csv'.
           format(area))

        # waves_df = pd.DataFrame(waves)
        # waves_df.to_csv('/home/ehua/clustering/090623_data/{}_waves
           .csv'.format(area))

        # samp_df = pd.DataFrame(PEV_samp)
```

```
        # samp_df.to_csv('/home/ehua/clustering/090623_data/{}
            _PEV_samp.csv'.format(area))

        # pred_df = pd.DataFrame(PEV_pred)
        # pred_df.to_csv('/home/ehua/clustering/090623_data/{}
            _PEV_pred.csv'.format(area))

        # depths_df = pd.DataFrame(depths)
        # depths_df.to_csv('/home/ehua/clustering/090623_data/{}
            _depths_jitter.csv'.format(area))

        # pred_df = pd.DataFrame(pred)
        # pred_df.to_csv('/home/ehua/clustering/090623_data/{}_pred.
            csv'.format(area))

        # samp_df = pd.DataFrame(samp)
        # samp_df.to_csv('/home/ehua/clustering/090623_data/{}_samp.
            csv'.format(area))


if __name__ == "__main__":
    main()
```

A.2 Pre-processing

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 27 22:52:30 2022

@author: huange
"""
from spynal import spikes, utils, sync, spectra
import numpy as np
import pandas as pd
from utils import *
import copy
import math
import matplotlib.pyplot as plt

def trialSelection(trial_info, session_info):
    '''
    Selects valid trials from data.

        Input: trialInfo (n_trials, n_variables) DataFrame for single
            session
        Output: (valid_trials,) vector of indices of trials to keep

    Valid trials are defined as:
```

```
        (a) correct and
        (b) <= 5 trials before drug injection onset

    '''
    ### isolates only trials that are <= 5 trials before drug onset
    if 'drugStartTrial' in session_info:
        drugStartTrial = session_info['drugStartTrial']
        beforeDrugTrials = trial_info.loc[trial_info['trial'] <=
           drugStartTrial-5]
        trials_df = beforeDrugTrials.loc[beforeDrugTrials['correct']]
        trials_keep = np.where(beforeDrugTrials['correct'])[0]
    else:
        trials_keep = np.where(trial_info['correct'])[0]
        trials_df = trial_info.loc[trial_info['correct']]

    sampInfo = copy.deepcopy(trials_df['sample'])

    ### identifies the non predictable and predictable trials
    block_trials = np.where(trials_df['blockType'] == 'block')[0]
    trial_trials = np.where(trials_df['blockType'] == 'trial')[0]

    ### sets block types to 1, trial types to 0
    trials_df.loc[trials_df['blockType'] == 'block'] = 1
    trials_df.loc[trials_df['blockType'] == 'trial'] = 0
    predInfo = copy.deepcopy(trials_df['blockType'])
    unpredSampInfo = sampInfo[trial_trials]

    return trials_keep, predInfo, sampInfo, block_trials,
       trial_trials, unpredSampInfo

def trialsKeep(trials_keep, spike_times, spike_waves):
    '''
    Filters out non-valid trials in given time, wave data
    '''
    times_trials = spike_times[trials_keep, :]
    waves_trials = spike_waves[trials_keep, :]

    return times_trials, waves_trials

def neuronSelection(times_trials):
    '''
    Selects valid neurons from data.

        Input: spikeTimes (n_trials, n_units) object array for single
            session
        Output: (n_units,) bool vector indicating which units to keep

    Valid neurons are defined as:
        (a) having an overall mean spike rate > 1 spike/s (weeding
           out silent cells) and
        (b) having < 0.1% ISIs within 1 ms (weeding out poorly
           isolated single neurons)

```

```
    '''
    rates, timepts = spikes.rate(times_trials, method='bin', lims =
        [-1, 0.5])

    #takes mean over all trials & timepts
    meanRates = np.mean(np.mean(rates, axis = 2), axis = 0)

    #find all indices where mean rate > 1
    neurons_keep = np.where(meanRates > 1)[0]

    valid_spikes = times_trials[:, neurons_keep]

    allISIs = spikes.isi(valid_spikes)
    ms_ISI = np.multiply(allISIs, 1000)
    concat = utils.concatenate_object_array(ms_ISI, 0)
    if concat.size > 1:
        #every ISI per unit
        flatAll = np.squeeze(utils.concatenate_object_array(ms_ISI,
            0))
    else:
        flatAll = concat[0]

    for idx, neuron in enumerate(neurons_keep):
        #all ISIs less than 1 ms for neuron
        shorts = np.where(flatAll[idx] <= 1)[0]

        num_shorts = np.size(shorts)
        total_isi = np.size(flatAll[idx])
        if num_shorts/total_isi >= 0.1:
            #don't keep the neuron
            neurons_keep = np.delete(neurons_keep, idx)

    return neurons_keep


def neuronsKeep(neurons_keep, times_trials, waves_trials):
    '''
    Filters out non-valid units in given time, wave data
    '''
    times_data = times_trials[:, neurons_keep]
    waves_data = waves_trials[:, neurons_keep]

    return times_data, waves_data

def depth(neurons_keep, unit_info, jitter = True):
    '''
    Saves depth information for two dataests. Depth data is
        discretized -- jitter can be added to depths for ease of
        plotting.

    *Scaled Andre's data to match Alex's data
    '''
    try:
```

```
121         depths = unit_info['laminarDepth'][neurons_keep]
122     except:
123         ### Scale Andre's data
124         depths = (unit_info['betaGammaDepth'][neurons_keep])/1000
125 
126     if jitter:
127         ### Create random vector of jitter
128         x = pd.Series(np.random.uniform(low = -0.02, high = 0.02,
                size = len(depths)))
129 
130         ### Add jitter to original depths. Fill nan values with -3.
131         depths = x.add(depths.reset_index(drop=True), fill_value =
                -3)
132 
133     return depths
134 
135 def rateData(time_data):
136     '''
137     Returns:
138         (a) meanRates: mean spike rate / unit
139         (b) rates: full np array of spike rates (units x timepts x
                trials)
140     '''
141 
142     rates, _ = spikes.rate(time_data, method='bin', lims = [-1, 0.5])
143     meanRates = np.mean(np.mean(rates, axis = 2), axis = 0)
144     meanRates = anscombe(np.expand_dims(meanRates, axis=0))
145 
146     return meanRates, rates
147 
148 def isiData(time_data):
149     '''
150     Returns: ISI data from given spike data. On the scale of
            milliseconds (scaled by 1000).
151     '''
152 
153     allISIs = spikes.isi(time_data)
154     ms_ISI = np.multiply(allISIs, 1000)
155 
156     return ms_ISI
157 
158 def waveAlign(waves_data, spike_waves_schema, trial_subset_indices =
        None):
159     '''
160     Gets the mean-aligned waveform from data.
161 
162         Input: spikeWaves (n_trials, n_units) object array for single
                session
163                spikeWavesSchema:
164                trials_keep: all valid trials
165                neurons_keep: all valid units
166                trial_subset_indices: optional subset of (valid)
                      trials of interest
```

```
167
168         Output: meanAlignWaves (n_trials, n_units): aligned mean
                waveforms for each unit
169                 smpRate: 10x interpolated sampling rate
170
171     '''
172
173     if trial_subset_indices: #if we pass in some subset
174         waves_data = waves_data[trial_subset_indices, :]
175
176     n_trials, n_units = np.shape(waves_data)
177     timepts = spike_waves_schema['elemIndex'][0]
178
179     num_timepts = (np.size(timepts)-1)*10
180     meanAlignWaves = np.zeros((num_timepts, n_units))
181
182     for neuron in range(n_units):
183         spikesAll = utils.concatenate_object_array(waves_data[:,
                neuron], axis = 0, elem_axis = 1)
184         n_timepts, n_spikes = np.shape(spikesAll) #get # of time pts
185         x = np.arange(1, n_timepts+1)
186         xinterp = np.arange(1, n_timepts, 0.1) #keep length, divide
                step by 10
187         waves_interp = utils.interp1(x, spikesAll, xinterp, axis = 0)
188         meanWave = np.mean(waves_interp, axis=1) #get mean waveform
                over all spikes
189         meanTroughIdx = np.argmin(meanWave) #get mean trough idx
190
191         for spike_idx in range(n_spikes):
192             spike = waves_interp[:, spike_idx]
193             spikeTroughIdx = np.argmin(spike)
194             diff = spikeTroughIdx - meanTroughIdx
195             newSpike = np.full(np.shape(spike), np.nan)
196             if diff > 0:
197             #if the spike's trough is shifted ahead of mean trough
198                 newSpike[:-diff] = spike[diff:]
199                 #move it back
200             elif diff < 0:
201                 newSpike[abs(diff):] = spike[:diff]
202             else:
203                 newSpike = spike
204             waves_interp[:, spike_idx] = newSpike
205
206         # new waves_interp with aligned spikes
207         meanAlignWave = np.nanmean(waves_interp, axis=1) #take mean
                of all spikes
208         meanAlignWaves[:, neuron] = meanAlignWave
209
210     smpRate = spike_waves_schema['smpRate']*10
211
212     return meanAlignWaves, smpRate
213
214 def LV(ISIs):
```

```
215     '''
216     Returns:
217     '''
218 
219     n_trials, n_units = np.shape(ISIs)
220     allLV = np.zeros((1, n_units))
221 
222     for neuron in range(n_units):
223         neuronLV = np.zeros((n_trials,))
224         for trial in range(n_trials):
225             if len(ISIs[trial, neuron]) <= 1: #if there are no ISIs
                    to compare against each other
226                 neuronLV[trial] = float('NaN')
227             else:
228                 LV = spikes.isi_stats(ISIs[trial, neuron], stat='LV')
229                 neuronLV[trial] = LV
230         meanLV = np.nanmean(neuronLV, axis=0)
231         allLV[:, neuron] = meanLV
232     return allLV
233 
234 def waveform_check(repolTime):
235     '''
236     Returns: passed_neurons (n_units, ): all units with non-inverted
            waveforms.
237     '''
238 
239     passed_neurons = []
240     row, num_neurons = np.shape(repolTime)
241     for i in range(num_neurons):
242         if not math.isnan(repolTime[:, i]):
243             passed_neurons.append(i)
244 
245     newRepolTime = repolTime[:, passed_neurons]
246     return passed_neurons
247 
248 def spike_i(spikes, predInfo, sampInfo, area, idx):
249     '''
250     Saves the time data for each unit (data is 1 x trials)
251     '''
252     trials, units = np.shape(spikes)
253     for i in range(units):
254         np.save('/home/ehua/clustering/090623_data/spikes/{}_spikes_
                {}.npy'.format(area, i+idx), spikes[:, i])
255 
256         predInfo.to_csv('/home/ehua/clustering/090623_data/info/{}
                _predInfo_{}.csv'.format(area, i+idx))
257         sampInfo.to_csv('/home/ehua/clustering/090623_data/info/{}
                _sampInfo_{}.csv'.format(area, i+idx))
258 
259 def filterSingleElectrodes(electrode_info, depths, lfp, area_idx,
        lfp_probe_idx):
260     '''
261     Returns: indices of non-singular electrodes
```

```
    '''
    idx_keep = np.where(electrode_info['elecType'][area_idx][
        lfp_probe_idx] != 'single')[0]
    depths = depths[idx_keep]
    lfp = lfp[:, idx_keep, :]

    return depths, lfp

def preProcessing(spike_times, trial_info, session_info, spike_waves,
    spike_waves_schema, unit_info, area, unit_count):
    ### trial selection
    trials_keep, predInfo, sampInfo, block_trials, trial_trials,
        unpredSampInfo = trialSelection(trial_info, session_info)
    times_trials, waves_trials = trialsKeep(trials_keep, spike_times,
         spike_waves)

    ### neuron selection
    neurons_keep = neuronSelection(times_trials)
    time_data, waves_data = neuronsKeep(neurons_keep, times_trials,
        waves_trials)

    meanAlignWaves, smpRate = waveAlign(waves_data,
        spike_waves_schema)
    repolTime = spikes.waveform_stats(meanAlignWaves, stat='
        repolarization', smp_rate=smpRate)
    passed_neurons = waveform_check(repolTime)

    time_data = time_data[:, passed_neurons]

    # if time_data.shape[1] >= 2:
    #     spike_i(time_data, predInfo, sampInfo, area, unit_count)

    waves_data = waves_data[:, passed_neurons]
    meanAlignWaves = meanAlignWaves[:, passed_neurons]

    depths = depth(passed_neurons, unit_info)

    meanRates, rates = rateData(time_data)
    meanNeuronRate = np.mean(rates, axis=0)
    blockRates = np.mean(rates[block_trials, :, :], axis = 0)
    trialRates = np.mean(rates[trial_trials, :, :], axis = 0)

    ISIs = isiData(time_data)

    return trials_keep, passed_neurons, meanRates, ISIs,
        meanAlignWaves, smpRate, rates, meanNeuronRate, blockRates,
        trialRates, predInfo, sampInfo, depths, unpredSampInfo,
        trial_trials

def coupling(area_lfp, area_idx, depth_var, electrode_info, unit_info
    , spike_times, area, session, smp_rate):
    probeIDs = electrode_info['probeID'][area_idx].unique()

```

```
    for probeID in probeIDs:
        lfp_probe_idx = np.where(electrode_info['probeID'][area_idx]
           == probeID)[0]
        spk_probe_idx = np.where(unit_info['probeID'][area_idx] ==
           probeID)[0]

        depths = electrode_info[depth_var][area_idx][lfp_probe_idx].
           to_numpy()

        lfp = area_lfp[:, lfp_probe_idx, :]
        spk = spike_times[:, spk_probe_idx]

        if depth_var == 'betaGammaDepth':
            depths, lfp = filterSingleElectrodes(electrode_info,
               depths, lfp, area_idx, lfp_probe_idx)

        ### get idx of depth - superficial is negative, deep is
           positive. labels of 0 (layer 4) ignored
        sup_idx = np.where(depths < 0)[0]
        deep_idx = np.where(depths > 0)[0]

        lfp_sup = np.squeeze(np.mean(lfp[:, sup_idx, :], axis = 1))
        lfp_deep = np.squeeze(np.mean(lfp[:, deep_idx, :], axis = 1))

        spike_trains = spikes.times_to_bool(spk, lims=(-1,2))[0]

        _, n_units, _ = spike_trains.shape

        for unit in range(n_units):
            unit_spikes = np.transpose(np.squeeze(spike_trains[:,
               unit, :]))
            osc_sup,freqs_sup,timepts_sup,n_sup, phi_sup = \
                sync.spike_field_coupling(np.transpose(unit_spikes),
                                          np.transpose(lfp_sup),
                                          time_axis = 1,
                                          smp_rate = smp_rate,
                                          return_phase = True) \

            osc_deep,freqs_deep,timepts_deep,n_deep, phi_deep = \
                sync.spike_field_coupling(np.transpose(unit_spikes),
                                          np.transpose(lfp_deep),
                                          time_axis = 1,
                                          smp_rate = smp_rate,
                                          return_phase = True) \


            np.save('/home/ehua/clustering/090623_data/osc/{}
               _osc_sup_{}_{}_{}'.format(area, session, probeID, unit
               ), np.squeeze(osc_sup))
            np.save('/home/ehua/clustering/090623_data/osc/{}
               _osc_deep_{}_{}_{}'.format(area, session, probeID,
               unit), np.squeeze(osc_deep))

```

```
347             np.save('/home/ehua/clustering/090623_data/osc/{}
                    _phi_sup_{}_{}_{}'.format(area, session, probeID, unit
                    ), phi_sup)
348             np.save('/home/ehua/clustering/090623_data/osc/{}
                    _phi_deep_{}_{}_{}'.format(area, session, probeID,
                    unit), phi_deep)
349
350             sup_path = '/home/ehua/clustering/090623_data/figures/{}
                    _spec_sup_{}_{}_{}'.format(area, session, probeID,
                    unit)
351             deep_path = '/home/ehua/clustering/090623_data/figures/{}
                    _spec_sup_{}_{}_{}'.format(area, session, probeID,
                    unit)
352
353             sup_img, sup_ax = spectra.plot_spectrogram(timepts_sup,
                    freqs_sup, np.squeeze(osc_sup), sup_path, area,
                    session, probeID, unit)
354             deep_imag, deep_ax = spectra.plot_spectrogram(
                    timepts_deep, freqs_deep, np.squeeze(osc_deep),
                    deep_path, area, session, probeID, unit)
355
356         # np.save('/home/ehua/clustering/090623_data/osc/{}
                _phi_sup_session_{}_unit_{}.csv'.format(area, session, i),
                 phi_sup)
357         # np.save('/home/ehua/clustering/090623_data/osc/{}
                _phi_deep_session_{}_unit_{}.csv'.format(area, session, i)
                , phi_deep)
358
359 def featExtract(meanRates, ISIs, meanAlignWaves, smpRate, rates):
360     '''
361     Extracts features of interest from data.
362
363         Input: meanRates (n_units): mean spike rates for each unit
364                ISIs (n_trials, n_units): ISIs for each trial and unit
365                meanAlignWaves (n_timepts, n_units): mean aligned wave
                       for each unit
366                smpRate: 10x interpolated sampling rate
367         Output: featuresDF: dataframe containing features of interest
368                             meanRates, troughToPeak, repolTime, CV,
                                LV
369
370     '''
371
372     troughToPeak = spikes.waveform_stats(meanAlignWaves, stat='width'
            , smp_rate=smpRate) #axis is 0?
373     repolTime = spikes.waveform_stats(meanAlignWaves, stat='
            repolarization', smp_rate=smpRate)
374
375     mean_timepts = np.mean(rates, axis=2)
376     CV = spikes.rate_stats(mean_timepts, stat='CV', axis=0) #deal
            with timepts
377     allLV = LV(ISIs)
378
```

```
    features = {'meanRates': np.squeeze(meanRates).tolist(), '
       troughToPeak': np.squeeze(troughToPeak).tolist(), 'repolTime':
        np.squeeze(repolTime.tolist()), 'CV': np.squeeze(CV).tolist()
       , 'LV': np.squeeze(allLV).tolist()}
    if np.shape(troughToPeak) == (1, 1):
        featuresDF = pd.DataFrame(data=features, index = [0])
    else:
        featuresDF = pd.DataFrame(data=features)

    return featuresDF

def main():
    #validTrials, validNeurons, meanRates, ISIs, meanAlignWaves,
       smpRate, rates = preProcessing(spike_times, trial_info,
       session_info, spike_waves, spike_waves_schema)
    #featuresDF = featExtract(meanRates, ISIs, meanAlignWaves,
       smpRate, rates)
    pass

if __name__ == "__main__":
    main()
```

A.3 Clustering

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Nov  3 16:14:04 2022

@author: huange
"""

import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from statistics import mode
from sklearn.preprocessing import StandardScaler
import pandas as pd
from plotting import *
from analysis import *


def GMM(features, num_reps, area):
    '''
    Performs GMM clustering on data.

        Input: features (n_features, n_datapts): matrix of feature
           values for each datapoint
```

```
                num_reps: number of times clustering is performed for
                    a certain component value

        Output: gmm_min: model fitted using the best number of
            components
                min_labels: cluster assignments for each data point
                average_min_comp = number of components used for
                    clustering

    '''
    components = np.arange(2, 27) # 2-9 clusters
    bics = np.zeros((num_reps, len(components)))
    aics = np.zeros((num_reps, len(components)))
    models = np.empty((0, len(components)))
    labels = []
    min_comps = []

    for rep in range(num_reps): # what is num_reps
        min_bic = np.inf
        rep_models = []
        rep_labels = []
        for comp in components: # for each cluster #
            gmm = GaussianMixture(n_components=comp, random_state=rep
                )
            gmm.fit(features)
            label = gmm.predict(features)

            bic = gmm.bic(features)
            aic = gmm.aic(features)

            bics[rep, comp-2] = bic
            aics[rep, comp-2] = aic
            rep_models.append(gmm)
            rep_labels.append(label)

            if bic < min_bic:
                min_bic = bic
                min_comp = comp
        min_comps.append(min_comp)
        models = np.append(models, np.array(rep_models).reshape((1,
            25)), axis=0)
        labels.append(rep_labels)

    plt.figure(0)
    bics_mean = np.mean(bics, axis=0)
    bics_stds = np.std(bics, axis = 0)
    plt.plot(components, bics_mean, 'k', color='#CC4F1B')
    plt.fill_between(components, bics_mean-bics_stds, bics_mean+
        bics_stds,
                     alpha=0.5, edgecolor='#CC4F1B', facecolor='#
                         FF9848')
    plt.title(area + " bics")
    plt.savefig('/home/ehua/clustering/090623_data/figures/{}_bics.
```

```
        png'.format(area))
70
71      plt.figure(1)
72      aics_mean = np.mean(aics, axis=0)
73      aics_stds = np.std(aics, axis = 0)
74      plt.plot(components, aics_mean, 'k', color='#CC4F1B')
75      plt.fill_between(components, aics_mean-aics_stds, aics_mean+
            aics_stds,
76                       alpha=0.5, edgecolor='#CC4F1B', facecolor='#
                            FF9848')
77      plt.title(area + " aics")
78      plt.savefig('/home/ehua/clustering/090623_data/figures/{}_aics.
            png'.format(area))
79
80      average_min_comp = mode(min_comps)
81      print(min_comps)
82      min_model_idx = np.argmin(bics[:, average_min_comp-2])
83      min_model = models[min_model_idx, average_min_comp-2]
84      min_labels = labels[min_model_idx][average_min_comp-2]
85
86      return min_model, min_labels, average_min_comp\
87
88  def main():
89      area = 'dlPFC'
90      feat_df = pd.read_csv('/home/ehua/clustering/090623_data/{}_df.
            csv'.format(area), index_col = 0)
91      waves_df = pd.read_csv('/home/ehua/clustering/090623_data/{}
            _waves.csv'.format(area), index_col = 0)
92      depths = pd.read_csv('/home/ehua/clustering/090623_data/{}_depths
            .csv'.format(area), index_col = 0)
93      waves = waves_df.to_numpy()
94      waves_ptp = waves.ptp(axis = 0)
95      waves_norm = np.divide(waves, waves_ptp)
96
97      all_params = ['troughToPeak', 'repolTime', 'meanRates', 'CV', 'LV
            ']
98
99      cluster_stats = feat_df[all_params].to_numpy()
100     scaler = StandardScaler()
101     cluster_stats_norm = scaler.fit_transform(cluster_stats)
102
103     _, min_labels, _ = GMM(cluster_stats_norm, 500, area)
104
105     cluster_stats_df = pd.DataFrame(cluster_stats_norm)
106     cluster_stats_df['labels'] = min_labels
107     labels_df = feat_df.copy(deep=True)
108     labels_df['labels'] = min_labels
109     labels_df['depths'] = depths
110
111     labels_df.to_csv('/home/ehua/clustering/090623_data/clusters/{}
            _labels_df.csv'.format(area))
112     min_labels_df = pd.DataFrame(min_labels)
113     min_labels_df.to_csv('/home/ehua/clustering/090623_data/clusters
```

```
        /{}_labels.csv'.format(area))

if __name__ == "__main__":
    main()
```

A.4 Analysis

```
    #!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 21 13:06:39 2023

@author: ehua
"""

from spynal import spikes, info, randstats
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def cluster_count(labels, comp_num):
    counts = np.zeros(comp_num)
    for label in labels['0']:
        counts[label] += 1
    return counts

def pev_func(data, labels):
    pev = info.neural_info(data, labels)
    return pev

def anova(pev, labels):
    p = np.squeeze(randstats.one_way_test(pev, labels))
    #time_vec = np.linspace(-1.5, 2.5, 60)
    #plt.figure()
    #plt.plot(time_vec, p)
    return p

def ttest(data, labels):
    plt.figure()
    comp_num = max(labels)
    fig, axs = plt.subplots(comp_num, 1)
    fig.tight_layout()
    colors = ['xkcd:azure', 'mediumseagreen', 'tab:olive', 'xkcd:
       lavender']
    time_vec = np.linspace(-1, 0.5, 30)

    for i in range(comp_num):
        ax = axs[i]
        cluster_units = labels == i
```

```
        p = randstats.one_sample_test(data[cluster_units, :])
        ax.plot(time_vec, np.squeeze(p), color = colors[i])


def main():
    allPEVDf = pd.read_csv('/home/ehua/clustering/allPEV_samp_V4.csv'
       , index_col = 0)
    allPEV = allPEVDf.to_numpy()

    labels_df = pd.read_csv('/home/ehua/clustering/V4_labels.csv',
       index_col = 0)
    labels = labels_df['labels']

    p = anova(allPEV, labels)
    print(p)

    ttest(allPEV, labels)


if __name__ == "__main__":
    main()
```

A.5 Plotting

```
    #!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Feb  9 15:47:31 2023

@author: ehua
"""
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from matplotlib.ticker import NullFormatter
from scipy import stats
import pandas as pd
from sklearn.manifold import TSNE
from spynal import spikes
import math
from copy import deepcopy
from analysis import cluster_count

def outlier_id(labels_df, comp_num):
    rows, _ = labels_df.shape
    max_std = 2.5

    outlier_col = np.empty((rows, ))
    for i in range(comp_num):
```

```
26          cluster_units = labels_df.loc[labels_df['labels'] == i]
27          cluster_idx = cluster_units.index
28
29          troughToPeak = cluster_units['troughToPeak'].to_numpy()
30          repolTime = cluster_units['repolTime'].to_numpy()
31          ttp_std = np.std(troughToPeak)
32          ttp_mean = np.mean(troughToPeak)
33          ttp_std_filter = np.where(np.logical_or(troughToPeak <
               ttp_mean - (max_std*ttp_std), troughToPeak > ttp_mean + (
               max_std*ttp_std)), 1, 0)
34          rpt_std = np.std(repolTime)
35          rpt_mean = np.mean(repolTime)
36          rpt_std_filter = np.where(np.logical_or(repolTime < rpt_mean
               - (max_std*rpt_std), repolTime > rpt_mean + (max_std*
               rpt_std)), 1, 0)
37
38          outlier_all = np.logical_or(ttp_std_filter, rpt_std_filter)
39
40          outlier_col[cluster_idx] = outlier_all
41
42      labels_df['outliers'] = outlier_col
43      return labels_df
44
45  def pairplot(labels_df, outliers_df, comp_num, area, outlier=False):
46      all_params = ['troughToPeak', 'repolTime', 'meanRates', 'CV', 'LV
           ']
47      if outlier:
48          for i in range(comp_num):
49              cluster_units = outliers_df.loc[outliers_df['labels'] ==
                   i]
50              g = sns.pairplot(cluster_units, hue = "outliers", kind='
                   scatter',
51                                       diag_kind='kde', palette = 'muted
                                           ',  x_vars = all_params,
                                           y_vars = all_params)
52              g.fig.suptitle(area + " outlier pairplot for comp " + i,
                   y = 1.03, fontsize = 20)
53      else:
54          g = sns.pairplot(labels_df, hue = "labels", kind='scatter',
55                                   diag_kind='kde', palette = 'muted',
                                      x_vars = all_params, y_vars =
                                      all_params)
56          g.fig.suptitle(area + " cluster pairplot", y = 1.03, fontsize
                = 20)
57          plt.savefig('/home/ehua/clustering/090623_data/figures/{}
               _pairplot.png'.format(area))
58
59
60  def plot_avg_wave(allAlignWaves, df, comp_num, area):
61      fig, axs = plt.subplots(comp_num, 1)
62      fig.tight_layout()
63      fig.set_figheight(15)
64      fig.set_figwidth(10)
```

```
65
66      clusters =[]
67
68      colors = sns.color_palette("muted")
69      for i in range(comp_num):
70          ax = axs[i]
71          clusters.append(i)
72          cluster_units = df.loc[df['labels'] == i]
73          cluster_units_idx = cluster_units.index
74          outlier_idx = cluster_units.loc[cluster_units["outliers"] ==
                1].index
75          cluster_units_idx = list(set(cluster_units_idx) - set(
                outlier_idx))
76          cluster_waves = allAlignWaves[:, cluster_units_idx]
77          outlier_waves = allAlignWaves[:, outlier_idx]
78
79          ax.plot(cluster_waves, color = colors[i], alpha = 0.2)
80          mean_wave = np.mean(cluster_waves, axis = 1)
81          ax.plot(mean_wave, color = 'k')
82          if outlier_waves.size != 0:
83              ax.plot(outlier_waves, color="crimson", alpha = 0.5)
84      fig.suptitle(area + " cluster waveforms", y = 1.03, fontsize =
            20)
85      fig.legend(clusters)
86
87      plt.savefig('/home/ehua/clustering/090623_data/figures/{}
            _avg_waves.png'.format(area))
88
89
90  def elbow_plot(components, data_mean, data_std):
91      plt.figure(0)
92      plt.plot(components, data_mean, 'k', color='#CC4F1B')
93      plt.fill_between(components, data_mean-data_std, data_mean+
            data_std,
94                       alpha=0.5, edgecolor='#CC4F1B', facecolor='#
                            FF9848')
95
96  def tsne_plot(ax, perplexity, df_tsne, area):
97      ax.set_title(area + " for Perp=%d" % perplexity)
98      sns.scatterplot(data=df_tsne, x='comp1', y='comp2', marker='o',
            hue=df_tsne.label.astype('category').cat.codes, ax = ax)
99      ax.xaxis.set_major_formatter(NullFormatter())
100     ax.yaxis.set_major_formatter(NullFormatter())
101     ax.axis("tight")
102
103 def area_dist(df, comp_num, area):
104     colors = sns.color_palette("muted")
105     plt.figure(figsize = (10, 10))
106     datapts, _ = df.shape
107     labels = []
108     percs = []
109     for i in range(comp_num):
110         cluster_pts = df.loc[df['0'] == i].shape[0]
```

```
        perc = cluster_pts/datapts
        percs.append(perc)
        labels.append(str(i))

    plt.pie(percs, labels=labels, autopct='%1.1f%%',
            shadow=True, startangle=90, colors=colors)
    plt.title(area + " datapoints per cluster")
    plt.savefig('/home/ehua/clustering/090623_data/figures/{}_dist.
       png'.format(area))

def param_values(df, all_params, comp_num, area):
    fig, axs = plt.subplots(1, len(all_params))
    fig.tight_layout()
    fig.set_figheight(10)
    fig.set_figwidth(15)

    colors = sns.color_palette("muted")
    clusters = []

    for i in range(len(all_params)):
        ax = axs[i]
        param = all_params[i]
        param_df = df[[param, 'labels']]
        for j in range(comp_num):
            clusters.append(j)
            cluster_units = param_df.loc[param_df['labels'] == j]
            mean_param = np.mean(cluster_units[param])
            sem_param = stats.sem(cluster_units[param])
            ax.errorbar(x=0, y=mean_param, yerr=sem_param, fmt ='o',
               color = colors[j])
            ax.set_title(param)

    plt.legend(clusters)
    plt.title(area + " parameter values")
    plt.savefig('/home/ehua/clustering/090623_data/figures/{}
       _param_values.png'.format(area))

def psth(df, comp_num, blockRates, trialRates):
    time_vec = np.linspace(-1, 1.95, 60)
    fig, axs = plt.subplots(comp_num, 1)
    fig.tight_layout()
    fig.set_figheight(15)
    fig.set_figwidth(10)

    for i in range(comp_num):
        ax = axs[i]
        cluster_units = df.loc[df['labels'] == i]
        cluster_units_idx = cluster_units.index

        ax.plot(time_vec, np.mean(blockRates[cluster_units_idx, :],
           axis = 0), color="crimson", linewidth = 2)
        ax.plot(time_vec, np.mean(trialRates[cluster_units_idx, :],
           axis = 0), color="cyan", linewidth = 2)
```

```
159
160     fig.legend(["block", "trial"])
161
162 def pev_plot(data, labels, comp_num, area, label_type):
163     plt.figure()
164     #fig, axs = plt.subplots(comp_num, 1)
165     #fig.tight_layout()
166     colors = sns.color_palette("muted")
167     clusters = []
168     time_vec = np.linspace(-1, 0.5, 30)
169
170     for i in range(comp_num):
171         #ax = axs[i]
172         clusters.append(i)
173         cluster_units = labels == i
174
175         data_mean = np.mean(data[cluster_units.to_numpy().flatten(),
                :], axis = 0)
176         plt.plot(time_vec, data_mean, color=colors[i])
177
178         sems = stats.sem(data)
179         plt.fill_between(time_vec, data_mean-sems, data_mean+sems,
180                          alpha=0.5, edgecolor='#CC4F1B', facecolor='#
                             FF9848')
181
182     plt.title(area + ' PEV plot for ' + label_type)
183     plt.savefig('/home/ehua/clustering/090623_data/figures/{}_PEV_{}.
            png'.format(area, label_type))
184     plt.legend(clusters)
185
186 def feat_reduction(df, min_labels, area):
187     '''
188     Reduces N-d data to a 2-d feature space using TSNE method.
189
190         Input: df (n_features, n_datapts): dataframe of features to
                be reduced
191                min_labels: cluster assignments for each datapoint
192                area: cortical area of data
193
194         Output: scatterplot of data in 2-d space.
195
196     '''
197     num_pts, num_vars = df.shape
198     perplexities = [10, 30, 40, 50, 60, 70, 80, 100]
199     (fig, subplots) = plt.subplots(2, 4, figsize=(16, 8))
200     axes = subplots.flatten()
201
202     for i, perplexity in enumerate(perplexities):
203         ax = axes[i]
204
205         tsne = TSNE(
206             n_components=2,
207             init="random",
```

```
208             perplexity=perplexity,
209             learning_rate="auto",
210             n_iter = 5000
211         )
212         df_embedded = tsne.fit_transform(df)
213 
214         df_tsne = pd.DataFrame(df_embedded, columns=['comp1', 'comp2'
                ])
215         df_tsne["label"] = min_labels
216 
217         tsne_plot(ax, perplexity, df_tsne, area)
218 
219 
220 def raster(area, labels, cluster, cond_data, cond_type, unit_num,
        cluster_units):
221     '''
222     Generates raster plots of spike data per cluster for different
            PEV labels.
223 
224         Input: spike_data (n_units, n_timepts): ndarray of spike
                timestamps
225             labels (n_units, ): column of cluster labels
226             comp_num: number of clusters
227             cond_data (n_units, ): labels of conditions
228             cond_type: condition type (samp or pred)
229 
230         Output: scatterplot of data in 2-d space.
231 
232     '''
233     fig, axs = plt.subplots(math.ceil(unit_num/10), 10)
234     fig.suptitle('Raster Plot of Spikes for ' + cond_type + ' in ' +
            area)
235     cond_name = str(cond_type + 'Info')
236     ax_count = 0
237 
238     for i in cluster_units.index:
239 
240         spikes_i = np.load('/home/ehua/clustering/090623_data/spikes
                /{}_spikes_{}.npy'.format(area, i), allow_pickle=True)
241         info_i = pd.read_csv('/home/ehua/clustering/090623_data/info
                /{}_{}_{}.csv'.format(area, cond_name, i), index_col = 0)
242 
243         r, c = divmod(ax_count, 10)
244         ax = axs[r, c]
245 
246         spikes_df = pd.DataFrame(spikes_i)
247         info_i.index = spikes_df.index
248         df = pd.concat([spikes_df, info_i], axis = 1)
249 
250         if cond_type == 'samp':
251             df = df.sort_values('sample')
252             df = df.reset_index(drop = True)
253 
```

```
254             spikes.plot_raster(df[0], ax = ax)
255
256             last1Trial = df['sample'].where(df['sample']==1.0).
                    last_valid_index()
257             last2Trial = df['sample'].where(df['sample']==2.0).
                    last_valid_index()
258             last3Trial = df['sample'].where(df['sample']==3.0).
                    last_valid_index()
259             ax.axhspan(0, last1Trial, facecolor='b', alpha=0.3)
260             ax.axhspan(last1Trial, last2Trial, facecolor='m', alpha
                    =0.3)
261             ax.axhspan(last2Trial, last3Trial, facecolor='y', alpha
                    =0.3)
262             #plt.axhline(y = last1Trial, color = 'r', linestyle =
                    '-')
263             #plt.axhline(y = last2Trial, color = 'r', linestyle =
                    '-')
264
265         else:
266             df = df.sort_values('blockType')
267             df = df.reset_index(drop = True)
268
269             spikes.plot_raster(df[0], ax=ax)
270
271             last1Trial = df['blockType'].where(df['blockType']==1.0).
                    last_valid_index()
272             last2Trial = df['blockType'].where(df['blockType']==2.0).
                    last_valid_index()
273             plt.axhspan(0, last1Trial, facecolor='b', alpha=0.3)
274             plt.axhspan(last1Trial, last2Trial, facecolor='m', alpha
                    =0.3)
275
276         ax_count += 1
277
278 def depth_analysis(depths, labels, counts, area):
279     depths = depths.rename(columns={"0": "depth"})
280     labels = labels.rename(columns={"0": "label"})
281     df = pd.concat([depths, labels], axis=1)
282
283     fig, ax = plt.subplots(figsize=(19,19))
284     df.label = df.label.astype("category")
285     sns.swarmplot(data=df, x="label", y="depth", palette='muted', ax=
            ax)
286
287     # weights = np.ones(len(labels))
288     # for i in range(len(labels)):
289     #     weights[i] = 1/counts[labels['label'][i]]
290
291     # sns.histplot(data=df, x="depth", hue="label", multiple="stack",
            weights = weights, palette='muted')
292
293     plt.title(area + 'Depth Distribution per Cluster')
294     plt.savefig('/home/ehua/clustering/090623_data/figures/{}
```

```
        _cluster_depth_nojitter.png'.format(area))

def main():
    areas = ['7A', 'V4', 'LIP', 'PFC']
    pev_types = ['samp', 'pred']


    for area in ['LIP']:
        #feat_df = pd.read_csv('/home/ehua/clustering/090623_data/{}
           _df.csv'.format(area), index_col = 0)
        waves_df = pd.read_csv('/home/ehua/clustering/090623_data/{}
           _waves.csv'.format(area), index_col = 0)
        waves = waves_df.to_numpy()
        waves_ptp = waves.ptp(axis = 0)
        waves_norm = np.divide(waves, waves_ptp)

        #spikes_df = pd.read_csv('/home/ehua/clustering/090623_data
           /{}_spikes.csv'.format(area), index_col = 0)
        #spikes = spikes_df.to_numpy()

        # jitter_df = pd.read_csv('/home/ehua/clustering/090623_data
           /{}_depths_jitter.csv'.format(area), index_col = 0)
        # jitter = jitter_df.to_numpy()
        # depths_df = pd.read_csv('/home/ehua/clustering/090623_data
           /{}_depths.csv'.format(area), index_col = 0)
        # depths = depths_df.to_numpy()


        all_params = ['troughToPeak', 'repolTime', 'meanRates', 'CV',
            'LV']
        labels_df = pd.read_csv('/home/ehua/clustering/090623_data/
           clusters/{}_labels_df.csv'.format(area), index_col = 0)
        labels = pd.read_csv('/home/ehua/clustering/090623_data/
           clusters/{}_labels.csv'.format(area), index_col = 0)
        comp_num = max(labels['0']+1)
        counts = cluster_count(labels, comp_num)

        #depth_analysis(depths_df, labels, counts, area)

        #feat_reduction(feat_df, labels, area)

        # outliers_df = outlier_id(labels_df, comp_num)
        # plot_avg_wave(waves_norm, outliers_df, comp_num, area)
        # pairplot(labels_df, outliers_df, comp_num, area)

        # area_dist(labels, comp_num, area)
        # param_values(labels_df, all_params, comp_num, area)

        #psth(labels_df, comp_num, allBlockRates, allTrialRates)

        for pev_type in pev_types:
            pev_df = pd.read_csv('/home/ehua/clustering/090623_data
               /{}_PEV_{}.csv'.format(area, pev_type), index_col = 0)
```

```
            pev_data = pev_df.to_numpy()

            pev_plot(pev_data, labels, comp_num, area, pev_type)

        for cond_type in pev_types:
            cluster = 4
            cond_data = pd.read_csv('/home/ehua/clustering/090623
               _data/{}_PEV_{}.csv'.format(area, cond_type),
               index_col = 0)

            cluster_units = labels.loc[labels['0'] == cluster]
            unit_num = cluster_units.size
            raster(area, labels, cluster, cond_data, cond_type,
               unit_num, cluster_units)

if __name__ == "__main__":
    main()
```

A.6 Utilities

```
    #!/usr/bin/env python3
    # -*- coding: utf-8 -*-
    """
    Created on Tue Feb 14 15:22:24 2023

    @author: ehua
    """
    import numpy as np

    def anscombe(x):
        return 2.0*np.sqrt(x + 3.0/8.0)
```