Neuronal Analysis Assesment

Ocansey Kevin  
Department of Computing and Informatics  
Bournemouth University Bournemouth, United Kingdom  
s5631590@bournemouth.ac.uk

Adwoa Yeboah Danquahss  
Department of Computing and Informatics  
Bournemouth University Bournemouth, United Kingdom  
s5612692@bournemouth.ac.uk

Abstract

Our study focuses on making statistical inferences from two datasets (‘DatasetCoursework\_OCA’ &  ‘DatasetCoursework\_YEB’) obtained through experiments with various trials. Both datasets consist of 8 columns. Columns 1 to 6 represent neuronal data, where 1s represents a spike and 0s represents no spike. Column 7 contains ground truth values from 1 to 4, which represents mice behaviour and column 8 records the trial information. Initially we analyzed the firing rates using a window size of 0.2, aiming to establish a correlation between the firing rates and the spike train data. After, we aimed to model artificial neural data to mimic our datasets using Leaky Integrate and Fire.

Additionally, we utilized machine learning models from scikit-learn to train on our data, aiming to predict and further understand the patterns within the neuronal firing rates. Our methodology involves using Leaky Integrate and Fire Models to mimic the firing rate in the actual dataset. We introduce noise in this model as a way to also mimic biological activity.

# Introduction

Neurons, often referred to as the brain’s nerve cell, are the basic building blocks of conveying and processing information in the form of electric impulses known as spikes or action potentials. Firing rate refers to how many times in one millisecond a neuron would spike and is used in combination with other metrics to determine neural activities and dynamics in neurons. This forms a basis to identify correlation between activity and neuron behavior. These spiking activities are termed as action potentials and are measured per unit time for the specific neuron/neuron pool.

Our aim is to, in parallel with firing rate and spike train data, generate classifiers that can accurately predict the actions of the mice. The use of LIF model allows reconstructing the artificial spike trains in a similar way to which is present in datasets. The first step is to figure out the proper window size. This size must capture the correct variability in the behavior of the spikes while soothing and aggregating spike trains in a specific timeframe as one. When the p-value is 0.05 or less for correlation between the firing rates and the spike trains, then it is statistically significant. Our window size of 0.2 seconds is recommended since every row in both Dataset is 10ms our window size is 200ms.

# Methodology

## Generating artificial spikes

We generated artificial spike trains using a Leaky Integrate and Fire model. We relied on the presynaptic current (psc) to generate spike. For Neuron 1 of Dataset Coursework\_YEB, we analyzed a subset of data from trials 1 to 3, referred to as ‘df\_trial\_1\_3’. For Neuron 1 of DatasetCoursework\_OCA we used trial 10 to 12 referred to as ‘df\_trials\_10\_11\_12\_kev’ to experiment with the timing of the presynaptic currents(psc) relative to the spike time. Initially a psc for each spike was introduced 0.004 before its spike time (e.g., if the spike time was at 0.134 seconds, the psc began at 0.130 seconds). This approach successfully generated a spike. Next, we increased the timing offset to 0.010 seconds before the spike (e.g., psc starting at 0.124 seconds for a 0.134-second spike). This adjustment led to the generation of two bursts instead of a single spike. More information about this experiment can be found in Discussion. The artificial spikes we generated with LIF model can be seen below.

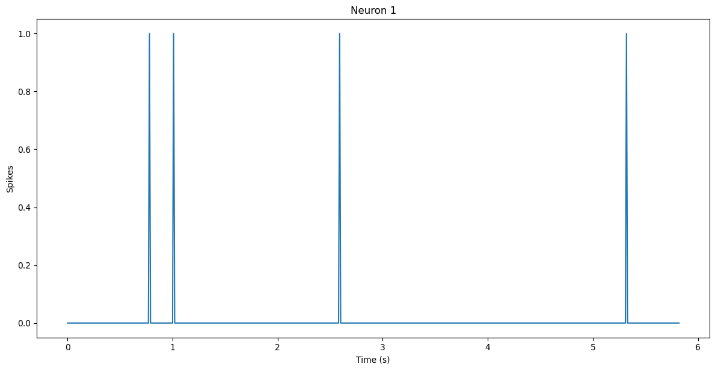


Fig 1.1 *Spike Trains for Neuron 1 of trials 1,2 and 3. From DatasetCourseWork\_YEB*

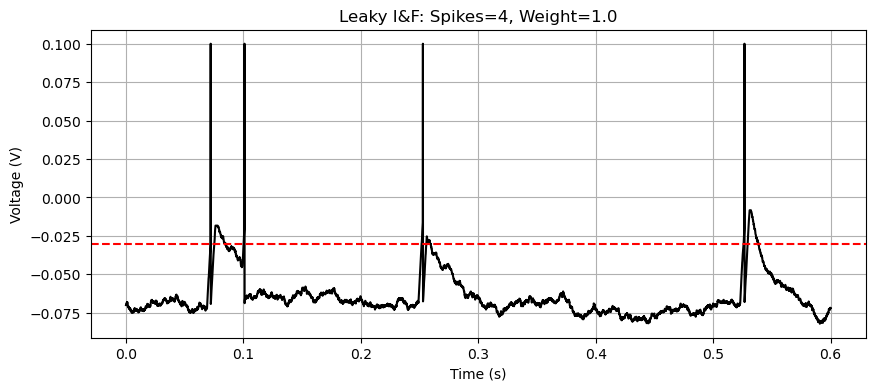
**

Fig 1.2 *Artificial Spike Trains generated with LIF Model* *DatasetCourseWork\_YEB for Neuron 1*

**1. psc1 = np.arange(0.069, 0.078,0.0001) #0.009**

**2. psc2 = np.arange(0.099, 0.101,0.0001) #0.002**

**3. psc3 = np.arange(0.249, 0.256, 0.0001) #0.007**

**4. psc4 = np.arange(0.523, 0.532, 0.0001) #0.009**

**5. psc = np.concatenate([psc1, psc2, psc3, psc4])**

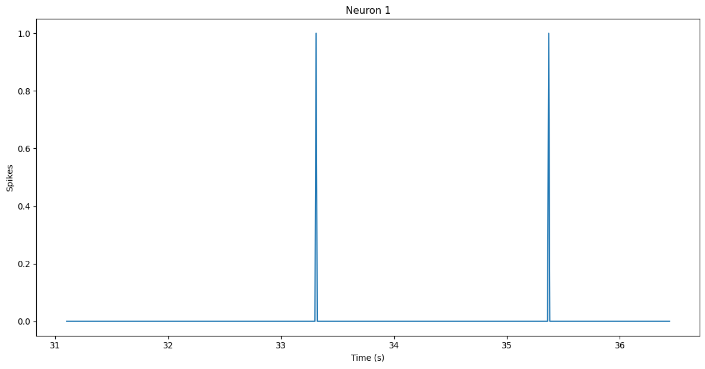


Fig 1.3 *Spike Trains of trials 10, 12 and 13. From DatasetCourseWork\_OCA for Neuron 1*

* Fig 1.1 Artficial Spike Trains of trials 10 12 and 13 using LIF model. DatasetCourseWork_OCA
*

Fig 1.4 *Artificial* *Spike Trains of Neuron 1 for trials 10,12 and 13 using LIF model. DatasetCourseWork\_OCA*

## Discussion

The diagrams for the artificial spikes, generated under varying hard coded psc conditions. This illustrates an important observation: the dataset does not explicitly capture the precise characteristics of the psc required to trigger spikes. For instance, the difference between a psc offset of 0.004 seconds and 0.010 seconds resulted in distinct spiking behaviors (Fig 2), indicating that the membrane dynamics such as weights, membrane resistance, adaptive threshold decay, and absolute refractory period of the neuron play a significant role. This necessitated a trial-and-error approach to reverse-engineer the unique properties of each neuron’s membrane potential and spiking thresholds. For artificial spikes using a LIF model a group of neurons can have similar membrane characteristics but different psc including spikes from other neurons nearby.

Artificial spike trains for Neuron 1 across trials 1 to 3 were generated by applying four presynaptic currents (psc1 to psc4) at specific times, each with varying offsets from the actual spike times.

These psc produced a similar spike train to the main spike data in Fig 1.1 as seen Fig 1.2.

On the other hand the figure below highlights how offsets beyond 0.010 seconds for psc1, led to consecutive bursts of spikes.

By offsetting the psc by 0.010 seconds instead of 0.004 from the actual spike times we see a burst of spike trains in Fig 2.1 below.

A graph of a graph showing a number of data

Description automatically generated with medium confidence

Fig 2.1 *Effect of Exceeding the Threshold Offset creates two bursts. Dataset\_YEB*

With that in mind, we moved on in exploring the network dynamics by selecting two neurons that `appeared` to influence each other, regardless of the spike trains of other neurons. The goal was to create artificial spike trains of one neuron by feeding it, spike times (post synaptic input) from another neuron as presynaptic input. This step aimed to understand how a specific pair of neurons could affect the overall network behavior.

For DatasetCoursework\_YEB, our priority was neurons with low firing rates to reduce complexity and better isolate interactions. For DatasetCoursework\_OCA, we chose one neuron with a high firing rate and another with a low firing rate to explore contrasting dynamics. For this analysis we will analyze the parallel trials 1 to 3 of these neurons for both Datasets to evaluate their interdependence.

For Dataset Coursework\_YEB, we are going to generate artificial spikes for neuron 3 and pass those spikes as psc to neuron 1. We start generating artificial spikes by looking at the original spike times of neuron 3.

Input

1. spike\_times\_n3\_YEB = df\_trials\_1\_2\_3\_YEB['Neuron\_3'] == 1

2. spike\_times\_n3\_YEB = spike\_times\_n3\_YEB[spike\_times\_n3\_YEB == True].index

3. spike\_times\_n3\_YEB = spike\_times\_n3\_YEB \* 0.001

4. spike\_times\_n3\_YEB

Output: Index([0.067, 0.325], dtype='float64')

The above code shows the original spike times. To generate artificial spikes, we passed in the psc with an offset ranging from 0.004 to 0.005 for each particular spike.

1. psc1 = np.arange(0.062, 0.067,0.0001) #0.005

2. psc2 = np.arange(0.319, 0.325,0.0001) #0.005

3. psc = np.concatenate([psc1, psc2])

4. input\_currents = {'psc': psc\_n3, 'I\_0': 1e-8, 'U\_0': 0.3}

5. time, U\_plot, artificial\_spike\_times\_n3\_YEB = LeakyIF\_6(input\_currents=input\_currents, duration = 0.6, dt = 0.0001)

The code above generates the artificial spike times for Neuron 3 as seen in Fig 1.4

We passed the generated spikes as psc\_5 along with the other pscs. After concatenating all pscs into the neuron 1 model, ensure that the final psc is sorted.

The code below generates the artificial spike times for Neuron 1 as seen in Fig 1.2

1. psc1 = np.arange(0.07509, 0.078,0.0001) #0.00291

2. psc2 = np.arange(0.098, 0.101,0.0001) #0.003

3. psc3 = np.arange(0.25519, 0.259, 0.0001) #0.00381

4. psc4 = np.arange(0.52821, 0.532, 0.0001) #0.0041

5. psc\_5 = generate\_dynamic\_psc(artificial\_spike\_times\_n3\_YEB) #post synaptic currents from Neuron 3

6. psc\_n1 = np.concatenate([psc1, psc2, psc3, psc4, psc\_5])

7. psc\_n1.sort()

8.

9. input\_currents = {'psc': psc\_n1, 'I\_0': 1e-8, 'U\_0': 0.3}

10. time, U\_plot, artificial\_spike\_times\_n1\_YEB = LeakyIF\_6(input\_currents=input\_currents, duration = 0.6, dt = 0.0001)

The results are seen below

Artificial spikes for Neuron 1. LIF Model 


*Fig 3.1 Artificial Neuron Network Showing the effect of artificial spikes of Neuron 3 on Neuron 1.*

As we can see from above, the spikes from Neuron 3 caused huge increase in action potentials at 0.06 and 2.

For Dataset Coursework\_OCA, we are going to pass neuron 3 spikes as psc to neuron 5.

The code for generating artificial spikes for neuron 3

1. psc1 = np.arange(0.051, 0.057, 0.0001) #0.005

2. # Burst: 0.177 and 0.178

3. psc2 = np.arange(0.1659, 0.178, 0.0001) #0.009

4. psc3 = np.arange(0.24619, 0.251, 0.0001) #0.004

5. psc4 = np.arange(0.287, 0.294, 0.0001) #0.004

6. psc5 = np.arange(0.60, 0.61, 0.0001) #0.004

7. psc6 = np.arange(0.65, 0.66, 0.0001) #0.004

8. psc7 = np.arange(0.936, 0.941, 0.0001) #0.004

9. # Burst: 0.177 and 0.178

10. psc8 = np.arange(1.307, 1.316, 0.0001) #0.016

11. psc\_n3 = np.concatenate([psc1, psc2, psc3, psc4, psc5, psc6, psc7, psc8])

12. input\_currents = {'psc': psc\_n3, 'I\_0': 1e-8, 'U\_0': 0.3}

13. time, U\_plot, artificial\_spike\_times\_n3\_OCA = LeakyIF\_6(input\_currents=input\_currents, duration = 1.5, dt = 0.0001)

The results for the above code can be seen below.

A graph showing a number of times

Description automatically generated with medium confidence

*Fig. 3.2 Artificial spike train for Neuron 3*

1. # Generate psc from neuron 3

2. psc1 = np.arange(0.028, 0.034, 0.0001) # Spike at 0.034

3. psc2 = np.arange(0.059, 0.065, 0.0001) # Spike at 0.065

4. # Burst: 0.115 and 0.121

5. psc3 = np.arange(0.109, 0.121, 0.0001) # Spikes at 0.115, 0.121

6. psc4 = np.arange(0.132, 0.138, 0.0001) # Spike at 0.138

7. psc5 = np.arange(0.146, 0.152, 0.0001) # Spike at 0.152

8. # Burst: 0.24 and 0.242

9. psc6 = np.arange(0.232, 0.242, 0.0001) # Spikes at 0.24, 0.242

10. psc7 = np.arange(0.340, 0.346, 0.0001) # Spike at 0.346

11. # Burst: 0.363, 0.366, 0.37, 0.372

12. psc8 = np.arange(0.357, 0.372, 0.0001) # Spikes at 0.363, 0.366, 0.37, 0.372

13. psc9 = np.arange(0.468, 0.474, 0.0001) # Spike at 0.474

14. psc10 = np.arange(0.532, 0.538, 0.0001) # Spike at 0.538

15. psc11 = np.arange(0.551, 0.557, 0.0001) # Spike at 0.557

16. psc12 = np.arange(0.592, 0.598, 0.0001) # Spike at 0.598

17. psc13 = np.arange(0.664, 0.67, 0.0001) # Spike at 0.67

18. psc14 = np.arange(0.710, 0.716, 0.0001) # Spike at 0.716

19. psc15 = np.arange(0.801, 0.807, 0.0001) # Spike at 0.807

20. psc16 = np.arange(0.916, 0.922, 0.0001) # Spike at 0.922

21. # Burst: 0.967 and 0.974

22. psc17 = np.arange(0.961, 0.974, 0.0001) # Spikes at 0.967, 0.974

23. psc18 = np.arange(1.156, 1.162, 0.0001) # Spike at 1.162

24. # Burst: 1.349, 1.352, 1.359

25. psc19 = np.arange(1.343, 1.359, 0.0001) # Spikes at 1.349, 1.352, 1.359

26. psc20 = np.arange(1.405, 1.413, 0.0001) # Spike at 1.413

27. psc21 = generate\_dynamic\_psc(artificial\_spike\_times\_n3\_OCA) #post synaptic currents from Neuron 3

28. psc\_n5 = np.concatenate([psc1, psc2, psc3, psc4, psc5, psc6, psc7, psc8, psc9, psc10, psc11, psc12, psc13, psc14, psc15, psc16, psc17, psc18, psc19, psc20, psc21])

29. psc\_n5.sort()

30. input\_currents = {'psc': psc\_n5, 'I\_0': 1e-8, 'U\_0': 0.3}

31. time, U\_plot, artificial\_spike\_times\_n5\_OCA = LeakyIF\_6(input\_currents=input\_currents, duration = 1.6, dt = 0.0001)

The results for this code is seen below

A graph showing a number of times

Description automatically generated with medium confidence

*Fig. 3.3 Artificial Neuron Network Showing the effect of artificial spikes of Neuron 3 on Neuron 5.*

## Discussion Summary

First we compare Neuron 3 original spike trains with the artificial spike trains

A graph with text and numbers

Description automatically generated

*Fig 4.1 Comparisons of Original Spike vs Artificial Spikes neuron 3\_YEB*

*A graph with text and numbers

Description automatically generated*

*Fig 4.2 Comparisons of Original Spike vs Artificial Spikes neuron 1\_YEB*

*A graph with text and numbers

Description automatically generated with medium confidence*

*Fig 4.3 Comparisons of Original Spike vs Artificial Spikes neuron 3\_OCA*

*A graph with numbers and lines

Description automatically generated with medium confidence*

*Fig 4.4 Comparisons of Original Spike vs Artificial Spikes neuron 5\_OCA*

**Replication of Original Spike Trains**

The most effective method to replicate the original spike trains involved standardizing all neurons and applying offset postsynaptic currents (PSCs) for each neuron. Additionally, certain spikes were transmitted as PSCs to other neurons. Specifically, for Neuron 5\_OCA and Neuron\_1\_YEB, the network modeling resulted in minimal disparity compared to the original spike trains.

# *B. Estimating Firing Rates*

## Walkthrough

After generating our artificial spike times for both datasets

* artificial\_spike\_times\_n5\_OCA
* artificial\_spike\_times\_n3\_OCA
* artificial\_spike\_times\_n3\_YEB
* artificial\_spike\_times\_n1\_YEB

The spike trains were transformed into binary series, and a smoothing rectangular time window was applied to compute the firing rate. This approach was implemented using a custom Python function, which iteratively calculated the firing rate for each window

We calculated the rates for our artificial spike times using the following code.

1. def rates\_estimation\_for\_neural\_network(spikes, neurons=None, window\_size=0.2, bin\_size=0.01, duration = 0.583):

2.     """

3.     Transform spike trains (binary series) to single-trial time-dependent firing rates

4.     by applying a smoothing rectangular time window. Optionally, process and plot only specified neurons.

5.

6.     Parameters:

7.     spikes: numpy array

8.         Matrix where each column (except the last two) is a neuron and each row is an observation.

9.         The second-to-last column contains ground truth, and the last column contains the trial number.

10.     neurons: list of int, optional

11.         List of neuron indices to process and plot. If None, all neurons are processed.

12.     window\_size: float, optional

13.         Size of the rectangular smoothing window in seconds (default is 0.2).

14.     bin\_size: float, optional

15.         Temporal resolution of each row in the spike matrix in seconds (default is 0.001).

16.

17.     Returns:

18.     rate: numpy array

19.         Matrix with time-dependent firing rates for the specified neurons in spikes/second.

20.         The last two columns contain ground truth and trial numbers, respectively.

21.     """

22.

23.

24.     if duration > 1.5:

25.         ground\_truth = spikes[:, -2]

26.         trial\_num = spikes[:, -1]

27.     if duration > 0.6:

28.         ground\_truth = df\_trials\_1\_2\_3\_OCA.values[:, -2]  # Extract ground truth

29.         trial\_num = df\_trials\_1\_2\_3\_OCA.values[:, -1]     # Extract trial numbers

30.     else:

31.         ground\_truth = df\_trials\_1\_2\_3\_YEB.values[:, -2]

32.         trial\_num = df\_trials\_1\_2\_3\_YEB.values[:, -1]

33.

34.     # ground\_truth = spikes[:, -2]  # Extract ground truth

35.     # trial\_num = spikes[:, -1]     # Extract trial numbers

36.

37.     # Determine which neurons to process

38.     if neurons is None:

39.         neuron\_indices = list(range(spikes.shape[1] - 2))  # All neuron columns

40.     else:

41.         # Validate neuron indices

42.         max\_neuron = spikes.shape[1] - 3

43.         if any(n < 0 or n > max\_neuron for n in neurons):

44.             raise ValueError(f"Neuron indices should be between 0 and {max\_neuron}")

45.         neuron\_indices = neurons

46.

47.     spikes\_data = spikes[:, neuron\_indices]  # Select specified neurons

48.     n, m = spikes\_data.shape                # Number of rows (time points) and selected neurons

49.     times = np.arange(0, n \* bin\_size, bin\_size)  # Time vector

50.     window\_steps = int(np.round(window\_size / bin\_size))  # Steps in one window

51.

52.     rate = np.zeros((n, m))  # Initialize the rate matrix

53.

54.     for j in range(m):  # Loop over selected neurons

55.         smoothed = []

56.         for i in range(n - window\_steps + 1):

57.             spikes\_one\_neuron = spikes\_data[i:i + window\_steps, j]

58.             rate\_in\_window = np.sum(spikes\_one\_neuron) / window\_size

59.             smoothed.append(rate\_in\_window)

60.

61.         # Complete the series by repeating the last value

62.         smoothed = np.concatenate([smoothed, [smoothed[-1]] \* (window\_steps - 1)])

63.         rate[:, j] = smoothed

64.

65.     # Add ground truth and trial numbers back

66.     rate = np.column\_stack((rate, ground\_truth, trial\_num))

67.

68.     # Plotting

69.     num\_plots = m \* 2  # Each neuron has two plots: spikes and rate

70.     plt.figure(figsize=(12, 4 \* m))  # Adjust figure size based on number of neurons

71.

72.     for j in range(m):

73.         # Plot spike trains

74.         plt.subplot(m, 2, 2 \* j + 1)

75.         plt.plot(times, spikes\_data[:, j], drawstyle='steps-pre')

76.         plt.title(f"Neuron {neuron\_indices[j] + 1} Spike Train")

77.         plt.ylabel("Spikes")

78.         plt.xlabel("Time (s)")

79.

80.         # Plot firing rates

81.         plt.subplot(m, 2, 2 \* j + 2)

82.         plt.plot(times, rate[:, j], 'r')

83.         plt.title(f"Neuron {neuron\_indices[j] + 1} Firing Rate")

84.         plt.ylabel("Spikes/sec")

85.         plt.xlabel("Time (s)")

86.

We generated firing rates for all our neurons in our original datasets as well as our two artificial neurons for both datasets. The results can be seen below.

A group of graphs showing different sizes of lines

Description automatically generated with medium confidence

*Fig 5.1 Firing rates for artificial neurons 3 and neuron 5 from Dataset\_OCA relabelled as neuron1 and 2 respectively.*

A graph of a number of lines

Description automatically generated with medium confidence

*Fig 5.2 Firing rates for artificial neurons 1 and neuron 3 from Dataset\_YEB relabelled as neuron 1 and neuron 2 respectively.*

A graph of a graph

Description automatically generated with medium confidence

*Fig 5.3 spike trains and Firing rates df\_trials\_1\_2\_3 from Dataset\_OCA*

A screenshot of a graph

Description automatically generated

*Fig 5.3 spike trains and Firing rates df\_trials\_1\_2\_3 from Dataset\_YEB*

# *Classification and Decoding*

The final stage that was conducted in our analysis was the classification and decoding of neuronal activity in order to anticipate behavioral patterns. We trained and evaluated multiple classifying models employing a range of machine learning algorithms using the original and artificial spike trains firing rates. The goal was to find a pattern in the neuronal firing rates that could be associated with the given behavioral.

*Building the Machine Learning Models*

We utilized three classifiers for this job that are widely accepted.

Logistic Regression (LR): Linear model estimating the probability of given behavior conditioning on the obtained firing rates.

Support Vector Machine (SVM): Kernel approach designed with the purpose of locating and determining behavioral classes based on an optimal hyperplane.

Linear Discriminant Analysis (LDA): Statistical technique which embeds the data forcefully onto a space of lower dimensions in order to achieve better class separability.

*Training and Evaluation*

The classifiers trained in this study were given firing rate data with labels representing the ground truth. The performance of the models was measured through the metrics: accuracy, precision, recall and F1-score. During the experiments, cross-validation was utilized to validate the results and avoid overfitting.

For the models, the classification for the original dataset df\_trials1\_2\_3 had these results for accuracy:

* Logistic Regression: 40.17%
* SVM: 38.46%
* LDA: 40.17%

Logistic Regression Accuracy: 0.4017094017094017

precision recall f1-score support

1.0 0.59 0.27 0.37 48

2.0 0.67 0.06 0.11 33

3.0 0.35 0.89 0.50 36

accuracy 0.40 117

SVM Accuracy: 0.38461538461538464

precision recall f1-score support

1.0 0.55 0.23 0.32 48

2.0 0.40 0.06 0.11 33

3.0 0.35 0.89 0.50 36

LDA Accuracy: 0.4017094017094017

precision recall f1-score support

1.0 0.59 0.27 0.37 48

2.0 0.67 0.06 0.11 33

3.0 0.35 0.89 0.50 36

accuracy 0.40 117

macro avg 0.54 0.41 0.33 117

weighted avg 0.54 0.40 0.34 117

On the other hand our artificial spike trains produced the following results.

For Dataset\_OCA the two neurons 3, and 5 produce the following results.

SVM Accuracy: 0.728448275862069

precision recall f1-score support

1.0 0.00 0.00 0.00 184

3.0 0.73 1.00 0.84 676

4.0 0.00 0.00 0.00 68

accuracy 0.73 928

macro avg 0.24 0.33 0.28 928

weighted avg 0.53 0.73 0.61 928

LDA Accuracy: 0.7262931034482759

precision recall f1-score support

1.0 0.00 0.00 0.00 184

3.0 0.73 1.00 0.84 676

4.0 0.25 0.01 0.03 68

accuracy 0.73 928

macro avg 0.33 0.34 0.29 928

weighted avg 0.55 0.73 0.61 928

For Dataset\_OCA the two neurons 3, and 1 produce the following results.

SVM Accuracy: 0.358974358974359

precision recall f1-score support

1.0 1.00 0.06 0.12 48

2.0 1.00 0.09 0.17 33

3.0 0.32 1.00 0.49 36

accuracy 0.36 117

macro avg 0.77 0.38 0.26 117

weighted avg 0.79 0.36 0.25 117

LDA Accuracy: 0.37606837606837606

precision recall f1-score support

1.0 0.50 0.19 0.27 48

2.0 1.00 0.09 0.17 33

3.0 0.33 0.89 0.48 36

accuracy 0.38 117

macro avg 0.61 0.39 0.31 117

weighted avg 0.59 0.38 0.31 117

# Concluding remarks

The current investigation delved into the ways neurons function during specific tasks, building on the interactions between neuron activity and behavior by employing machine learning techniques. In addition to employing a variety of methodologies, we were able to reproduce neuronal recorded datasets with the use of the LIF models through the generation of artificial spike trains through rate estimating and behavioral decoding. Our findings from the previous methods provided us with a more complete understanding of the reasoning neuron activity and the behavior being analyzed were linked.

*Key findings include*:

* Any type of noise was included into the LIF model, such as biologically inspired noise or inter-neuronal interactions, the efficiency of the model increased greatly; moreover, it was possible to reproduce the patterns of the spike train.
* By windowing and controlling the size to around 200ms, trace signals were taken without losing too much spatial information.
* Behavior reconstruction from neuronal signals was shown by the machine learning models to be possible; however there was a low accuracy of classification indicating that modelling complex biological systems remains a challenge.
* The findings highlighted the advantages of using computational and statistical tools, while also showing the drawbacks: linear classifiers performed poorly and capturing nonlinear dynamics still proves difficult. This shows the prospective work which - combines the power of artificial neural networks and other models - still remains ahead.

##### References

1. Balaguer- Ballester et al., 2020 Pios Computational Biology
2. Balaguer- Ballester Emili, lecture notes, 2024-2025
3. Ballester, E. and Rupp, A. (2019). Modelling and MEG evidence of early consonance processing in auditory cortex PLoS Computational Biology, 15 (2)Durstewitz, D. (2017). Advanced Data Analysis in Neuroscience: Integrating Statistical and Computational Models (Bernstein Series in Computational Neuroscience). Springer, 320 p. ISBN-13: 9783319599748
4. Gerstner,W.,Kistler,W.M.,Naud,N.and2014 <https://neuronaldynamics.epfl.ch/>
5. Hodkin-Huxley model, theory of action potential, 1952
6. Muller-Komorowska et al., 2023 Nature Communications
7. Murphy, K., (2023) Probabilistic Machine Learning: Advanced Topics Massachusetts Institute of Technology Press, 1360 p. ISBN: 0262048434, 978026204843
8. Nogueira et al., 2017 Notare Communications
9. P., Dayan and, L. F. Abbot.,2005 Theoretical Neuroscience: Computational And Mathematical Modelling of Neural Systems. Massachusetts Institute of Technology Press, 460 p. ISBN: 0262541858, 9780262541855. Classic text, more focused on machine learning.
10. Rasmussen and Williams (2006).Gaussian Processes for Machine Learning. MIT press ISBN 0-262-18253-X.