Janurary 8, 2020

Master's Thesis in Physics

Evaluation of Automatic Spike Sorting Algorithms: Reproducible Workflows for Comparison with Ground Truth and Surrogate Data

Author
Shashwat Sridhar

Supervisor
Dr. Michael Denker

presented to
RWTH Aachen University
Faculty of Mathematics, Computer Science and Natural Sciences

performed at
Jülich Forschungszentrum
Institute of Neuroscience and Medicine (INM–6)
Institute for Advanced Simulations (IAS–6)

Reviewers
Prof. Dr. Sonja Grün
Prof. Dr. Andreas Offenhäusser

Declaration of Authorship

I, Shashwat Sridhar, declare that this thesis titled, "Evaluation of Automatic Spike Sorting Algorithms: Reproducible Workflows for Comparison with Ground Truth and Surrogate Data" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Acknowledgements

There are many people towards whom I'd like extend my gratitude for helping me through this thesis.

First and foremost, I'd like to thank my group leader, Prof. Dr. Sonja Grün, for giving me the opportunity to work on this thesis as a part of her research group at the INM – 6 of the Forshungszentrum Jülich, and for guiding me with valuable feedback.

This thesis would not have seen the light of day without the patient and encouraging guidance of my supervisor, Dr. Michael Denker. He has been a great supervisor and I cannot thank him enough for it.

Any amount of appreciation and love I show for my colleagues at the INM – 6 would be insufficient. From master's and PhD students, to the PIs and the support staff, they all played a role in making my time at the institute memorable and comfortable. Lastly, I could not have done any of this without the comforting presence of my family and my friends – distant and close. They have been an endless source of love and support.

Contents

1. Introduction

The brain, regardless of the organism it belongs to, is a wonderfully complex organ that functions as an information handling device, allowing the organism to interact with the physical world around it. It receives sensory input from various parts of the organism's body, processes this information and returns appropriate responses back to the body. All of this information transfer – to, from and within the brain – happens through specialized cells called *neurons*. Neurons exhibit a large variability in their structure and function, across species, individuals and even brain regions, and facilitate the complex information processing capabilities of a brain. Thus, in order to unravel the complexity of the brain, we must begin by understanding how neurons process information.

Most functional properties of individual neurons are fairly well understood. What remains to be understood is the emergent behaviour of large ensembles, or networks, of neurons. It is by virtue of the network activity of neurons that the brain is able to exhibit higher cognitive function (Bressler, 1995). A major focus of neuroscientific research over the past half a century has been on probing and analysing this network activity, and although the field has seen significant progress, it remains a challenge to reliably measure the activity of multiple individual neurons simultaneously (Buzsáki, 2004).

There are several approaches through which the activity of neurons can be observed and analysed. In this thesis, we concern ourselves with the approach of extracellular electrophysiology, wherein the extracellular electrical activity of neurons is measured at high temporal precision. Specifically, we evaluate the quality of algorithms used for a crucial pre-processing step in electrophysiolgical data analysis, called *spike sorting*, with the help of state-of-the-art tools.

We begin this chapter with a brief introduction to electrophysiology, delving into the history and state-of-the-art of spike sorting. This is followed by a brief introduction to electrophysiolgical data. Lastly, we outline the structure of the rest of this thesis.

1.1. Electrophysiological Background

Each neuron, in its resting state, maintains a constant electrical potential difference between its intracellular medium and the extracellular medium (known as the *membrane potential*) by means of active ion channels located on its cell membrane (Llinas, 2008). When the neuron is stimulated, this equilibrium is disturbed. If the disturbance exceeds a cell-specific threshold, the membrane potential at the location of the stimulus rapidly rises and falls, termed as a *depolarisation*, which causes adjacent regions of the cell to depolarise in turn. This depolarisation propagates through the length of the cell, and is called an *action potential*. Such depolarisations occur on the scale of millivolts and

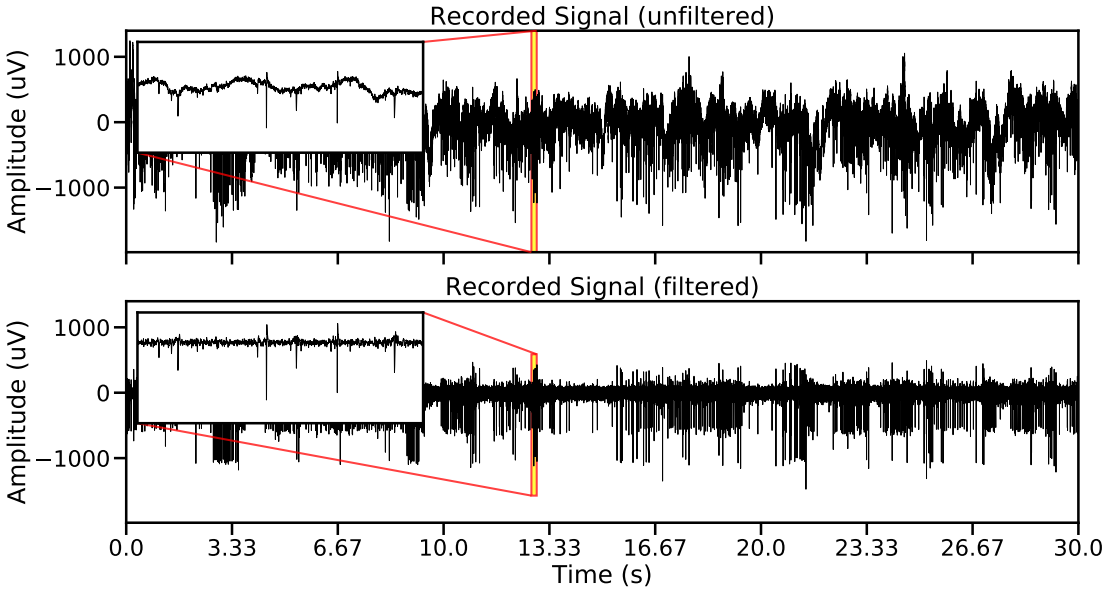


Figure 1.1.: Unfiltered and filtered signals from a single electrode. The two panels show the same 30 seconds of recorded data before filtering (top panel) and after filtering (bottom panel) using a second order Butterworth bandpass filter (250Hz - 5000Hz). The insets contain zoomed versions of the respective signals, clearly showing the spiking activity in the form of fast and large negative deviations. Filtering makes it easier to extract neuronal activity with the use of a threshold.

can be measured using probes placed inside or around the cell. This forms a basis for electrophysiology, wherein the electrical properties and dynamics of neurons are studied by measuring the voltage fluctuations inside (intracellular) or outside (extracellular) the cell. Intracellular recording methods are limited by the number of neurons that can be simultaneously measured. To record from large ensembles of neurons, extracellular recording methods must be used. In this thesis, we concern ourselves with extracellular recordings only.

Typically in extracellular electrophysiology, probes are inserted into brain tissue (*in vivo* or *in vitro*) to continuously record the extracellular voltage against a reference potential. The fluctuations in voltage arise largely due to the collective electrical activity of neurons in the region around the tip of each probe. The top panel of Figure 1.1 shows an example of such an extracellular signal recorded at a sampling rate of 30,000 samples per second, i.e. 30 kHz. Throughout this thesis, we refer to this time-ordered sequence of voltage measurements as the *recorded signal*. The recorded signal shows large fluctuations across a range of frequencies. However, on filtering out low (below 250Hz) and high (above 5000Hz) frequencies from the same signal (bottom panel of Figure 1.1), one observes many fast, large negative deviations that were not apparent in the recorded signal. The filtered form of the recorded signal will hereafter be called the *filtered signal*. A large fraction of these deviations correspond to individual action potentials of neurons

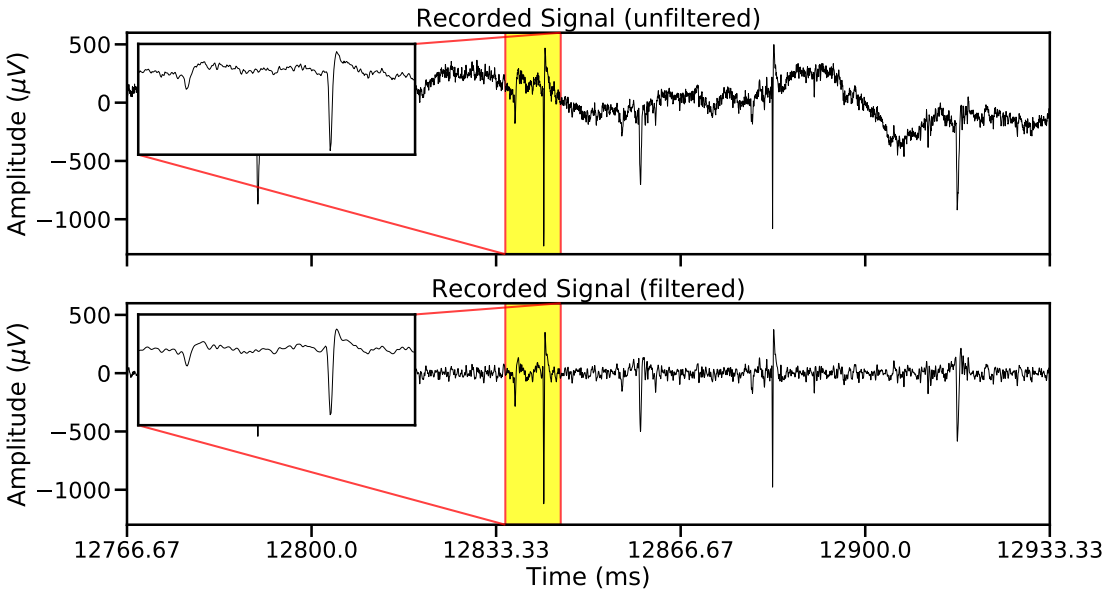


Figure 1.2.: Zoomed view of unfiltered and filtered signals on a single electrode. The two panels show the unfiltered (top panel) and filtered (bottom panel) signals from the insets of the corresponding panels of figure 1.1. The extended shape of each waveform becomes apparent in the insets of this figure. Additionally, two distinct waveforms shapes can be seen in the insets.

in the immediate vicinity ($<50\mu m$) (Buzsáki, 2004) of the electrode tip, and are called *spikes*, owing to their pulse-like appearance. Some of the deviations, though, may not have neuronal origins and could be artefects of the recording setup.

On zooming further into the filtered signals shown in the insets of Figure 1.1, we obtain the signals shown in Figure 1.2. Here, we observe that each spike has an extended shape (spanning on the order of 1-2 ms), which we call the *waveform* of the spike. Each waveform corresponds to a single extracellularly observed action potential of a single neuron in the vicinity of the probe. This is the neuronal activity we seek in electrophysiolgical data.

1.1.1. Spike Sorting

The brain has a high density of neurons (for primate motor cortex: 21,500 per mm^3; for human motor cortex: 10,500-30,000 per mm^3; human visual cortex: 106,000 per mm^3; (Abeles, 1991)). A probe inserted into the primate motor cortex tissue, for example, should have on the order of 10 neurons in its immediate vicinity (extending calculations from Pedreira et al., 2012). In practice, however, we observe discernible waveforms from, on the order of, 2 to 5 neurons, the reasons for which are not agreed upon (Henze et al., 2000; Shoham et al., 2006; Pedreira et al., 2012). In addition, the shape of a given neuron's waveform is characteristic, in that whenever the neuron "fires" (i.e. depolarizes

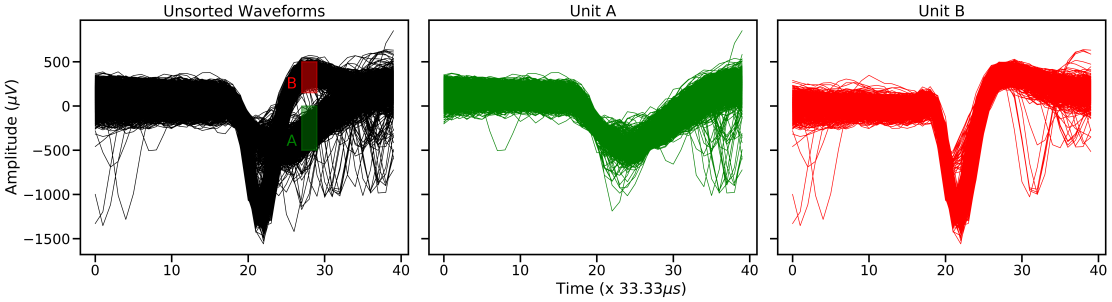


Figure 1.3.: Illustrative example of spike sorting using window discriminators. The left most plot shows a set of unsorted waveforms obtained using a simple threshold on a filtered signal (typically seen on an oscilloscope in real time). The green (A) and red (B) windows are regions defined by the data analyst such that all waveforms crossing either of the two regions are assigned to clusters A and B, respectively. The second and third plots show the green (A) and red (B) clusters obtained at the end of this exercise. This simple spike sorting technique is effective at quickly separating waveforms with significantly different shapes, however, it fails when the shapes are very similar.

to emit an action potential), the resulting waveform has a stereotypical shape (Harris et al., 2000).

In order to analyse the activity of neurons around a probe, it must first be extracted from the filtered signal. In the nascent days of extracellular electrophysiology, probes were inserted into brain tissue and waveforms corresponding to events where the recorded signal crossed a user-set threshold were displayed on an accompanying oscilloscope. When more than one shape was discernible, methods such as window discriminators were used to separate the different waveforms shapes into putative groups called *units* Figure 1.3 (Abeles and Goldstein, 1977; Lewicki, 1998). Each unit putatively corresponds to one neuron. The term unit is used to reflect the uncertainty in the identity of the clusters. Although such methods could be used to extract neuronal activity in "real time" (during the recording), they are not effective at differentiating units with similar waveform shapes and work only when the waveforms are significnatly larger than the background signal. They also do not scale well with a large number of probes, since the threshold would have to be manually set for signals from each probe.

Alternatively, instead of distinguishing waveforms corresponding to different neurons, the average population activity of neurons around the probe can be extracted. This activity is termed as *multiunit activity (MUA)*, and is shown to be a good predictor of behaviour (Stark and Abeles, 2007). MUA can either be obtained by calculating the root-mean-squared (RMS) value of the filtered signal (Stark and Abeles, 2007), or by using a threshold to flag all events which exceed it in the filtered signal Figure 1.4 (Supèr and Roelfsema, 2005). In the first case, the MUA is a continuous signal representing the population activity, whereas in the second case, the MUA is a collection of discrete time stamps where the signal exceeded the threshold. In either case, the MUA provides a measure of the average activity of the population of neurons around the tip of the probe.

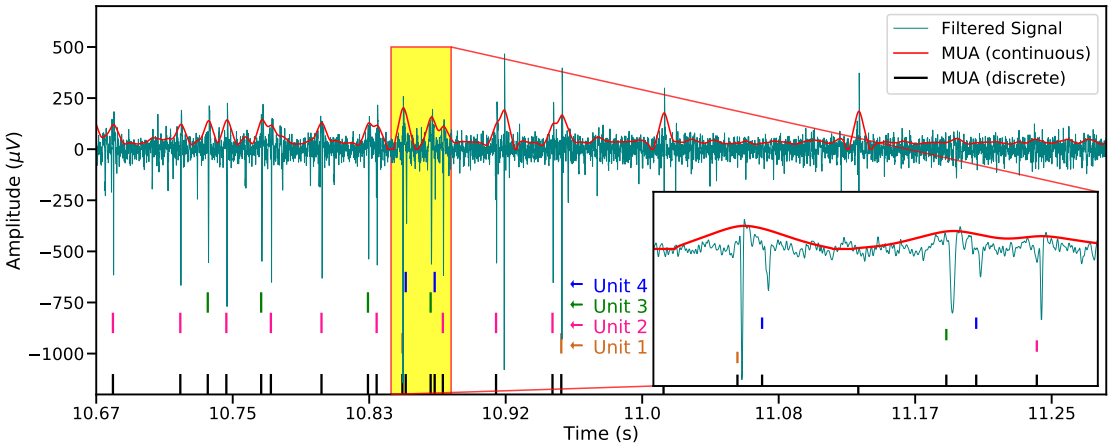


Figure 1.4.: Difference between SUA, continuous MUA and discrete MUA. A snippet of filtered signal is shown along with the continuous MUA signal overlayed in red. Black vertical bars at the bottom constitute the discrete MUA and represent times of threshold crossing. Coloured vertical bars (not black) show different single units obtained at the end of spike sorting. The inset shows a magnified portion of the signal and the colour bars. Each spike time is flagged at the time of threshold crossing, which always occurs before the spike minimum, as is seen clearly in the inset.

It is important to understand that MUA does not attempt to identify the individual neurons which contribute to the population activity, and as a result, cannot be used to find the tuning of individual neurons to behaviour (Supèr and Roelfsema, 2005).

The most commonly used approach to extracting the underlying neuronal activity, however, involves the extraction and segregation of the activity of multiple single neurons from the filtered signal. This process is called *spike sorting*. It has been the standard procedure for extracting the activity of individual neurons from electrophysiological data for several decades (Gerstein and Clark, 1964; Abeles and Goldstein, 1977; Schmidt, 1984). It enables the correlation of the activity of individual neurons to behaviour, facilitates analyses which require precise spike timing and has proven to be an indispensable part of any electrophysiological toolkit (Buzsáki, 2004).

Spike sorting can be thought of as a signal processing problem, wherein the aim is to isolate the constituent components of a complex signal (Carlson and Carin, 2019). The constituent components for a recorded signal are 1. the neuronal activity (in the form of a train of waveforms) from neurons in the immediate vicinity of the electrode tip, and, 2. the background signal composed, in particular, of the activity of neurons further away from the electrode tip. However, without knowing how many proximal neurons contribute to the neuronal activity, it is difficult to demix the complex signal.

Instead, the classical approach to spike sorting involves extracting the largest waveforms (corresponding to the nearest neurons) from the recorded signal and grouping them together based on the similarity of their spike shape. This transforms spike sorting into a classification and unsupervised clustering problem, wherein each large positive or neg-

ative deviation in a given recorded signal is classified as a spike or not a spike, and the classified spikes are then clustered without knowledge of the correct number of clusters (i.e. units). For the past several decades, this exercise has been typically performed by human experts with the aid of computational and graphical tools (Lewicki, 1998).

A typical manual spike sorting workflow involves three steps – filtering, extraction and clustering. We discuss each of them briefly (summarized from Rey et al., 2015).

Filtering Figure 1.1 shows that individual waveforms in the recorded signal become apparent only after the signal is filtered. Due to this effect, a high-pass or band-pass filtering is often the first step in spike sorting since it removes frequencies from the recorded signal which mask the neuronal activity. There is no consensus on the range of frequencies to filter out, since this also depends on the sampling rate of the recorded signal. It is best to use acausal or zero-phase filters in order to reduce the effect of filtering on the shape of the extracted waveforms (Lewicki, 1998). The basic idea, however, is to remove very low and very high frequencies, enabling the isolation of individual waveforms.

Extraction Extraction refers to the process of isolating waveforms from the filtered signal. This is typically done by setting a threshold above or below the filtered signal and flagging all events which cross this threshold. The time stamp corresponding to threshold crossing is termed a *spike time*. The threshold is measured in volts, or in units of standard deviations (SD) around the mean of the signal. Once events have been flagged across a signal, pieces of the filtered signal in a time window (1-3ms) around the spike time are stored. These are the waveforms we mentioned earlier. The time window is chosen based on the nature of the filtered signal. These waveforms are passed on to the clustering step.

Clustering The final step in spike sorting involves segregating these waveforms into clusters such that all waveforms within one cluster have a similar shape. This is, in principle, an unsupervised clustering problem since we do not know the correct number of underlying clusters. Typically, clustering is performed after extracting relevant features of waveforms which best describe the variability inherent in them. Principal component analysis (PCA) is the most commonly used method in this regard and has replaced the conventional approach of selecting intuitive features such as waveform height and peak-to-peak amplitude (Lewicki, 1998). The clustering itself is done on the extracted features and may be performed using a range of algorithms, such as template matching or k-means clustering, each of which involving different levels of manual intervention (Sukiban et al., 2019). The clusters of waveforms obtained at the end of this step constitute units.

The result at the end of spike sorting includes the spike times and waveforms corresponding to each identified unit in the recorded signal. This is, however, a simplified description of the exercise, only considering the case where recorded signals are measured from a single electrode.

1.1.2. Limitations of Manual Spike Sorting

Over the past few decades, new probes with multiple recording electrodes have been introduced, owing to advances in electrode fabrication technology. These multi-site probes have varied configurations, such as stereotrodes (McNaughton et al., 1983), tetrodes (Gray et al., 1995) and multielectrode arrays (Gross et al., 1977; Blanche et al., 2005; Gross, 2011). There are two main advantages of using multiple electrodes to record from the same population of neurons. First, it becomes possible to record from many more neurons simultaneously. Second, in the case of densely arranged electrodes, it becomes easier to separate the activity of different neurons since their activity is independently measured at different locations and can, thus, be better distinguished.

Certain multi-site probes, such as the Utah Array (Maynard et al., 1997), can be implanted chronically into brain tissue, opening up avenues for more advanced experiments where neuronal activity of awake and behaving animals can be measured (Brochier et al., 2018). More recently, arrays with up to thousands of recording sites have been developed, allowing for thousands of neurons to be recorded simultaneously (Tsai et al., 2017).

Classical spike sorting methods, which involve the manual processing of recorded signals from individual electrodes, quickly become impractical as the number of electrodes increases. Particularly with data recorded using probes with a dense arrangement of electrodes, where the activity of single neurons can be detected on multiple electrodes, manual spike sorting is no longer possible (Einevoll et al., 2012).

Manual spike sorting results are also subject to a large variability when performed by different human experts, and tend to have large error rates, with up to 23% false positives and 30% false negatives reported on synthetic data (Wood et al., 2004b). This not only affects the reliability of the spike sorting result, but also makes the entire procedure non-reproducible. The accuracy of spike sorting has been shown to have a direct effect on the quality of higher order analysis (Brown et al., 2004; Pazienti and Grün, 2006), underlining the requirement for more accurate alternatives to manual spike sorting.

Lastly, manual spike sorting is time and labour intensive. The manual spike sorting of a typical 15 minute session recorded on a 96-channel Utah array, performed using template matching, involves the following steps (obtained via personal communication from Alexa Riehle):

1. Loading the data (approx. 5 minutes)
2. Choosing filter settings for all data ($<$ 1 minute)
3. For each electrode:
    a) Load recorded signal (approx. 1 minute)
    b) High pass filter signal ($<$ 1 minute)
    c) Inspect and set detection threshold to obtain spikes and waveforms (approx. 1 minute; often repeated to obtain clear spike waveforms)
    d) Remove artefacts, that is, signals which are extremely large (approx. 1 minute)

e) Run automatic k-means clustering to obtain putative clusters (approx. 2 minutes)

f) Select cluster centers to determine waveforms for template matching ($<$ 1 minute)

g) Run template matching ($<$ 1 minute)

h) Curate clusters by splitting or merging existing ones (approx. 2 minutes)

4. Save time stamps and waveforms (approx. 2 minutes)

The total time required to sort a dataset can, thus, take on the order of a few hours. If the number of recording channels were to increase, the required time for spike sorting would increase as well. Given the rate at which new data can be recorded, this is not a viable solution for longer recordings.

1.1.3. Automatization of Spike Sorting

In response to these challenges, several automatic spike sorting algorithms have been proposed. They employ different techniques, including hierarchical clustering (Fee et al., 1996), Independent Component Analysis (ICA) (Takahashi et al., 2003a,b), Gaussian mixture model Expectation Maximization (Wood et al., 2004a), superparamagnetic clustering (Quiroga et al., 2004), template matching (Franke et al., 2015; Yger et al., 2018), non-parametric Bayesian clustering (Lee et al., 2017; Wood et al., 2006) and non-parametric density-based clustering (Chung et al., 2017), among many others. Several review papers summarize the progress in this field over time and provide a good overview of the state-of-the-art of automatic spike sorting (Lewicki, 1998; Rey et al., 2015; Lefebvre et al., 2016; Carlson and Carin, 2019).

All these algorithms accept parameters which control the outcome of spike sorting. Such parametisation not only improves the reproducibility of the procedure, it also allows for better provenance capture and quality control of the analysis workflow, enabling easier data sharing and collaboration.

1.1.4. Evaluation of Spike Sorting Algorithms

The automatization of spike sorting does not ensure its accuracy, and each automatic algorithm must be evaluated on its ability to accurately sort a given dataset. This kind of an evaluation boils down to determining whether the spike times and cluster (unit) labels found by a certain algorithm are correct or not, in turn requiring prior knowledge of all correct spike times and unit labels in a given dataset. In other words, such an evaluation requires knowledge of the *ground truth*. Due to the nature of extracellular electrophysiolgical recordings, this ground truth is typically not available.

In the absence of ground truth, an algorithm can either be evaluated using metrics which don't require a ground truth, or by comparing its results to the closest approximation of the ground truth.

Additionally, the set of parameters of an algorithm which yields accurate results with one dataset may not yield accurate results with another. It is, thus, important to determine the set of parameters of an algorithm which yield the optimal spike sorting result for a given dataset. This is achieved by running the algorithm on that dataset for all possible parameter sets and evaluating the results from each. This amounts to performing *parameter scans* for each algorithm that we wish to evaluate.

While strategies and metrics have been proposed for the evaluation of spike sorting algorithms in the absence of ground truth (Pouzat et al., 2002; Schmitzer-Torbert et al., 2005; Neymotin et al., 2011; Barnett et al., 2016), these become impractical when used in combination with large parameter scans. The evaluation of such algorithms in the presence of ground truth is straightforward, since it involves the direct comparison of spike times and unit labels. There is, however, no standard framework to carry out such evaluations while performing large parameter scans.

1.2. Aim of this Thesis

In this thesis, we evaluate the performance of two spike sorting algorithms – Mountainsort (Chung et al., 2017) and Spyking Circus (Yger et al., 2018) – by performing parameter scans using scalable spike sorting pipelines. We compare the results of the algorithms with two kinds of ground truth – manually spike sorted data and synthetically generated surrogate data – in an effort to determine the extent to which these algorithms can replace manual spike sorting. We carry out the comparisons using a combination of standard and derived metrics. We, thus, propose an analysis framework using which any automatic spike sorting algorithm can be evaluated.

1.3. Data

Throughout this thesis, we use a single dataset – `i140703-001` – recorded from the motor cortex of a macaque monkey while it performs a behavioural task (Brochier et al., 2018). The data was recorded at a sampling rate of 30KHz using a Utah Array (Blackrock Microsystems, Salt Lake City, Utah, USA), which contains 96 electrodes on a single 10-by-10 grid. Each electrode on the array is $1.5mm$ long and the spacing between two adjacent electrodes is $400\mu m$. The manual spike sorting of this dataset was performed by a human expert (A. Riehle) with the help of the Plexon Offline Sorter (version 3.3.3), the results of which were taken as ground truth for the evaluation of the automatic spike sorting results.

1.4. Structure of this Thesis

We begin this thesis with a thorough description of the two selected automatic spike sorting algorithms and the reasons behind choosing them, in Chapter 2. Then, in Chapter 3, we describe how we generate surrogate datasets as synthetic ground truth. We use this surrogate data as ground truth when evaluating the selected algorithms. In Chapter 4,

we focus on the pipelines we developed for our analysis and in Chapter 5, we discuss in detail the various metrics which we use to compare the algorithms to ground truth. The results of these comparisons are given in Chapter 6. Chapter 7 and Chapter 8 wrap up the contents of this thesis and suggest ways in which our work can be continued.

2. Automatic Spike Sorting

This chapter is dedicated to the spike sorting algorithms that we have chosen to evaluate. We first look at the reasons behind choosing these algorithms, and list other algorithms which were considered. This is followed by a description of the algorithms themselves, focusing on the similarities and differences between the two approaches. We also list all the algorithm parameters which we use as a part of our analysis, and try to provide an intuitive explanation for each. Finally, we briefly compare the implentations of the two algorithms.

2.1. Selected Algorithms

The choice of automatic spike sorting algorithms used in this work was based on the following criteria:

1. The data analysed in this work was recorded using Utah Arrays which have a spacing of $400\mu m$ between neighbouring electrodes. This inter-electrode spacing prevents the activity of any given neuron to be observed on multiple electrodes, implying that the signal from each electrode can be spike sorted independently. Thus, the automatic spike sorting algorithms should be able to sort data from independent electrodes.

2. The time required by the human expert to sort one of our datasets is on the order of a few hours per dataset. The automatic spike sorting algorithms should be able to perform the analysis in comparable or less time.

3. The software implementations of the algorithms should not be restrictively tedious to setup and run. It should take as input the recorded voltage signals and yield spike times and cluster labels as output. The algorithm must be well documented, with a clear description of the parameters used. If the algorithm could be incorporated into an automated workflow, it would be an added benefit.

Several algorithms were available in literature at the time of choosing. The following is a list of algorithms that were considered:

1. MountainSort (Chung et al., 2017)

2. Spyking Circus (Yger et al., 2018)

3. Klusta (Rossant et al., 2016)

4. WaveClus (Chaure et al., 2018)

5. Tri des Clous (Garcia and Pouzat, 2016)

MountainSort and Spyking Circus were chosen because they satisfied all the criteria listed above. We will now discuss these two algorithms in detail.

2.2. MountainSort

MountainSort[1] was created with the goal of developing a completely automatic spike sorter, requiring little to no human intervention (Chung et al., 2017). The emphasis is on having as few free parameters as possible, to eliminate any subjectivity involved in choosing parameters. The algorithm does not make any assumptions regarding the nature of the noise or the spiking activity, and thus, is agnostic of the brain region from which the data is recorded. The only assumptions made are: 1. Each cluster of events representing a neuron has a uni-modal density function, and, 2. Two such clusters can be separated by a hyperplane around which there is a relatively low density of data points The algorithm is divided into three phases, each containing multiple steps:

1. Pre-processing
    a) Filtering
    b) Whitening
2. Sorting
    a) Event Detection
    b) Feature Extraction
    c) Clustering
    d) Fitting
3. Post-processing: Curation and Annotation

Each step in the algorithm has been implemented with high-density MEAs in mind and is thus parallelized for high computational efficiency. The datasets we analyze do not make use of this parallelization, so the details of the parallelization are not discussed here. Additionally, certain steps are only relevant for high-density MEA data and, hence, are only discussed briefly.

2.2.1. Filtering + Whitening

A bandpass filter is used to filter the input time-series data using a Fast Fourier Transform. The low-pass and high-pass frequencies are user-defined. This is followed by a whitening step to remove spurious correlations between recording channels (electrodes).

[1] https://github.com/flatironinstitute/mountainsort_examples/blob/master/README.md

2.2.2. Event Detection

Potential spiking events on each electrode (and its corresponding neighbourhood for high density electrode arrays) are obtained by subjecting the pre-processed time-series to a voltage threshold. Any time stamp t_0 that meets the following two criteria is flagged as an event time

$$|Y(t_0)| > \mu \tag{2.1}$$
$$|Y(t_0)| > |Y(t_1)| \text{ for all } |t_o - t_1| \leq \tau \tag{2.2}$$

where $Y(t_0)$ is the voltage at time t_0, μ is the detection threshold in units of standard deviations away from the mean and τ is the minimum allowable interval between two subsequent events on the same recording channel. These events are detected for each electrode neighbourhood independently. Chung et al. (2017) claim that the results of the sorter are independent of the detection threshold, provided it is low enough.

2.2.3. Feature Extraction and Clustering

For each event flagged in the previous step, the corresponding clip from the pre-processed time-series is extracted. The clip is of size T centered at time t_0. The PCA features of all the extracted clips is then computed and a clustering is performed on these feature vectors. The clustering method is called ISO-SPLIT and makes use of a statistical test called ISO-CUT internally. Both these are briefly described here.

2.2.3.1. ISO-SPLIT

Given the set of n-component feature vectors (n corresponding to the dimensions retained after PCA) as points in an n-dimensional space, the method begins with an over-clustering of the points and then iteratively redistributes the points or merges the clusters until convergence is achieved. The decision to redistribute or merge two given clusters is performed in two steps. First, the points in the two clusters are projected onto a line of optimal discrimination and second, this projection is tested for unimodality using the ISO-CUT statistical test (described below). If the projection is deemed unimodal, the clusters are merged. If not, the ISO-CUT yields an optimal cut point around which the points are redistributed. This is repeated until all cluster pairs have been compared once. Then the algorithm is deemed to have converged.

2.2.3.2. ISO-CUT

For a one-dimensional sample of values, this statistical test determines whether the sample is unimodal or not. If it is not unimodal, it also determines the optimal cut point for clustering. A maximum-likelihood unimodal fit to the distribution of samples is computed and the two distributions are compared using the Kolmogorov-Smirnov statistic. If the value of this statistic exceeds a subjective threshold of 1, unimodality is rejected.

When unimodality is rejected, the method returns the point of minimal density in the distribution, but calculates it non-parametrically without any density estimates. The authors claim this allows the method to handle cases where the densities of the two clusters are very different (for example, in the case one rapidly firing neuron and one sparsely firing neuron).

This entire step does not take any parameters, making it robust and reproducible. The result of this clustering would depend on the initial choice of clusters, but the authors claim that given a fine enough parcellation (high over-clustering), the results are independent of the initial clustering. This clustering is performed independently for every electrode neighbourhood.

2.2.4. Cluster Consolidation

So far in the algorithm, no attempt was made to handle cases wherein the same units were detected on multiple electrodes. Often, the approach to solving this problem involves merging clusters across electrodes, but the authors argue against merging, specifically quoting computational complexity, high chance of errors and non-transitivity of merge decisions (i.e. clusters A and B to be merged, and clusters B and C to be merged, but clusters A and C not to be merged). They propose an alternative approach called cluster consolidation, wherein, amongst all the conflicting clusters, the cluster with the highest peak in the average signal is retained. This is accomplished in two passes. In the first pass, clusters with peak average amplitudes significantly smaller than the maxiumum are discarded. In the second pass, the algorithm iterates over all clusters ordered by the absolute peak average waveform amplitude and looks for duplicates. If a duplicate is detected, the cluster with the lower absolute peak average waveform amplitude is rejected. Two clusters must fulfill the following criteria to be considered duplicates in the second pass: 1. they must have been detected on different electrodes, 2. they must have a comparable amplitude (since clusters with very different amplitudes were discarded in the first pass), and, 3. they must have at least 50% coincident events It is important to note here that for the datasets analyzed in this work, this step is not of importance because the electrode arrays used for recording have an inter-electrode distance that prohibits the same neuron from being detected on multiple electrodes, thus doing away with the need for cluster consolidation.

2.2.5. Fitting

The final step in the sorting phase of the algorithm targets individual threshold crossing events that were found on more than one electrode neighbourhood but were incorrectly assigned to non-conflicting clusters, and thus survived the consolidation stage. To this end, for every set of conflicting events, they choose that event which when removed from the pre-processed time-series, causes the largest reduction in the l^2-norm of the time-series. (Chung et al., 2017) discuss this in more mathematical detail in their work. For the purposes of this thesis, it suffices to say that this step removes events that may have been flagged twice - either on the same electrode, or on different electrodes. They claim

this step deals with overlapping spikes to a certain extent while still maintaining the original clustering labels assigned by previous steps.

2.2.6. Cluster Metrics and Curation

Althought not a part of the main sorting phase, this step is intended to refine the quality of the sorting. It essentially replaces the manual curation of sorting results with an automated implementation. For each cluster obtained in the sorting phase, a set of metrics are calculated. The user can set parameters which act as threshold criteria on these metrics, and all clusters that don't meet the criteria are rejected. The metrics are discussed below.

2.2.6.1. Isolation

The isolation of a cluster is a measure of how well it is separated from other clusters in the same electrode neighbourhood. For two clusters $A = \{a_1, a_2, \dots a_n\}$ and $B = \{b_1, b_2, \dots b_p\}$, where a_i and b_j represent the i^{th} and j^{th} clips of clusters A and B, respectively, the k-nearest neighbour overlap between A and B is defined as

$$m_{\text{overlap}}(A, B) = \frac{1}{N} \sum_{j=1}^{N} \frac{\#\{x \in A \cup B : \rho(n_j(x)) = \rho(x)\}}{\#(x \in A \cup B)}, \tag{2.3}$$

where $\rho(x)$ is the membership function such that $\rho(a_1) = 1$ and $\rho(b_1) = 2$, and $n_1(x) \dots n_k(x)$ are the k nearest neighbours of x in $A \cup B$. N is the number of points sampled for this calculation, chosen such that $N = min(500, min(size(A), size(B)))$, where $size(X)$ is the total number of points in cluster X and $min(p, q)$ returns the lower value of p and q. Once the overlap between A and all other clusters in its neighbourhood is calculated, the isolation score of A is given by

$$m_{\text{isolation}}(A) = 1 - \underset{\text{clusters X}}{MIN} \left[m_{\text{overlap}}(A, X) \right], \tag{2.4}$$

where $\underset{\text{clusters X}}{MIN} [f(P)]$ is the minimum of all values returned by any function $f(P)$ for all values of P, and X represents the set of all other clusters. In the algorithm, the user can set a minimum isolation score threshold, so that all units with an isolation score below this threshold would be rejected.

2.2.6.2. Noise Overlap

As the name suggests, the noise overlap metric measures the overlap of a cluster, say $A = \{a_1, a_2, \dots a_n\}$, with noise. The noise is generated artificially and is defined to be a cluster $Q = \{q_1, q_2, \dots q_n\}$ of n clips extracted from the pre-processed time-series at random time stamps. The clips in both Q and A are adjusted to remove the weighted average of the expected noise waveform, yielding the adjusted clusters $\tilde{Q}$ and $\tilde{A}$. The

noise overlap of A is then defined as

$$m_{\text{noise}}(A) = m_{\text{overlap}}(\tilde{A}, \tilde{Q}). \tag{2.5}$$

A maximum value of the noise overlap score can be set by the user, so that all units with a noise overlap greater than this threshold would be rejected.

2.2.6.3. Firing Rate

Given a cluster of event waveforms $A = \{a_1, a_2, \ldots a_n\}$ obtained from the pre-processed time-series of one electrode $Y_e(t) = \{t_1, t_2, \ldots, t_n\}$, the firing rate λ for the cluster is defined as

$$\lambda = \frac{size(A)}{t_n - t_1}, \tag{2.6}$$

where $size(A)$ is the total number of time points in A and n is the number of time points in the pre-processed time-series. For a real neuron, this quantity can be simply defined as the total number of spiking events per second. The firing rate threshold parameter allows the user to choose the minimum firing rate for the sorted units.

2.2.6.4. Cluster Signal-to-Noise Ratio

The signal-to-noise ratio (SNR) of a cluster of events is defined as the peak absolute amplitude of the average waveform divided by the peak standard deviation (SD). It is a measure of how much larger the waveform signal is compared to the underlying noise in the signal. Formally, given a cluster of event waveforms $W = [w_1, w_2, \ldots w_n]$ where each waveform is a vector $w_i = [a_{i1}, a_{i2}, \ldots, a_{iT}]$, we obtain the average waveform $\bar{W}$ as

$$\bar{W} = \left[\frac{1}{n}\sum_{i=1}^{n} a_{i1}, \frac{1}{n}\sum_{i=1}^{n} a_{i2}, \ldots, \frac{1}{n}\sum_{i=1}^{n} a_{iT}\right], \tag{2.7}$$

$$\Rightarrow \bar{W} = \left[\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_T\right], \tag{2.8}$$

and the standard deviation vector W_σ as

$$W_\sigma = \left[\sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_{i1} - \bar{a}_1)^2}, \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_{i2} - \bar{a}_2)^2}, \ldots, \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_{iT} - \bar{a}_T)^2}\right], \tag{2.9}$$

$$\Rightarrow W_\sigma = \left[a_{\sigma 1}, a_{\sigma 2}, \ldots, a_{\sigma T}\right]. \tag{2.10}$$

The SNR is calculated as

$$SNR = \frac{max(\bar{W})}{max(W_\sigma)}. \tag{2.11}$$

The peak SNR threshold parameter sets a minimum for the peak SNR value for the units.

2.2.7. Summary

The MountainSort package is designed to take as few parameters as possible to perform a completely automatic spike sorting. In this work, we attempt to find the parameter set that would yield the best spike sorting results. To that end, we perform parameter scans over all available parameters for MountainSort, which are listed here:

1. `detect_threshold` *(μ)* - threshold which the pre-processed time-series must cross to be flagged as an event
2. `detect_interval` *(τ)* - minimum allowable time difference between two subsequent events flagged on a certain channel
3. `clip_size` *(T)* - length of the event vectors extracted from the pre-processed time-series
4. `firing_rate_threshold` *(λ)* - threshold for the minimum firing rate of units
5. `isolation_threshold` *($m_{isolation}$)* - threshold for minimum isolation of units
6. `noise_overlap_threshold` *($m_{overlap}$)* - threshold for maximum overlap of units with noise
7. `peak_snr_threshold` *(SNR)* - threshold for minimum cluster SNR

In addition to the parameters listed here, there are other essential parameters required by the algorithm to successfully sort the data. These are: 1. *sampling rate* - rate of data collection in Hertz 2. *filter frequencies (low and high)* - frequency (in Hertz) used to band-pass filter the original time series; we used 250 Hz and 5000 Hz for low and high cut-offs throughout this thesis 3. *electrode layout* - the relative mapping of the electrodes is required by the algorithm in cases where the electrodes are dense and clusters needs to be consolidated across electrodes 4. *adjacency radius* - the radius within which neighbouring electrodes are expected to record from the same neurons (measured in the same units as used for the *electrode layout*) 5. *detect sign* - a binary (+1 or -1) parameter that determines whether a positive or a negative threshold will be used to detect threshold crossing events

2.3. Spyking Circus

Spyking Circus[2] is a spike sorting toolbox designed to be able to sort data from large, high-density multi-electrode arrays (Yger et al., 2018). The algorithm utilizes a clustering approach to determine centers of clusters corresponding to potential units and then uses template matching to finalize the spike sorting. One assumption made is that in a high-dimensional feature space, cluster centers of units will be points of maximum local density, although no assumption is made regarding the shape of the clusters. The clusters

[2] https://github.com/spyking-circus/spyking-circus

are characterized by templates and the templates are matched with the recorded signal in an attempt to "decompose" the recorded signal into two components (described in more detail below), as is given by the equation –

$$s(t) = \sum_{ij} a_{ij}\mathbf{w}_j(t - t_i) + b_{ij}\mathbf{v}_j(t - t_i) + \mathbf{e}(t), \tag{2.12}$$

where $s(t)$ is the recorded signal on a single electrode at time t, a_{ij} and b_{ij} are coefficients which determine the amplitude of the components $\mathbf{w}_j$ and $\mathbf{v}_j$, respectively, for the signal at the ith time point and jth template, and $\mathbf{e}(t)$ is the background signal, or noise. The idea is to represent the orignal signal as a linear sum of templates corresponding to putative individual neurons.

The algorithm consists of four major steps.

1. Pre-processing
    a) Filtering
    b) Spike detection
    c) Whitening
    d) Basis estimation
2. Clustering
    a) Masking
    b) Pre-clustering
    c) Clustering by local density search
    d) Defining centroids
    e) Merging
    f) Template estimation
    g) Removal of redundant templates
3. Template matching
4. Automated Merging

The efficiency and scalability of the algorithm results from a parallelized clustering step which divides the array into as many local neighbourhoods as there are electrodes and carries out the clustering on each neighbourhood independently.

Discussing each of these steps in detail here would involve repeating what has already been described in the original publication (Yger et al., 2018). We go through the prominent steps of the algorithm, without delving into all details.

2.3.1. Pre-Processing

A third order bandbass filter is used to filter the time-series data. Following this, putative spikes are detected by subjecting the time-series on each electrode k to a threshold θ_k defined as $\theta_k = \lambda \cdot MAD(s_k(t))$, where $s_k(t)$ is the time-series and $MAD(x)$ is the Median Absolute Deviation of x. The spike threshold multiplier, λ, is a free parameter. The spike times thus obtained are used to extract up to a maximum of 20 seconds of time series data that contains no spikes. This extracted signal is used to compute the covariance matrix of noise, which in turn is used to whiten the time-series data. The spike threshold θ_k is recomputed for the whitened time-series and new spike times are extracted. The extracellular waveforms (snippets) corresponding to each putative spike time are extracted and used for the basis estimation.

2.3.2. Basis Estimation

This step is aimed at reducing the dimensionality of a subset of the snippets extracted earlier. A subset N_p of all snippets are first upsampled, aligned to their minima and then downsampled to the original snippet size. Principal Component Analysis (PCA) is performed on the aligned snippets to reduce their dimensionality to N_{PCA} dimensions. These principal components are then used for clustering.

Clustering A clustering is performed prior to the template matching step in order to determine the putative clusters corresponding to different units. It is performed on a subset of events ($N_{\text{spikes}} \leq 10000$ in this thesis) collected randomly from the set of all threshold crossing events. The snippet corresponding to each selected event is projected onto the PCA basis estimated in the pre-processing step. Events on each electrode are clustered in parallel, which speeds up the entire process. In cases where a certain event is detected on multiple electrodes, it is flagged on the electrode with the lowest minimum of the corresponding snippet. The clustering itself is carried out in the following steps:

1. In the PCA space of the selected events, for each event x_i^k detected on electrode k, two measures are calculated:
   a) ρ_i^k, which is the mean distance of x_i^k to its S nearest neighbours ($S = 100$ for all analyses in this thesis)
   b) δ_i^k, which is the minimum distance of x_i^k to any other point $x_{m,m\neq i}^k$, such that $\rho_m^k \leq \rho_i^k$
2. Putative cluster centroids are defined by taking the $N_{\text{max clusters}}$ events with the highest δ/ρ ratio ($N_{\text{max clusters}}$ is set to 10 for all anayses in this thesis, and corresponds to the maximum expected number of units on any channel). The intuition here is that cluster centroids would be points of high local density (corresponding to low values of ρ) which are far apart from each other (leading to a high values of δ).

3. The remaining points (which were not classified as putative cluster centroids) are sorted by increasing ρ values and iteratively assigned to the same cluster as the nearest point with a lower ρ value.

4. Similar clusters are then merged if the normalized distance between them, ζ is less than a user-defined threshold σ_{similar}. ζ is defined as

$$\zeta(m,n) = \frac{\|\alpha_m - \alpha_n\|}{\sqrt{\gamma_m^2 + \gamma_n^2}}, \tag{2.13}$$

where α_m and α_n are the medians of the clusters m and n (taken to be the respective centres). γ_m (respectively, γ_n) is the dispersion of the projections of all events of cluster m (n) on to the axis joining the centroids of m and n (i.e. $\alpha_m - \alpha_n$) and is defined as

$$\gamma(m) = MAD(x_m \cdot (\alpha_m - \alpha_n)). \tag{2.14}$$

Additionally, clusters with less than 50 spikes ($\eta \cdot N_{\text{spikes}}$, where $\eta = 0.005$) were discarded.

5. For template matching, templates are estimated for all remaining clusters by calculating two components for each cluster – a) $\mathbf{w}_m$, the median of all snippets in the cluster, and b) $\mathbf{v}_m$, the direction of the largest variance of the projection of the snippets in a space orthogonal to $\mathbf{w}_m$

6. Template matching is performed on each electrode by iteratively fitting every threshold crossing event with the templates estimated on that electrode, such that both components, $\mathbf{w}_m$ and $\mathbf{v}_m$, satisfy certain requirements (see original publication).

2.3.3. Automated Merging

As an additional optional step, the cross-correllograms of spiketrains from all pairs of clusters obtained at the end of template matching are computed, and those pairs which show a high degree of similarity in firing activity (cross-correllograms and firing rates) are merged. Although this step is also parametrized, newer implementations of the tool provide an optional parameter `auto_mode` which automatizes this step completely. We have used this parameter in our analyses.

2.3.4. Summary

Spyking Circus is a tool which provides a lot of control over the spike sorting results. Specifically, each parameter described here (and additional ones described in the original publication) can be changed easily. Although not clear in the original publication, they do not recommend modifying most parameters. Through online documentation, user forums and personal communication with the authors, we arrived on 6 parameters which we believe would affect the sorting quality most greatly. These parameters are briefly summarized here.

1. `spike_thresh` (θ) - spike detection threshold measured in MADs of the pre-processed signal

2. `clip_size` (N_t) - temporal width of snippets used in the algorithm, measured in milliseconds

3. `sensitivity` - a clustering parameter which determines the sensitivity of the clustering procedure (internal parameter, with no explicit documentation)

4. `sim_same_elec` (σ_{similar}) - a clustering parameter which set a threshold for local cluster merges

5. `cc_mixtures` (cc_{similar}) - a clustering parameter which removes redundant templates; this parameter has been deprecated in recent versions of the tool, however the original publication describes this parameter in detail

6. `noise_thr` - a clustering parameter which determines the overlap of each cluster with noise (internal parameter, with no explicit documentation); this parameter has been deprecated in recent versions of the tool

7. `auto_mode` - a single parameter to control the automated merging after template matching

Additionally, the following parameters are required for spike sorting any dataset. 1. *sampling rate* (f_{rate}) - rate of data collection in Hertz 2. *filter frequencies (low and high)* - frequency (in Hertz) used to band-pass filter the original time series; we used 250 Hz and 5000 Hz for low and high cut offs throughout this thesis 3. *electrode layout* - the relative mapping of the electrodes is required by the algorithm in cases where the electrodes are dense and clusters needs to be consolidated across electrodes 4. *adjacency radius* - the radius within which neighbouring electrodes are expected to record from the same neurons (measured in the same units as used for the *electrode layout*) 5. *detect sign* - a parameter that determines whether a positive or a negative threshold will be used to flag events (takes values of `positive` or `negative`)

2.4. Overview and Comparison

The spike sorters described above take markedly different approaches to solve the problem of spike sorting. Mountainsort uses ISO-CUT + ISO-SPLIT to merge finely parcellated clusters of waveforms under the assumption that the waveforms of a single neuron form a unimodal density function in the high-dimensional space. It takes minimal free parameters and propounds a completely automatic approach to spike sorting. In the thesis, we attempt to vary those few parameters in an attempt to establish the extent to which spike sorting results can be optimized.

Spyking Circus, on the other hand, uses a heavily parametrized algorithm and attempts to estimate templates of clusters which would best recreate the original signal. It estimates the templates based on the idea that cluster centers (medians) would be

points of high local density - an assumption which is similar to that made by MountainSort. The template matching employed by Spyking Circus differs significantly from the Fitting step used in MountainSort. Notably, Spyking Circus allows for variations in the amplitude of the templates during template matching, which MountainSort does not explicitly account for. Additionally, Spyking Circus subtracts templates from the original signal after they have been matched, which should allow for a better detection of overlapping spikes, provided both overlapping spikes are detected as separate threshold crossing events.

In terms of usability, both spike sorters have well documented terminal-based user interfaces, with optional graphical interfaces to curate or merge clusters. MountainSort itself is written in Python, but uses the mountainlab-js package as a framework to execute in.

3. Surrogate Data (Ground Truth) Generation

This section deals with the generation of surrogate ground truth datasets. We begin by describing why surrogate data needs to be generated and the constraints it must adhere to. We then delve into the details of the methods used for the generation of these datasets. Lastly, we briefly compare real data and artificial data.

This thesis focuses on the comparison of the performance of spike sorting algorithms on real-world data. The challenge with spike sorting real data is that we typically do not know the underlying ground truth, making it difficult for us to evaluate whether the result from one sorting is better than another. One standard approach is to rely on the experience of human experts for spike sorting. However, this renders the analysis non-reproducible, since it is subject to human bias. Wood et al. (2004b) have shown that the results of spike sorting performed on the same data by different human experts vary by a large margin, bringing into question the reliability of manual sorting. Thus, in the absence of ground truth spiking information, comparing automatic algorithms only to manual spike sorting is not sufficient.

To objectively assess an algorithm's performance, it must be tested on ground truth data. There are two ways such ground truth data can be obtained. One way is to record the activity of individual neurons using patch-clamp techniques or optogenetics while simultaneously recording the voltage using electrodes (Shew et al., 2010; Hemberger et al., 2019). Such an approach is not always feasible due to the increased complexity of the experimental setup and due to the size of the brain limiting our ability to record from the same population of neurons using different techniques. The other way to obtain ground truth data is to generate it artificially using a model (Rey et al., 2015). This approach is easier to implement and is only limited by available computational power and our ability to accurately model the electrical activity in the extracellular areas of the brain. In this thesis, we take the second approach.

An important goal for us is to assess these spike sorting algorithms in the framework of a reproducible workflow (see Chapter 4). To that end, in the following sections, we point out how each step fits into our surrogate dataset pipeline. Additionally, each step of the pipeline accepts certain parameters, allowing the user to tailor the generated datasets to their needs. These parameters are described wherever necessary.

3.1. Generation of Ground Truth Datasets

To test the accuracy of spike sorting algorithms, we generate datasets that closely resemble real-world data. This section describes the process we use to generate such datasets.

The spike sorting algorithms we analyze take as input the unprocessed signal recorded from the microelectrode array (MEA). The unprocessed signal from a single electrode is composed of the time-ordered values of the extracellular voltage measured on that electrode, and we call it the *recorded signal*. We do not process the recorded signals before feeding them to the sorters, since both algorithms filter and whiten the signals as an internal pre-processing step. Our surrogate dataset pipeline, thus, generates artificial recorded signals as output.

A given recorded signal can be broken down into two components, classified on the basis of their relevance to spike sorting algorithms:

1. the *neuronal activity*, which is made of the extracellular action potentials originating from the neurons in the direct vicinity of the recording electrode

2. the *background signal*, which is the residual recorded signal once neuronal activity is removed

To generate surrogate data, we superimpose artificially generated neuronal activity onto background signals obtained from experimental data. The sorting algorithms then attempt to extract the neuronal activity from the artificial recorded signal, and we assess the performance of the sorter based on how well it is able to retrieve the embedded neuronal activity. To distinguish between the signals obtained on real electrodes in the experimental dataset from the signals we generate, from here on, we refer to each surrogate electrode as a *channel*.

The artificially generated neuronal activity is characterised by:

1. *Spike times*, which are the time stamps of all true spiking events in the recorded signal for each channel (spike times for each spike are taken as the time corresponding to the minimal voltage of the spike waveform).

2. *Unit labels*, corresponding to each spike time, that specify the correct neuronal identity of the spike, i.e. which unit (putative neuron) the spike originated from.

Any spike sorting algorithm also yields spike times and unit labels. Thus, the fundamental problem of evaluating a sorter's performance boils down to comparing the sorter's result to the ground truth spike times and unit labels. To this end, we organize our ground truth dataset so that it aids in this comparison. We generate two versions of each dataset and store each in a separate file. One version - termed the *sorter's version* - contains only the recorded signal from each channel. This is the version that is provided as an input to both sorting algorithms. The second version - termed the *evaluator's version* - contains the neuronal activity characterized by the spike times and unit labels. This version contains all the information we need to evaluate results of the spike sorting algorithms.

To summarize this distinction more concisely, each generated dataset is independent and is saved in two versions:

1. Sorter's version, containing:

- **Recorded signals**, which are the time-ordered values of the extracellular voltage, for each channel

2. Evaluator's version, containing the neuronal activity, consisting of:
   - **Spike times** which are the time stamps of all true spiking events in the analog signal on each channel (spike times for each spike are taken as the time of threshold crossing)
   - **Unit labels** corresponding to each spiking event which define the correct units, or clusters of spikes

The generation of this data takes place in three steps. In step 1, we generate the background signal, which serves as a base on which we superpose artificial neuronal activity. In the next step, we generate this neuronal activity. This step is broken into two parts. In the first part, we generate the waveforms and in the second part we generate the spike times and insert the waveforms at these spike times onto the background signal.

In summary, the generation of each dataset takes place in the following steps:

1. Generation of background signals
2. Generation of neuronal activity
   a) Creation of waveform templates
   b) Insertion of waveform templates

We now discuss each step in detail.

3.1.1. Generation of Background Signals

The background signal is that part of an recorded signal that does not contain relevant spiking events. Here we describe how we generate such signals.

Although we create a distinction between neuronal activity and the background signal, the background signal is not devoid of all spiking activity. Recorded signals from each electrode reflect the complex firing activity of neurons in an extended volume of brain tissue around the electrode tip, such that that a large number of neurons are contributing to the signal. Due to electrical properties of the brain tissue, the extracellular potentials of only the nearest neurons are observed as significant deviations from the population average. Thus, we can only isolate the spiking activity of these nearby neurons and we classify that as neuronal activity.

Thus, due to the complex nature of the underlying spiking activity, we cannot use a simple noise model to generate realistic realizations of the background signal. Instead, we obtain different realizations of it by removing the neuronal activity from the recorded signal obtained on electrodes from real datasets.

We use a dataset obtained from experiments that made use of the Utah array, thus it contains recorded signals from 96 electrodes. The dataset has been manually spike sorted by an expert, which means we have access to the underlying neuronal activity in the recorded signals, i.e., the times and unit labels of each generated spiking event in the

recorded signal from each electrode. In order to remove this inherent neuronal activity from the recorded signal, we tried two approaches.

In the first approach, we deleted the recorded signal in a window around each spike time. We did this by iterating over every pair of consecutive spikes detected for each electrode and concatenating the recorded signal between them. The result was a concatenated signal containing all sections of the recorded signal that had no spiking events. This would be our potential background signal. However, this approach introduced large changes in the voltage values and also returned shorter signals than the original (Figure 3.1a). Since the number of spike sorted events changes from electrode to electrode, the signals extracted from different electrodes of a dataset will have different sizes. This would introduce an unnecessary complexity when constructing a surrogate dataset from signals of unequal length. To remedy this, we used a different approach.

In the second approach, we made use of the fact that every spike sorted unit has a characteristic waveform shape that becomes apparent when you calculate the mean over all the individual waveforms of the unit. The shape of individual waveforms fluctuates around this mean. We calculated the mean waveform for every unit found at the end of manual spike sorting on each electrode. Then, we subtracted this mean waveform from a window of the same length around every spike time belonging to the unit in the recorded signal. This method removes the underlying spike shape of every waveform in the signal, but preserves the inherent noise (variation around the mean) for each waveform, yielding a signal containing no threshold crossing events, and without introducing any sudden changes Figure 3.2. Thus, we obtain our background signals Figure 3.1b.

Although, subtracting the mean waveform leads to a smoother signal, it could introduce artefacts into the signal when the original spike sorting is not completely correct. An example of such an artefact is show in the cyan insets of Figure 3.1.

This step takes only one parameter - the length of the waveform, which is also the size of the window that is to be subtracted around each spike, measured in units of the sampling period of the signal. Additionally, this step relies on the availability of the analog signals and the corresponding spike sorted data. We obtain as many realizations of the background signal as there are electrodes in the dataset.

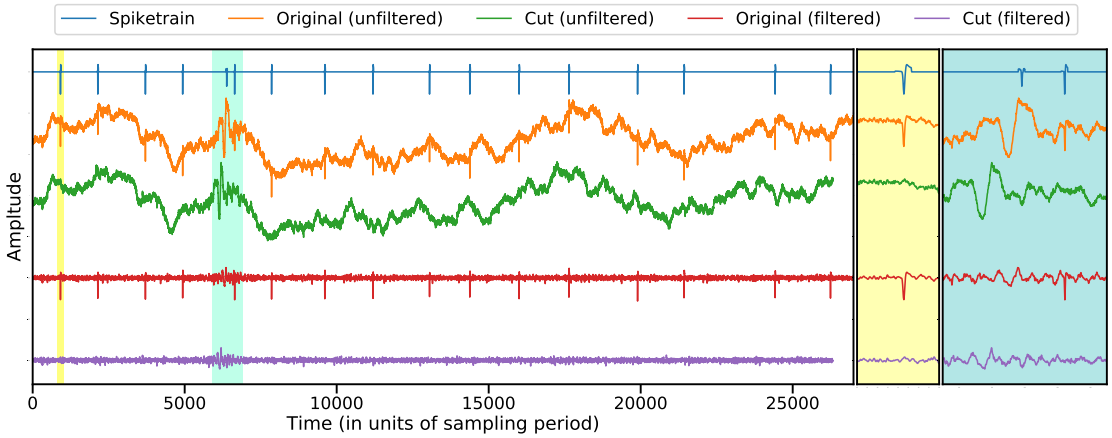
3.1.2. Generation of Neuronal Activity

The neuronal activity contains the spiking information relevant to the spike sorters. Here we describe how we generated it.

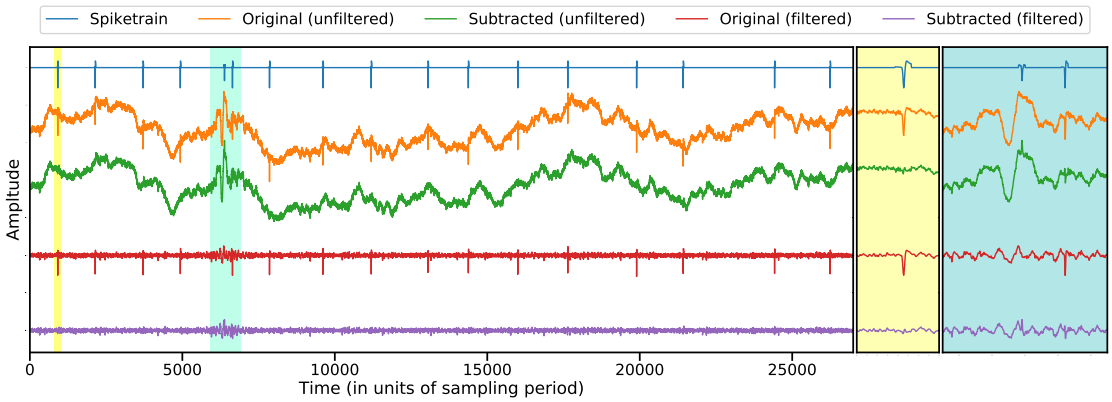
As mentioned in section 3.1.1, the mean waveform of a unit represents the underlying shape of the action potential generated by the putative neuron the unit represents. The individual waveforms in the unit vary around this mean due to:

1. The variation in the spike shape caused by the inherent stochasticity in the shape of the action potential generated by the neuron, combined with small shifts in the position of the electrode tip relative to the neuron (σ_{inherent})

2. The stochastic nature of the background signal due to the underlying complex spiking activity ($\sigma_{\text{background}}$)



(a) Removal of waveforms.



(b) Subtraction of mean waveforms.

Figure 3.1.: Extraction of background signals from the recorded signal by (a) removal of spike sorted waveforms and (b) subtraction of mean waveforms. The spike trains (blue), recorded signal (orange), recorded signal after removal of waveforms (green), filtered signal (red) and filtered signal after removal of waveforms (purple), from top to bottom, are shown in each figure for approx. 1 second of recording. The yellow and cyan insets show zoomed views of a small portion of the signals and are comparable across the figures. In (a), due to the removal of many waveforms from the signal, the green signal is shorter than the orange one, and similarly, the purple signal is shorter than the red one. The removal of the waveform in the yellow inset causes a noticeable change in the signal (orange to green). By comparison, the same region appears smoother in (b) than in (a). Occasionally, the subtraction of the mean waveform in a badly sorted unit introduces artefacts into the signal (cyan insets).

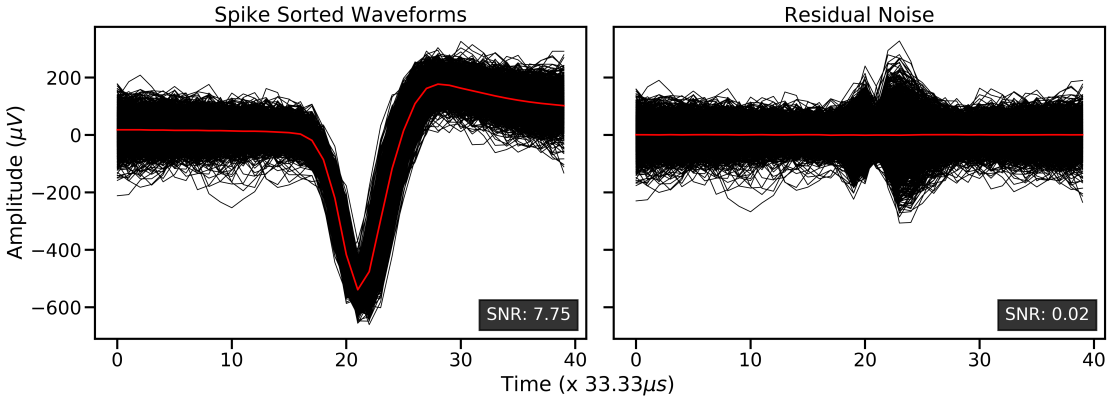


Figure 3.2.: Subtraction of mean waveform of unit to obtain residual noise. The left panel shows all waveforms assigned to a single unit during spike sorting. The red line is the mean waveform. The right panel shows the same waveforms after subtracting the mean waveform from each. The signal-to-noise ratio (SNR) (Hatsopoulos et al., 2007) values in the bottom right corner of each panel show that the residual noise is not significantly larger than the background noise and such signals will likely not be detected by any spike sorting algorithm.

In more mathematical terms, given the matrix W_{ij}^a of all waveforms belonging to a certain unit a, where the W_{ij}^ath element is the voltage measured at the jth time point of the ith waveform, we calculate the mean waveform,

$$\mu_j^a = \frac{\sum_i^M W_{ij}}{M}, \tag{3.1}$$

and its standard deviation,

$$\sigma_j^a = \sqrt{\frac{\sum_i^M \left[\left(W_{ij}^a - \mu_j^a\right)^2\right]}{M}}, \tag{3.2}$$

where M is the number of waveforms in the unit, $\mu^a = [\mu_1^a, \mu_2^a, ..., \mu_T^a]$ is the mean waveform of the unit and $\sigma^a = [\sigma_1^a, \sigma_2^a, ..., \sigma_T^a]$ is the standard deviation of the waveforms around the mean. T is the number of time points in each waveform. We can then say

$$\sigma^a = \sigma^a_{\text{inherent}} + \sigma^a_{\text{background}}, \tag{3.3}$$

In this work, we set $\sigma_{\text{inherent}} = 0$. We assume that the inherent stochasticity from action potential generation causes deviations that are negligible when compared to $\sigma_{\text{background}}$ because the variations in the amplitude at the level of individual waveforms is much smaller than the variations in the amplitude of the background activity. Also, to our

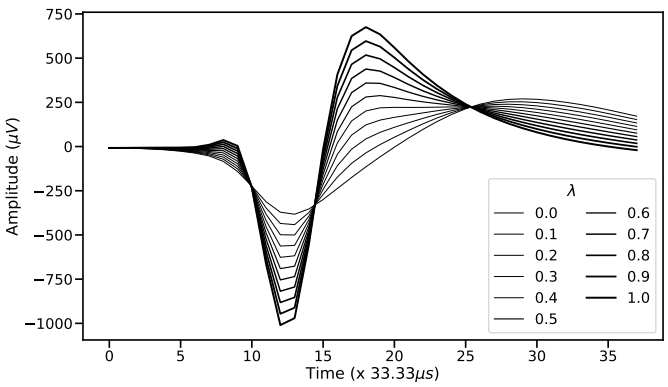


Figure 3.3: Variation of waveform shape with λ. The waveform templates used in this thesis are combined according to equation (3.4) to obtain different waveform shapes, shown by lines of different thickness. The shapes of the source templates correspond to $\lambda = 0.0$ and $\lambda = 1.0$.

knowledge, there are no measurements of inherent variablity to base our model on.

Therefore, to generate new units in a channel, we must only create a mean waveform for each unit and superpose it onto the background activity at specific time points. We achieve this in two steps:

3.1.2.1. Generation of Waveform Templates

The mean waveform of a unit represents the characteristic shape of the extracellular action potential of a neuron, measured at the tip of the implanted electrode. The action potential is a result of complex biophysical processes that take place inside the neuron. Although there are existing models which we could use to simulate action potentials similar to the ones we observe in our dataset, they require model parameters which are unknown to us. Without specific model parameters, the simulated action potentials may differ significantly from the ones we observe in our data. This is something we would avoid, since we are interested in the performance of spike sorting algorithms in the specific context of our datasets.

Instead, we take an approach based on surrogate data. We create new mean waveforms, or *waveform templates*, by taking a linear superposition of two *source templates*. We use mean waveforms from existing spike sorted units for source templates and employ one parameter λ that controls the shape of the resulting waveform. More specifically, given two source templates W^a and W^b, we can construct the waveform template W^c as

$$W^c = \lambda W^a + (1 - \lambda) W^b, \tag{3.4}$$

$$\lambda \in \mathbb{R}; 0 \leq \lambda \leq 1.$$

Here, the templates W^a, W^b and W^c are vectors with the number of elements equal to the temporal width of the template at sampling resolution and λ is a positive scalar. By varying λ, we can obtain a range of different waveform shapes that transition from W^a to W^b Figure 3.3.

For a given artificial dataset, the same source templates are used to generate waveform templates on all channels. The performance of the sorter is analysed independently on

data from each channel. Since the λ values represent the different shapes a waveform can take, we test the sorter on its ability to differentiate units that differ by a certain $\Delta\lambda$. The difficulty of the sorting task would decrease with an increasing $\Delta\lambda$.

Once we create these templates for each unit on the channel, they are ready to be superposed with the background activity.

3.1.2.2. Insertion of Waveform Templates

The final step in the generation of artificial data is the insertion of the waveform templates we generated in the previous step into the background signal. To this end, we first need to determine where we should insert the templates. In other words, we need to generate spike trains - time-ordered series of spike times - for each unit we wish to insert.

Spike trains are generated by randomly sampling time points within a certain time range based on an underlying probability distribution. Various methods have been developed which generate spike trains with different properties. However, the spike sorting algorithms considered here are agnostic of the underlying spiking dynamics of the neurons. Therefore, we employ the homogenous Poisson process (Grün and Rotter, 2010) to generate spike trains. This is a simple model that takes only one parameter - the desired mean firing rate r. The method begins by first constructing an Inter-Spike Interval Histogram (ISIH) that follows a Poisson distribution. The spikes times of the spike train are generated by drawing the time of each subsequent spike from a homogenous Poisson-distributed ISIH. The homogenous Poisson distribution can be written as

$$\rho(\tau) = re^{-r\tau}, \tag{3.5}$$

where $\rho(\tau)$ is the probability density function for detecting the next spike in a time interval of τ, given an average firing rate of r. The distribution is called homogenous because of the constant underlying mean firing rate. The probability of observing a spike increases exponentially with time and the rate r determines how steeply the curve rises. Before we proceed with the insertion of the waveform templates, we must normalize the amplitude of the waveform template with the amplitude of the *filtered background signal*. The filtered background signal is obtained by band-pass filtering the background signal between 250 Hertz and 5000 Hertz, and as a result of filtering, it has a decreased voltage amplitude compared to the original signal. We perform this normalization for three reasons:

1. the background signal and the waveform template are extracted from different sources, and it is highly improbable that they have a comparable voltage amplitude,

2. the spike sorting algorithms use the filtered form of the recorded signal to perform the spike sorting, therefore, the waveforms we insert must have an amplitude comparable to that of the filtered background signal,

3. this procedure gives us parametric control over the difficulty of the sorting task, since we can now specify how much larger or smaller each unit must be in relation to the filtered background signal.

To this end, we begin by filtering the chosen background signal using an acausal 2^{nd}order Butterworth bandpass filter. Let $f(t)$ be the original background signal and $F(t)$ be it's filtered counterpart. Let W_{original} be the waveform template for the unit that is to be inserted. Then, we define

$$\sigma_{\text{filt}} = std\left[F(t)\right], \tag{3.6}$$

as the standard deviation of the filtered signal around it's mean. Here $std(X)$ represents the standard deviation of X. Further,

$$w_{\text{extent}} = w_{\text{peak}} - w_{\text{trough}}, \tag{3.7}$$

where w_{peak} and w_{trough} are the values of the waveform template at its maximum and minimum, respectively. This is the extent of the waveform template which we will scale to a desired value. The desired extent, w^*_{extent}, is parametrized by an input parameter α ($\geq 0, \in \mathbb{R}$). Thus,

$$w^*_{\text{extent}} = 2 \cdot \alpha \cdot \sigma_{\text{filt}}. \tag{3.8}$$

Now we can calculate the multiplier we need to obtain the scaled waveform template. The multiplier m is given by

$$m = \frac{w^*_{\text{extent}}}{w_{\text{extent}}}, \tag{3.9}$$

which finally allows us to calculate the scaled waveform template, W_{scaled}

$$W_{\text{scaled}} = m \cdot W_{\text{original}}. \tag{3.10}$$

We now have all the information we need to create the surrogate recorded signal composed of the background signal and inserted spikes. At every spike time in our Poisson spike train, we superpose the waveform template with the (unfiltered) background signal. We repeat this for every unit we wish to insert into that channel. Since the waveform template is a vector and the spike times are scalars, we must choose a standard time point in each vector to be the true time of spiking. We take this as the mid-point of the vector and define it as

$$t_s = \begin{cases} \frac{T}{2} + 1 & \text{if T is odd} \\ \frac{T}{2} & \text{if T is even} \end{cases}, \tag{3.11}$$

where t_s is the spike position and T is the length of the template vector.

3.1.2.3. Overlapping Spikes

During spike sorting, one often comes across channels on which two units have a strong tendency to fire together. Such channels are difficult to sort, even for human sorters, because the waveforms of one unit are influenced by the waveforms of another unit. It then becomes a challenge to distinguish the two units by looking at just their waveforms or the corresponding representation in PCA space. An example of such a channel is shown in Figure 3.4. We want to test the ability of the spike sorting algorithms to correctly sort such channels.

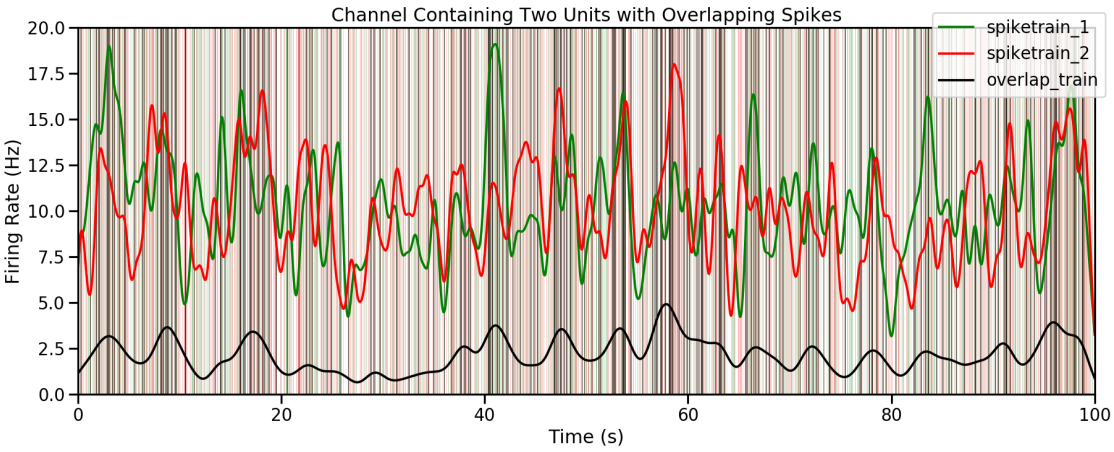


Figure 3.4.: Overlapping units on a single channel. The instantaneous firing rate profiles of two surrogate units are shown in green (spiketrain_1) and red (spiketrain_2), along with the instantaneous firing rate profile of the overlapping spike times in black (overlap_train). The green, red and black vertical lines correspond to individual spike times of the respectively coloured rate profiles. Both units here have an average firing rate of 10Hz with an overlap fraction of 0.2, which yields an average firing rate of 2Hz for the overlapping spikes.

To this end, we create channels that contain two overlapping units. The overlap between the units is characterized using two parameters - the *overlap fraction*, and the *jitter*. The overlap fraction determines what fraction of spikes from the first unit overlap with spikes from the second unit. This parameter directly determines the extent of the overlap between the two units. The jitter determines the shift of the spikes of one unit with respect to those of the other and thus, characterizes the temporal extent of overlaps. For every overlapping spike t_s in the first unit, the time of the corresponding spike in the second unit will be drawn randomly from $[t_s - j, t_s + j]$, where j is the jitter. The jitter is measured in units of the sampling period of the data.

Let r be the firing rate of both units that are to be generated. Let ϕ be the overlap fraction. The aim is to generate two spiketrains, S_1 and S_2, corresponding to the first and second unit, respectively. To generate S_1 and S_2, we do the following:

1. Using the homogenous Poisson process, we create spiketrains S_A, S_B and S_{overlap}, with firing rates $(1-\phi)\cdot r$, $(1-\phi)\cdot r$ and $\phi \cdot r$, respectively

2. We remove any existing overlaps between S_A and S_B. We do this by shifting each overlapping spike from the second spiketrain in a window of $[t_s - 2\cdot j, t_s + 2\cdot j]$, where t_s is the corresponding overlapping spike in the first spike train. A pair of spikes is deemed to be overlapping when they lie within $\pm j$ of each other. We do this step iteratively, until no overlaps remain.

3. We further remove any existing overlaps between $S_A \oplus S_B$ and $S_{overlap}$. The $\oplus$ represents a merge of the two spiketrains.

4. We create S'_{overlap} by jittering every spike in S_{overlap} by a random value in $[-j, j]$.
5. Finally, we create $S_1 = S_A \oplus S_{\text{overlap}}$ and $S_2 = S_B \oplus S'_{\text{overlap}}$.

3.1.3. Parametrization

To summarize the generation of independent artificial datasets, we list here the parameters used at every step. These parameters have direct equivalents in the

1. Generation of background signal
   a) *waveform_size* - an integer for the length of the waveform that is subtracted around each spike time, measured in units of the sampling period of the dataset
2. Generation of neuronal activity
   a) Generation of waveform templates
      i. *source_templates* - two vectors of length equal to the desired width of the inserted waveforms, which are linearly combined to create waveform templates for every unit
      ii. *linear_multiplier* - a positive real number (λ) which is used as a linear multiplier when combining the source templates to create waveform templates; the difference in λ values between units determines how easy or diffiicult they are to distinguish; takes values between 0 and 1
   b) Insertion of waveform templates
      i. *scaling_factor* - a positive real number (α) which determines the relative amplitude of the waveform template of each unit with respect to the background signal where it's being inserted; the smaller this value, the more difficult this unit is to sort
      ii. *mean_firing_rate* - a positive real number (r) representing the mean firing rate of each unit inserted on any channel of a generated dataset
      iii. Overlapping spikes:
         A. *overlap_fraction* - a positive real number (ϕ) which determines the extent of overlap between both units on the channel; takes values between 0 and 1
         B. *jitter* - a positive integer (j) that determines the time window within which overlapping spikes are generated around each other; measured in units of the sampling period

4. Spike Sorting Pipelines

In the introduction to this thesis, we described the origin and structure of the data that we are analysing. We followed that with a description of the automatic spike sorting algorithms that we evaluate in this thesis and how these algorithms are parametrized. The previous chapter focussed on artificial data generation, wherein we described how we generate surrogate data which mimics real-world data. This chapter focuses on the analysis framework we developed on the basis of these components.

Much of the analysis in this thesis requires the repeated execution of specific steps, for instance in the generation of surrogate data and in the automatic spike sorting of datasets. Wherever we encountered such repeated analysis steps, we organized them into a *pipeline*. A pipeline contains all the steps required by a certain analysis arranged in a sequential and modular fashion. Pipelines allow for reproducible and efficient execution of a certain analysis workflow. For example, the pipelines developed in the following sections can be seamlessly integrated into the overall data processing workflow for datasets similar to the one used in this thesis. This chapter describes the pipelines we designed along with details of their implementation. It lays the practical groundwork upon which the work for this thesis was done.

4.1. Basic Structure

Each pipeline described in subsequent sections is intended for a specific analysis step and thus, fulfills a specific purpose. However, the underlying structure of the pipelines can still be generalized. Figure 4.1 depicts this structure as a flowchart. The dataset generation/acquisition and automatic spike sorting are common to all pipelines, whereas the ground truth generation and comparison of results are required only in certain pipelines.

More explicitly, we developed the following three pipelines.

1. Parameter Scan Pipeline
2. Surrogate Dataset Pipeline
3. Spike Sorting Pipeline

We describe each of these separately in the following subsections. Following that discussion, we clarify our choices of file formats and comment on the design of the configuration files used in the pipelines.

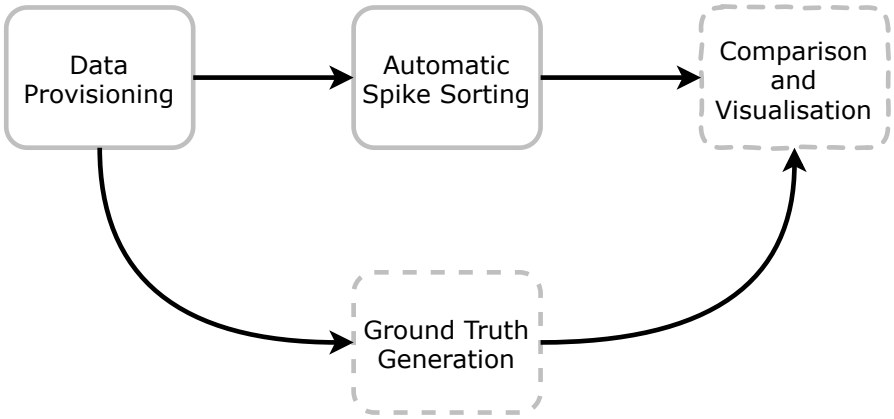


Figure 4.1.: General structure of the pipeline. The flowchart outlines the underlying structure of all presented pipelines. Each box represents a different analysis step. Boxes with dashed borders indicate steps which are contained in only some pipelines. **Data Provisioning:** data acquisition or surrogate data generation. **Automatic Spike Sorting:** automatic spike sorting of data using selected algorithms. **Ground Truth Generation:** manual spike sorting or surrogate data generation. **Comparison and Visualisation:** comparison of results from automatic spike sorting to ground truth. Dotted lines represent steps or paths specific to certain pipelines.

4.1.1. Parameter Scan Pipeline

The automatic spike sorting of data takes place in a sequence of steps, where each step accepts certain parameters that influence the spike sorting results. In order to determine which parameters yield optimal results for a given sorting algorithm, we analyze the effect of each individual parameter on the final result. To this end, we perform a *parameter scan* over the space of selected parameters for each spike sorting algorithm. A parameter scan involves traversing the parameter space spanned by all the parameters of the algorithm to locate specific parameter sets which perform optimally. Since the number of parameter sets to analyze is typically large, we created a pipeline which could automate this analysis.

The primary aim of this pipeline is to run the spike sorting algorithm for all parameter sets and compare the result from each run with manual spike sorting performed by experts. Figure 4.2 gives a diagrammatic representation of the pipeline. For a certain dataset X, the pipeline proceeds as follows:

1. Data acquired from experiments is located and accessed. The data is stored in *X.nev* and *X.ns6* files. The *X.nev* file also contains metadata relevant to the experiment. Additional metadata stored in other files may be used.

2. The datasets are manually spike sorted by a human expert. The manual spike sorting is performed using the Plexon Offline Spike Sorter (Offline SorterTM, Plexon Inc., Dallas, TX, USA) and yields a file *X-01.nev*, where the suffix *-01* is an arbitrary choice. The sorting results stored in *X-01.nev* are converted to *X-manual.npy*.

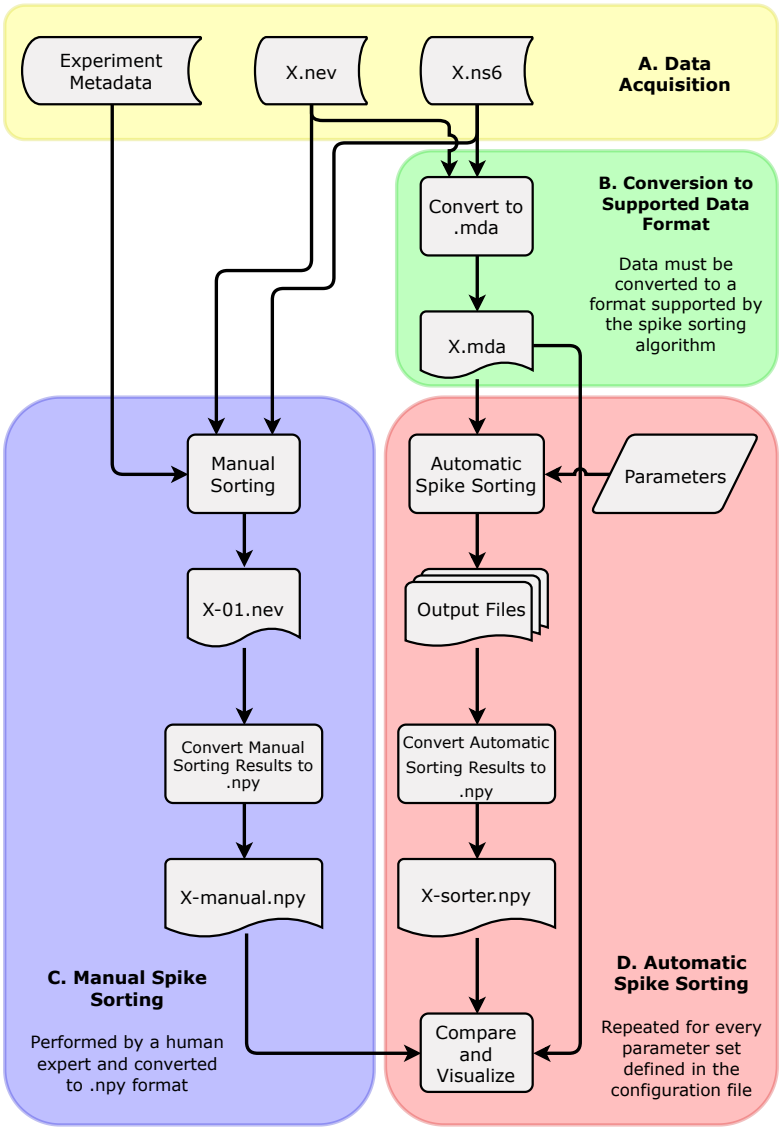


Figure 4.2.: Flowchart showing the parameter scan pipeline. The four coloured boxes contain four distinct parts of the pipeline. **A.** The experimental data is collected and stored into the *X.nev* and *X.ns6* file formats. Metadata related to the experiments is also stored separately. **B.** The *X.nev* and *X.ns6* files are used to create *X.mda*. This file contains the recorded signals from all recording electrodes. **C.** The acquired data is spike sorted manually by a human expert and the results are stored in the *X-01.nev* file. The contents of this file are then stored in *X-manual.npy*. **D.** The file *X.mda* is provided to the spike sorting algorithm as input and the spike sorting is performed once for every parameter set defined in the configuration file (shown here as *Parameters*). The results of each spike sorting are stored in the *X-sorter.npy* format. Finally, a comparison between the results in *X-sorter.npy* and *X-manual.npy* is made.

3. The *X.ns6* file, containing the recorded signals, is converted to the *X.mda* file.

4. For every parameter set defined in the configuration file:
    a) The *X.mda* file is supplied to the spike sorting algorithm, which sorts the dataset using the defined parameter set and returns corresponding output files
    b) The results in these output files are consolidated and stored in *X-sorter.npy*
    c) The results stored in *X-sorter.npy* are compared to the results stored in *X-manual.npy*, and the performance of the sorter using that parameter set is evaluated

5. Based on the comparisons, we obtain a certain set of optimal parameters for each sorter.

4.1.2. Surrogate Dataset Pipeline

Once we obtain the optimal parameter sets for each algorithm, we run the algorithms on artificially generated surrogate datasets to evaluate their performance under different theoretical scenarios. Surrogate data generation takes place in two steps - 1. background signal extraction 2. data generation Background signal extraction is performed once to obtain a library of background signals from manually spike sorted data. These background signals are passed on to the data generation step, which yields surrogate datasets based on the supplied parameters. Multiple realizations of each dataset are generated so that we can derive statistics over the performance of each algorithm. Finally, each dataset is then spike sorted using the automatic algorithms and the results are compared to the artificially generated ground truth. We integrate all these steps into a pipeline, shown in Figure 4.3. The pipeline contains the following steps:

1. Dataset acquired from experiments is located and accessed. These files are stored in *X.nev* and *X.ns6* files.

2. A human expert manually performs spike sorting on this dataset to yield *X-manual.nev*.

3. Background signals are extracted from the recorded signals by removing the neuronal activity stored in *X-manual.nev* (see 3.1.1). These signals are stored in a *.npy* file.

4. Surrogate datasets are generated using the parameters provided in the configuration file. Multiple realizations are created using each parameter set, and each dataset is stored in two files - a) *.npy* file containing ground truth spiking information b) *.mda* file containing surrogate recorded signals.

5. For each surrogate dataset that is generated
    a) the recorded signal (stored in the *.mda* file) is provided as input to the spike sorting algorithm, which yields corresponding output files

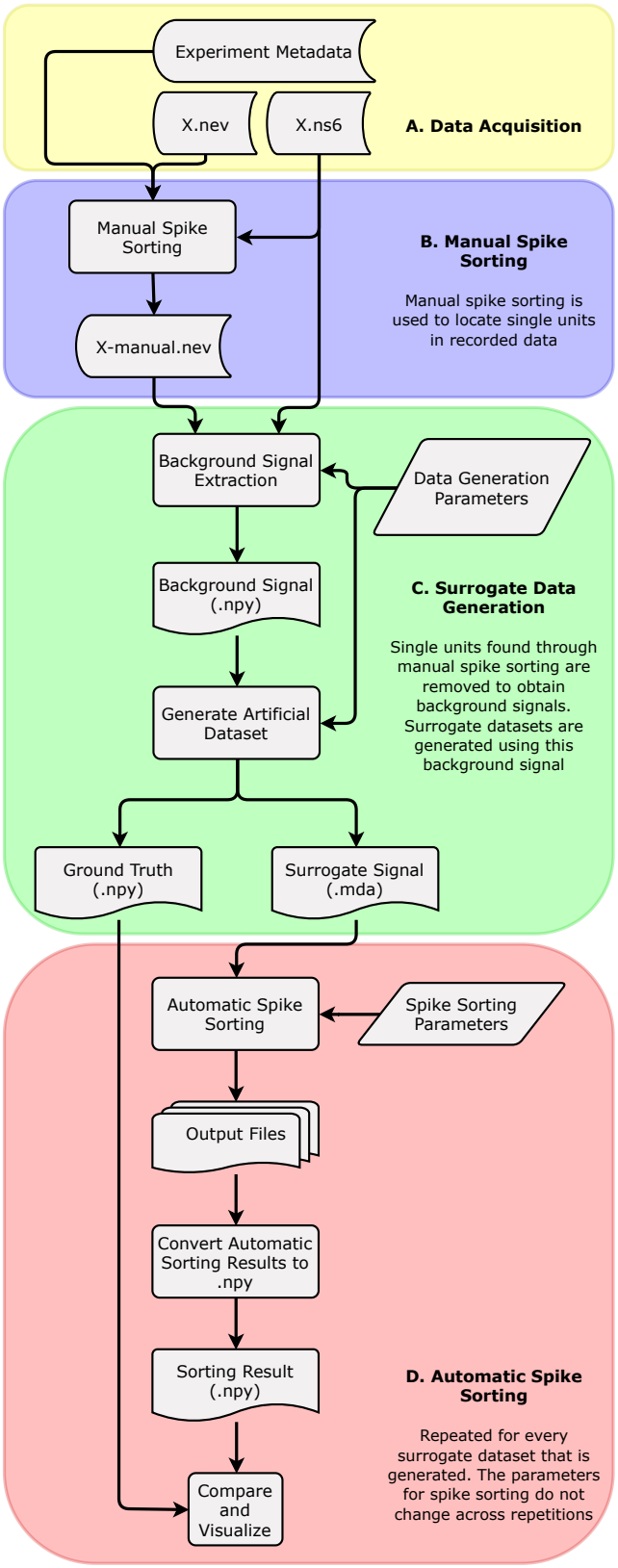


Figure 4.3: Flowchart showing the surrogate dataset pipeline. The four coloured boxes contain the four distinct parts of the pipeline. **A.** The experimental data is collected and stored into the *X.nev* and *X.ns6* files. **B.** The dataset is manually spike sorted by a human expert so as to obtain a record of all single units. The results are stored in *X-manual.nev*. **C.** Background signals containing no spiking activity are obtained by removing the single units recorded in the *X-sorted.nev* from the *X.ns6* file. The background signals are stored in a *.npy* file. These, along with the data generation parameters, are used to generate surrogate datasets. Each surrogate dataset has a *.npy* file containing ground truth spike times and a *.mda* file containing the generated surrogate signals. This step is repeated multiple times to yield many realizations from each parameter set. **D.** Each surrogate signal is automatically spike sorted and the results are compared to the generated ground truth.

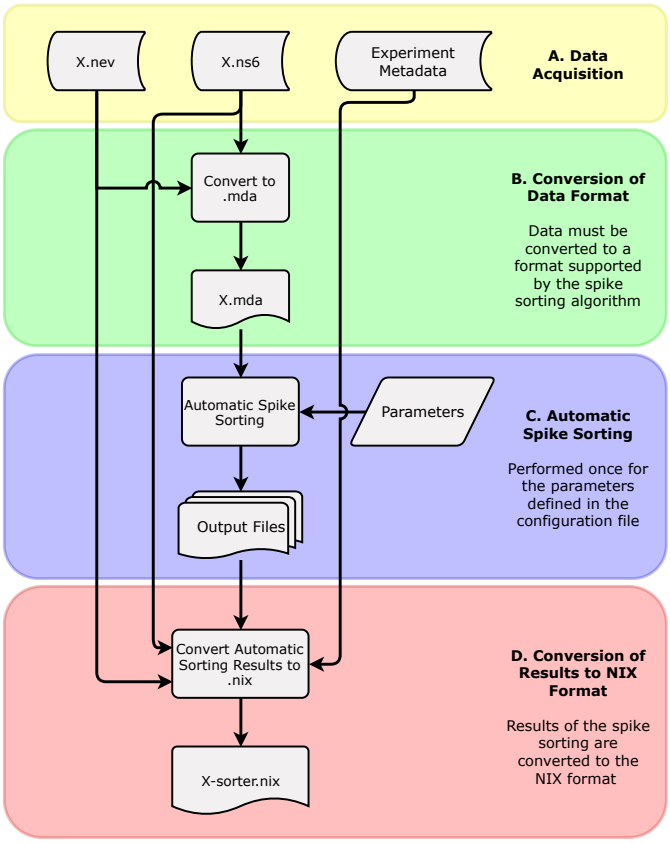


Figure 4.4: Flowchart showing the automatic spike sorting pipeline. The four coloured boxes contain the four major parts of the pipeline. **A.** The experimental data is collected and stored into the *X.nev* and *X.ns6* files. **B.** The *X.nev* and *X.ns6* files are used to create *X.mda*. This file contains the recorded signals from all electrodes. **C.** The acquired data is automatically spike sorted to obtain the corresponding output files. **D.** The spike sorting result from the output files is used in conjuction with the recorded data and metadata to create the final *X.nix* file, which contains all the spike times, corresponding waveforms and unit labels for all electrodes.

b) the results in these output files are consolidated into a single *.npy* file

c) the results in the *.npy* file are compared to the surrogate ground truth stored in its *.npy* file to evaluate sorter's performance on that dataset

Evaluating the performance of each spike sorting algorithm on surrogate datasets as ground truth gives us a more objective measure of how well the algorithms are able to retrive underlying neuronal activity.

4.1.3. Automatic Spike Sorting Pipeline

Now that we have obtained the optimal parameter set for each sorter and tested it against surrogate datasets, we design a pipeline that allows us to spike sort multiple real-world datasets simultaneously and yield results in a standardized data format. This ensures that the analysis steps we have used here can be seamlessly integrated into larger data workflows. Figure 4.4 is a diagrammatic representation of this pipeline. For each dataset X to be sorted by a given spike sorting algorithm, it consists of these steps:

1. Dataset acquired from experiment is located and accessed. These files are stored in *X.nev* and *X.ns6* files. Metadata corresponding to the experiment is stored in the *X.nev* file.

2. The *X.ns6* file, containing the recorded signals, is converted to an *X.mda* file.

3. The *X.mda* file is supplied as input to the spike sorting algorithm, along with the set of optimal parameters, to yield a set of output files

4. The results in these output files is consolidated and stored into an *X-sorter.nix* file. The *X-sorter.nix* file also contains the metadata stored in *X.nev* and a copy of the recorded signals from *X.ns6*.

Since this pipeline accepts as input the files generated during an experiment and yields as output a single file containing spike sorted data, it can be easily integrated into an existing data analysis workflow.

4.2. File Formats

At various steps in each pipeline, we used specific file formats to store datasets and intermediate files. Since the choices were common across pipelines, we briefly discuss here the motivation behind using those file formats.

- The datasets obtained from experiments are typically stored in proprietary file formats created by the manufacturers of the recording equipment. In our case, we use systems created by Blackrock Microsystems, and the datasets we obtain are stored in *.nev* and *.ns6* files. The spike sorting softwares we study do not support these file formats, so we must convert our data to a suitable file format before we can use the algorithms. We decided to use the *.mda* file format to store our recorded signals since it is a binary file format supported by both spike sorters.

- The results from both spike sorting algorithms are structured differently and stored in different file formats. In the parameter scan pipeline and surrogate dataset pipelines, we convert these results to a common data structure and store them in the *.npy* file format. This file format was chosen for convenience and cross-platform compatibility, since we use these files in later steps to draw comparisons between the results of the algorithms and ground truth.

- In the automatic spike sorting pipeline, we store the final results of spike sorting in *.nix* files. The NIX (Neuroscience Information Exchange) (INCF Data Sharing Program, 2016) format is an open data format created with the goal of having a single file container capable of storing experimental data and metadata together.

4.3. Configuration Files

Each pipeline we develop requires the specification of certain configuration parameters which include the spike sorting or data generation parameters. These configuration parameters are provided in the form of text files and contain necessary information for the execution of the pipeline, such as the location of the stored datasets, desired location

for storage of results, number of processing cores to utilize when running the pipeline, etc. While of practical importance, these details are not relevant to this thesis and are documented in the accompanying code.

4.4. Snakemake

All the pipelines described in this chapter were created using the Snakemake workflow management system (Köster and Rahmann, 2012). According to the its documentation website,

> The Snakemake workflow management system is a tool to create reproducible and scalable data analyses. Workflows are described via a human readable, Python based language. They can be seamlessly scaled to server, cluster, grid and cloud environments, without the need to modify the workflow definition. Finally, Snakemake workflows can entail a description of required software, which will be automatically deployed to any execution environment.

The pipelines we created here using snakemake demonstrate this scalability and flexibility. The same pipelines run seamlessly on a compute cluster with several compute nodes and cores, as well as on a single, local machine containing just a few cores. This inherent scalability allows these pipelines to be used on a variety of machines and makes them easily shareable.

5. Analysis Framework

In the previous chapter we introduced the spike sorting pipelines which we used to analyse the performance of the spike sorting algorithms being considered in this thesis. In this chapter, we describe how we analyze the performance - what metrics we use and how those metrics relate to performance.

Before we begin describing the analysis framework, it is important that we define what we mean by an algorithm's performance. Since we're comparing spike sorting algorithms, the performance of the algorithm could be defined as its ability to correctly identify and classify spiking events in a given dataset. This definition implies that there is a correct spike sorting for every dataset and, in our case, we consider this correct sorting to either be the result of manual spike sorting by experts or the artificially generated surrogate data. The performance of an algorithm is then determined by comparing the spike times of the spikes in the spike sorting result to those present in the ground truth. Also, we want to determine parameter sets which perform optimally for each spike sorting algorithm, which is why the analysis framework described here is designed to handle results of parameter scans.

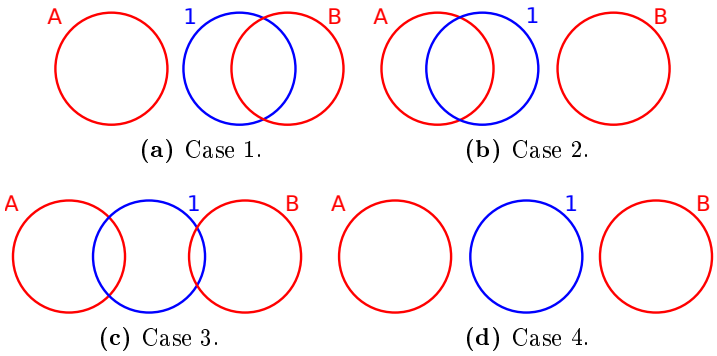
This chapter begins with an introduction to classification of events and the corresponding terminologies we use therein. This is followed by several sections dedicated to a description of how we derive metrics from these classifications.

5.1. Comparing Spike Sorting Results

In this section, we introduce the basic terminologies on which our analysis framework is based.

The datasets we analyze in this thesis are recorded using microelectrode arrays with a large inter-electrode spacing, implying that activity from a given neuron cannot be observed on more than one electrode. Thus, when determining how well an algorithm performs on a given dataset, we consider signals from each electrode independently.

Furthermore, spike sorting is a combination of a classification problem ("a certain detected event is a spike, or is not a spike") and a clustering problem ("a certain spike belongs to unit x"). This leads to a non-trivial comparison between the ground truth (GT) and the spike sorting result (SR), since we must first determine the best match between the units in the GT and those in the SR before we can evaluate the quality of the matches. As an illustration, consider an electrode with two ground truth units (GTUs) which was spike sorted by an automatic algorithm to obtain one spike sorted unit (SU) (Figure 5.1). The spike sorting algorithm has found one unit less than in the GT, and has thus performed sub-optimally. We further wish to evaluate the spike



(a) Case 1. **(b)** Case 2. **(c)** Case 3. **(d)** Case 4.

Figure 5.1.: Different case scenarios in the comparison of two ground truth units (GTUs) with a single spike sorted unit (SU). Four case scenarios are depicted showing four possible ways in which two GTUs (red circles – *A* and *B*) can overlap with a single SU (blue circle – *1*). The circles represent sets of spike times belonging to each unit and an overlap between two circles suggests that the two units contain common spike times. (a) The SU overlaps with GTU *B*, but not with GTU *A*. (b) The SU overlaps with GTU *A*, but not with GTU *B*. (c) The SU overlaps with both GTUs, *A* and *B*. (d) The SU overlaps with neither of the GTUs.

sorting performance on a spike-by-spike basis. However, we do not know which of the two GTUs is best represented by the single SU. In this simple construction, with the two GTUs being GTU_A and GTU_B and the SU being SU_1, we have four possible scenarios (compare figures Figure 5.1 (a) - (d), respectively)

1. Case 1: SU_1 has spikes in common with (or, overlaps with) only GTU_B
2. Case 2: SU_1 overlaps with only GTU_A
3. Case 3: SU_1 overlaps with both GTU_A and GTU_B
4. Case 4: SU_1 has no overlaps with either GTU_A or GTU_B

In cases 1 and 2, SU_1 can be trivially matched with GTU_A and GTU_B, respectively. Case 4 is also trivial because SU_1 does not have any events in common with the GT. However, in case 3, we must decide on a criterion which assigns the correct GTU to SU_1. It is a recurring case in the results we obtain from our spike sorting algorithms. Before we explain how to resolve this, we must first introduce some concepts.

5.1.1. Basic Definitions and Metrics

Consider the simple scenario depicted in Figure 5.2, with one GTU (red circle, GTU_A) and one SU (blue circle, SU_1) which overlap with each other. The bounding box represents the set of all events. We then categorize the spiking events as follows.

True Positives (TPs) Events common to both GTU_A and SU_1 (correct classifications).

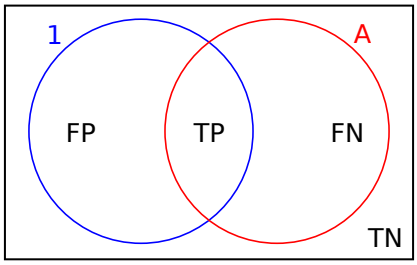


Figure 5.2.: Simple case scenario of an overlap between one GTU and one SU on a single electrode. The Venn diagram depicts the case where one GTU (red) is compared to one SU (blue) on a single electrode. The circles in the diagram represent the spike times of each unit, with the size of the circle corresponding to the relative number of events. In the case above, for reasons of simplicity, both circles have the same size, implying that the units have the same number of spikes. The labels within the circles represent TP: True Positives; FP: False Positives; FN: False Negatives; TN: True Negatives.

False Positives (FPs) Events sorted into SU_1, but are not present in GTU_A (false alarms).

False Negatives (FNs) Events present in GTU_A, but not in SU_1 (misses).

True Negatives (TNs) Events which are neither present in GTU_A nor in SU_1 (correct rejections).

Each event represented in Figure 5.2 can be placed in one of the above categories. By comparing the number of events in these categories, it is possible to characterize the goodness of the match between, in this case, GTU_A and SU_1.

Using these categories, we derive metrics which characterize the performance of the spike sorting algorithm (Powers, 2011). Each of these metrics is always computed for a pair of units (i.e. one GTU versus one SU), and yields information about the level of agreement between the two units.

1. Precision (P) is a measure of the proportion of spikes in SU_1 which were correctly classified. This measure must be **maximized** for a good spike sorting. It is given by the formula
$$P = \frac{TP}{TP + FP} \tag{5.1}$$

2. Recall (R) is a measure of the proportion of the spikes in GTU_A which were correctly classified into SU_1. This measure must be **maximized** for a good spike sorting. It is given by the formula
$$R = \frac{TP}{TP + FN} \tag{5.2}$$

3. Fallout (F) indicates what proportion of spikes were classified into SU_1 which should not have been classified into SU_1. This measure must be **minimized** for a

good spike sorting. It is given by the formula

$$F = \frac{FP}{FP + TN} \tag{5.3}$$

4. F1-Score (f_1) combines the information in precision and recall, and is high when both precision and recall are high. Thus, this measure must be **maximized** for a good spike sorting. It is given by the formula

$$f_1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{5.4}$$

$$= \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{5.5}$$

5.1.2. Detailed Scenario

In section 5.1.1, we introduced the classification categories for the comparison of a pair of units and listed measures which could be used to characterize the quality of spike sorting. However, the scenarios we considered in Figure 5.1 and Figure 5.2 were simplified and the measures we defined there cannot be used directly on units obtained from real data because of the following issues.

1. Spike times from the GT and the SR are not aligned. For every spiking event detected in the recorded signal, the time of threshold crossing is flagged as the spike time for that event in the GT. In contrast, for both spike sorting algorithms, the time of the minimum of the corresponding spike waveform is flagged as the spike time for the event. This leads to a discrepancy in the precise spike time of each event. Moreover, each spike sorting algorithm uses a different method to filter and whiten the recorded signals. Differences in filtering affect the time of the minimum of individual waveforms in the processed signal, causing a discrepancy in precise spike times even between the two automatic spike sorting algorithms. This discrepancy prohibits a spike-by-spike comparison of the spiking events in the GT and the SR.

2. The set of ground truth events and the set of spike sorted events are not identical. Although we compare GTUs and SUs found on the same electrode, it is not necessary that the GT and the SR contain comparable sets of spiking events. The spiking events detected during automatic spike sorting depend, in particular, on the detection threshold specified in the parameters (with the number of events decreasing as the threshold is increased). As a result, the SR may have more or less spiking events than the GT. In order to be able to perform a spike-by-spike evaluation of the two, we must account for this imbalance in the number of spiking events between both sets.

3. The number of units and the number of spikes per unit varies between the GT and the SR. In the examples earlier, we considered simple cases where the number of

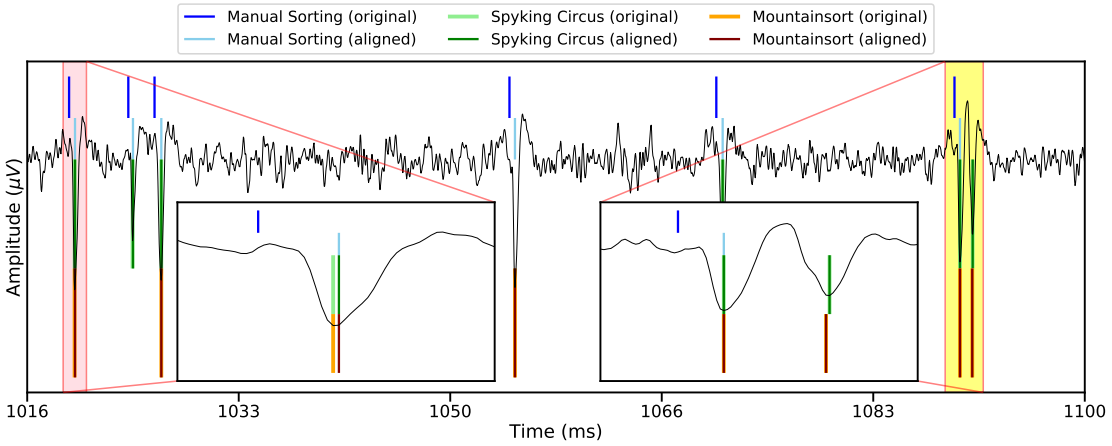


Figure 5.3.: Alignment of spike times. A small section of the filtered signal from one electrode is shown as the black trace. Spike times from different sources are shown as coloured vertical lines: original spike times from manual sorting (blue), original spike times from Spyking Circus (green), original spike times from Mountainsort (orange), aligned spike times from manual sorting (sky blue), aligned spike times from Spyking Circus (dark green) and aligned spike times from Mountainsort (maroon). The left inset (pink) shows a typical case where the original spike time from manual sorting is a few time stamps before the minimum of the dip in the filtered signal (corresponding to the extracellular spike waveform), while original spike times of the two sorters (green and orange) are the same. After alignment, the spike times from all three sources are aligned (sky blue, dark green, maroon). The right inset shows a case with a complex waveform shape (probably caused by overlapping waveforms) of which only one was detected in manual spike sorting and both were flagged as spikes in Mountainsort (Spyking Circus did not find this spike). Our alignment method finds the minimum correctly, matching only the first dip and not the second.

GTUs and SUs was small and the number of events in each unit was similar. In real data, however, the number of units in the GT can vary significantly from the number of units in the SR. The relative number of events in the units also vary greatly. This leads to the possibility that each SU overlaps with multiple GTUs - a case which must be specifically handled.

To be able to apply the measures of 5.1.1 to real data, we begin by aligning the spike times of the GT and the SR (issue 1).

5.1.2.1. Aligning Spike Times

An example of the misalignment of spike times is shown in Figure 5.3 for real data from one electrode. We remedy the misalignment of spikes in two steps:

1. by adjusting the spike times in the GT. On each electrode of a dataset, we look for the minimum of the filtered signal t'_{gt} around every GT spike time t_{gt} in a window

of $[t_{gt}, t_{gt} + w]$, where w is $30 \times 33.33\mu s$ ($33.33\mu s$ is the sampling period of the recorded signal); t'_{gt} is the new time stamp of that spiking event.

2. by adjusting the spike times in the SR. We align each spike time in the SR to its closest adjusted GT spike time. We accomplish this by iterating over every SR spike time t_{ss} in an increasing order and looking for a GT spike time in a window of $[t_{ss} - u, t_{ss} + u]$, where u is $15 \times 33.33\mu s$. If a GT spike at time t''_{gt} is found, we assign a new time stamp $t'_{ss} = t''_{gt}$ to that spiking event. Otherwise, $t'_{ss} = t_{ss}$.

This two step approach ensures that spike times corresponding to the same spiking event from different spike sorters are aligned, allowing for a direct spike-by-spike comparison.

We now address issues 2 and 3 outlined earlier, which are a result of differing detection thresholds and sorting quality. With different detection thresholds, the automatic algorithm finds more or less events than those seen by the human expert during manual spike sorting. Events found by the automatic spike sorter are either clustered into units or discarded entirely. During manual spike sorting, events which have not been assigned to any single unit are stored in a noise unit, hereafter referred to as the ground truth noise unit (GTN). For synthetic ground truth data, there are no discarded events. The categories defined in 5.1.1 are insufficient to describe the various possibilities that arise when evaluating a sorter's performance on real world data. We must construct a more nuanced picture than the one in Figure 5.2 to account for all possibilities.

5.1.2.2. Detailed Scenario

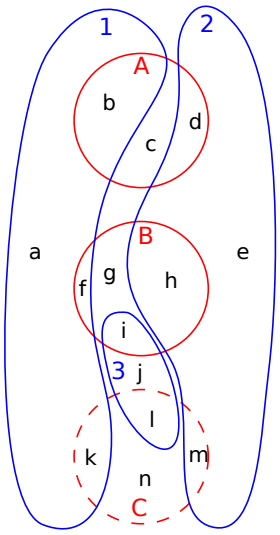
Consider the scenario depicted in Figure 5.4a of an electrode containing two GTUs (GTU_A and GTU_B) and one noise unit GTN (labeled C and shown with a dashed border in the figure). After automatic spike sorting, three SUs were found by a certain algorithm (SU_1, SU_2 and SU_3). Each SU can be matched with any of the of the GTUs, but not with the GTN (because it is, by definition, a set of noise events). For every possible GTU-SU pair, we introduce new categories for the events in those units. As an example, consider the pair GTU_A and SU_1. We define the following categories and subcategoreis of spiking events for this pair.

True Positives (TPs) Events common to both GTU_A and SU_1

False Positives (FPs) Events sorted into SU_1 which are not present in GTU_A; these are further categorized into

- **New (FP_{new})** Events sorted into SU_1 which are not present in any unit in the GT
- **Noise (FP_{noise})** Events sorted into SU_1 which are present in the GTN
- **Misclassified ($FP_{\text{misclassified}}$)** Events sorted into SU_1 which are present in GTU_2 (or any other GTU)

False Negatives (FNs) Events present in GTU_A which are not sorted into SU_1; these are further categorized into



(a) Venn Diagram Comparison.

$1 \iff A$	$2 \iff B$	Subcategory	Category
b	h	TP	TP
a	e	FP_{new}	FP
k	m	FP_{noise}	
f	d	$FP_{misclassified}$	
d	$f+i$	$FN_{classified}$	FN
c	g	FN_{missed}	
$j+e$	$j+a$	TN_{new}	TN
$l+m$	$k+l$	TN_{noise}	
$h+i$	b	TN_{sorted}	
g	c	TN_{missed}	
n	n	$TN_{missed\,noise}$	

(b) Table of Comparisons.

	GTU_A	GTU_B
SU_1	0.8	0.2
SU_2	0.2	0.85
SU_3	0.0	0.05

(c) Finding Optimal Unit Matches

Figure 5.4.: Illustrative example of comparing spike sorting results on a single electrode in order to define different events. **(a)** Example of overlapping sets of spikes corresponding to units on a certain electrode. Blue shapes depict the sets covered by events of SUs (1, 2 and 3) . Red circles depict the sets covered by events of the GTUs (A, B and C). The red circle with a dashed border represents the GTN. The regions of overlap between red and blue shapes denote events which are common to both. Each region is marked with a lowercase letter used in (b) to explain various subcategories of classifications. **(b)** A tabular description of the various regions in found in (a). The first column lists regions corresponding to the different categories and subcategories for the case where GTU_A and SU_1 were matched. The second column lists these regions for a match between GTU_B and SU_2. **(c)** A grid with SUs on the left and GTUs on the top, showing how optimal matches are determined for the scenario depicted in (a). Each cell shows an example f_{1_0} score for the corresponding two units (GTU and SU). The cells in green indicate successful matches, while those in red indicate failed matches.

Classified ($FN_{\text{classified}}$) Events in GTU_A which are sorted into neither SU_2 nor SU_3 (nor any other SU)

Missed (FN_{missed}) Events in GTU_A which were not present in any unit in the SR

True Negatives (TNs) Events not sorted into SU_1 which are also not present in GTU_A, but exist in other GTUs or SUs; these are further categorized into

New (TN_{new}) Events not present in any GTU, which are detected and sorted into another SU

Missed (TN_{missed}) Events from GTU_B (and any other GTU) which are not present in any SU

Noise (TN_{noise}) Events from GTN which are sorted into other SUs

Sorted (TN_{sorted}) Events sorted into other SUs which are also present in other GTUs

Missed Noise ($TN_{\text{missed noise}}$) Events from GTN which were not sorted into any SUs

To summarize these, the total number of events for each category is

$$\begin{aligned} FP &= FP_{\text{new}} + FP_{\text{noise}} + FP_{\text{misclassified}}, && (5.6)\\ FN &= FN_{\text{missed}} + FN_{\text{classified}}, && (5.7)\\ TN &= TN_{\text{new}} + TN_{\text{noise}} + TN_{\text{missed}} + TN_{\text{sorted}} + TN_{\text{missed noise}} && (5.8) \end{aligned}$$

The new definitions introduced here account for events which might be present in the GT and not present in the SR, or vice versa. We use these definitions to modify the basic metrics defined in 5.1.1 in order to be able to find matching pairs of units.

5.1.2.3. Matching Units

We use the measures defined above to match SUs to the GTUs that they resemble the most. For this, we define the modified forms of f_1, P and R given by

$$\begin{aligned} P_0 &= \frac{TP}{TP + FP_{\text{new}} + FP_{\text{misclassified}}} && (5.9)\\ R_0 &= \frac{TP}{TP + FN_{\text{missed}} + FN_{\text{classified}}} && (5.10)\\ f_{1_0} &= 2 \cdot \frac{P_0 \cdot R_0}{P_0 + R_0} && (5.11)\\ &= \frac{2 \cdot TP}{2 \cdot TP + FP_{\text{new}} + FP_{\text{misclassified}} + FN_{\text{missed}} + FN_{\text{classified}}} && (5.12) \end{aligned}$$

Here, P_0 describes what proportion of spikes in the SU are from a given GTU, while specifically excluding events classified as noise in the GTN. Noise events are excluded

because the number of events in the GTN shows a large variation across electrodes. This variation arises from the subjective manner of choosing a detection threshold for each electrode during manual spike sorting. A lower detection threshold increases the chances of obtaining noise events. In order to discount for this subjectivity in looking for best matched units, we ignore the GTN events.

R_0 measures the proportion of the spikes from the GTU are present in the SU. When both P_0 and R_0 approach their maximum values of 1, the SU can be said to closely resemble the GTU. The f_{1_0} score is the harmonic mean of P_0 and R_0 and, thus, combines the information contained in them. When both P_0 and R_0 approach 1, f_{1_0} takes values approaching 1. When either P_0 or R_0 or both are low, the f_1 score is also low. In order to ascertain which pairs of units on a certain electrode should be matched, we calculate the f_{1_0} score for every pair and look for the optimal matches by choosing those pairs which have the highest f_{1_0} scores (see Figure 5.4c for an example with dummy values – green cells indicate a match, red cells are not a match). Once all optimal SU-GTU pairs have been found, we proceed with evaluating the spike sorting quality.

5.2. Metrics for Evaluation

In 5.1.1, we introduced categories and sub-categories into which events could be segregated when comparing a SU-GTU pair. We then introduced metrics which characterized the performance. However, these metrics don't assess all aspects of the spike sorting. In this section, we introduce more metrics which work together to give a more comprehensive picture of the spike sorters' performance and help determine optimal parameter sets for each algorithm.

5.2.1. Classification Performance Score

The idea behind this metric is to consolidate information held in equations 5.1, 5.2 and 5.3 because these three metrics account for all sub-categories of events. Precision, Recall and Fallout are widely used in various fields of research to evaluate the perfomance of algorithms. Typically, these metrics are used for problems where the classification algorithm yields the probability with which each observation belongs to different classes. A variable probability threshold is used to set the minimum probability for each observation to belong to a certain class. The Fallout, Recall and Precision are computed for different probability thresholds in order to determine the best performing classifier. This is achieved with the help of Receiver Operating Characteristic (ROC) curves (Fawcett, 2006) and Precision-Recall (PR) curves (He and Garcia, 2009).

An ROC curve is a plot of Recall (ordinate) versus Fallout (abscissa) for different observations and decision thresholds. ROC curves provide a succinct overview of the performance of a certain classifier. Since Fallout and Recall take values between 0 and 1 – with a high Recall and low Fallout implying a good classification – the point $(0, 1)$ on the plot represents a perfect classification. For any classifier that performs better than chance, all points on the plot lie above the diagonal, the *line of no discrimination*. To construct an ROC curve, one can plot the Fallout and Recall values from a classification

result for different probability thresholds and connect these points to form a continuous curve.

A PR curve is a plot of Precision (ordinate) versus Recall (abscissa) for different observations and decision thresholds. In contrast to ROC curves, PR curves rely on Precision and Recall – both of which should be maximized in order to obtain a good classification. Since both take values between 0 and 1, the point representing perfect classification on a PR plot is $(1, 1)$. These curves are typically used in place of ROC curves in cases when there is a large number of TNs which skew the Fallout to small values. By not accounting for the TNs, PR curves provide a more accurate estimate of the classifier's performance when working with imbalanced datasets (Saito and Rehmsmeier, 2015). Similar to ROC curves, PR curves can be created by plotting Precision and Recall values from a classification result and connecting the points to form a continuous curve.

Spike sorting algorithms, however, are not probabilistic classifiers. Each spiking event is assigned a definite class (unit) by the algorithm. We then match each SU with a GTU on the same electrode (as described in 5.1.2.3). The quality of each match is computed using the P, R and F metrics. These metrics translate to single points in the ROC and PR spaces. Since we obtain the spike sorting results from a large range of parameter sets, each GTU in the dataset is represented by as many unique points in the ROC and PR spaces as the number of parameter sets in which it was matched. These points provide a graphical overview of the sorter's ability to retrieve events of each individual unit for different parameter values.

The dataset we analyze in this thesis has 175 GTUs and the algorithms are run over 9,216 (Spyking Circus) and 18,432 (Mountainsort) parameter sets. The performance of each GTU across all these parameter sets cannot be analyzed graphically. We instead develop scores which reduce the performance of each matched pair to a single value for both ROC and PR spaces, making use of the fact that the ROC and PR plots have points – $(0, 1)$ and $(1, 1)$, respectively – which represent optimal classification. The first score, S_{FR}, calculates the Euclidean distance of each point on the ROC plot to $(0, 1)$ and is defined as

$$\begin{aligned} S_{FR} =& \sqrt{(1-R)^2 + (0-F)^2} \\ =& \sqrt{(1-R)^2 + F^2}. \end{aligned} \tag{5.13}$$

This score characterizes the quality of each matched pair of units on the ROC plot. The second score, S_{RP}, which calculates the Euclidean distance of each point on the PR plot to $(1, 1)$ and is defined as

$$S_{RP} = \sqrt{(1-P)^2 + (1-R)^2}. \tag{5.14}$$

This score characterizes the quality of each matched pair on the PR plot. Both S_{FR} and S_{RP} are defined such that a value of 0 would represent a perfect match.

Now that the information held in the ROC and PR plots has been condensed into single value scores, we develop the classification performance score, S_{CP}, to further condense

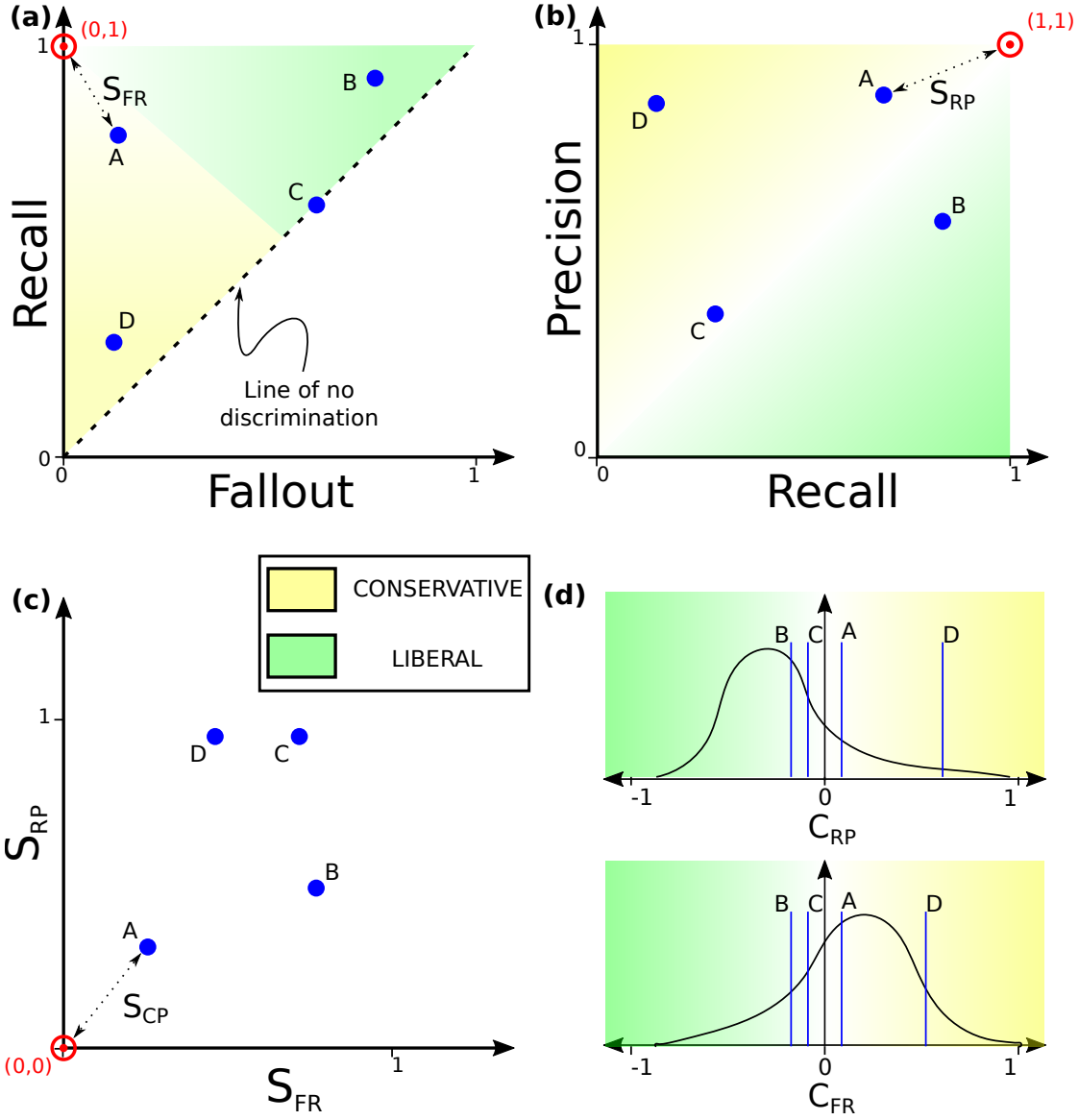


Figure 5.5.: ROC and PR curves. **(a)** ROC curve with four arbitrarily chosen points A, B, C and D. The regions shaded with a yellow gradient indicate a conservative sorting, with a darker shade for more conservative results. The regions shaded with a green gradient indicate a liberal sorting, with a darker shade for more liberal results. These colour gradients have the same interpretation for all panels of the figure. The dashed line from $(0, 0)$ to $(1, 1)$ is the line of no discrimination, and all points on the ROC plot lie above this line. S_{FR} is shown for point A. The point of optimal classification is shown in red in all relevant panels (here, shown at $(0, 1)$). **(b)** PR plot with points corresponding to those shown in (a). S_{RP} is shown for point A. **(c)** S_{FR} and S_{RP} values for the points A – D are plotted, and S_{CP} is shown for point A. **(d)** Sample distributions of C_{RP} (top subpanel) and C_{FR} (bottom subpanel), with positions of A, B, C and D denoted in blue vertical lines.

these values. It is defined as

$$\begin{aligned} S_{CP} =& \frac{\sqrt{(0 - S_{FR})^2 + (0 - S_{RP})^2}}{2} \\ =& \frac{\sqrt{S_{FR}^2 + S_{RP}^2}}{2}. \end{aligned} \tag{5.15}$$

which is the Euclidean distance of the point (S_{FR}, S_{RP}) from the origin at $(0, 0)$ on a plot of S_{FR} versus S_{RP}, normalized to values between 0 and 1. A value of 0 for S_{CP} indicates a perfect classification and a value of 1 indicates a completely incorrect classification (0 TP events and 0 TN events).

5.2.2. Conservativeness Scores

The scores defined in 5.2.1 give an intuitive idea of how far away a certain match of SU and GTU is from a perfect classification. However, they do not give an indication of where the corresponding points lie within the ROC and PR plots. The location of points on these plots provide insight into the liberal/conservative nature of the algorithm. A liberal algorithm would try to get as many TP events as possible, at the risk of a large number of FP events. A conservative algorithm would try to minimize the number of FP events, potentially leading to fewer TP events and more FN events. For the ROC plots, these regions are

1. low F and low R. Points in this region represent a conservative classification, since the algorithm sacrificed TPs (low R) to get a smaller number of FPs (low F).

2. high F and high R. Points in this region represent a liberal classification, since the algorithm allowed for a many FPs (high F) in order to get many TPs (high R).

3. low F and high R. Points in this region represent a balanced classification, with a small number of FPs and a high number of TPs.

On the PR plot, these regions are

1. low P and high R. Points in this region represent a liberal classification, since the algorithm allowed for a high number of FPs (low P) in order to get fewer FNs (high R).

2. high P and low R. Points in this region represent a conservative classification, since the algorithm preferred a high number of FNs (low R) in order to minimize the FPs (high P).

3. high P and high R. Points in this region represent a balanced classification, with a small number of FPs and FNs.

This information can be used to infer the general tendency of the algorithm to be liberal/conservative. However, the scores S_{FR} and S_{RP} defined earlier do not capture this

aspect because they use the Euclidean distance, which treats points in any direction equally. Thus, we introduce two more metrics which capture the differences in the conservativeness of the algorithms. The first conservativeness score is

$$C_{FR} = 1 - R - F \tag{5.16}$$

which would indicate a conservative sorting for positive values and a liberal sorting for negative values on the ROC plot. The second conservativeness score is

$$C_{RP} = P - R \tag{5.17}$$

which would indicate a conservative sorting for positive values and a liberal sorting for negative values on the PR plot.

Since these scores are defined for each matched pair, a distribution of these scores over all matched pairs in a dataset for a given parameter set would indicate the conservativeness of the parameter set.

5.2.3. Other Metrics

In addition to the scores developed in section 5.2.1 and section 5.2.2, we use the following metrics to account for more nuanced aspects of the spike sorting quality

5.2.3.1. Noise Fraction

Given a matched SU-GTU pair on a certain electrode, we determine what fraction of events in the SU belong to the GTN of that electrode. The noise fraction, M_{noise}, is defined as

$$M_{\text{noise}} = \frac{FP_{\text{noise}}}{TP + FP} \tag{5.18}$$

5.2.3.2. New Fraction

Given a matched SU-GTU pair on a certain electrode, we determine what fraction of events in the SU are not present at all in the GT. The new fraction, M_{new}, is defined as

$$M_{\text{new}} = \frac{FP_{\text{new}}}{TP + FP} \tag{5.19}$$

5.2.3.3. Units Ratio

On a certain electrode, the SR may contain more or less units than the GT. If an algorithm retrieves all GTUs on an electrode but also yields other SUs, it would be an undesirable trait. Likewise, if the algorithm retrieves a few GTUs completely, but is unable to find the rest, it would also be an undesirable trait. We defined the units ratio as the ratio of the number of SUs to the number of GTUs on a certain electrode.

5.2.3.4. Retrieved Units

Most metrics discussed so far looked at the classification performance for a pair of matched units, which is a local measure of performance. However, we also want to look at the algorithm's global performance across the dataset. The number of units that a sorter retrieves is a general measure to ascertain how closely the SR matches the GT for a certain parameter set. Thus, for each parameter set, we determine how many GTUs were matched across the dataset with SUs and use this number as a metric for each parameter set.

6. Results

In this chapter we describe the results of the evaluation of the two spike sorting algorithms based on the concepts and tools introduced earlier. We first discuss the parameter scans of the two spike sorting algorithms by comparing the outcomes to a sorting performed by a human expert, and, thus infer the set of optimal parameter sets for each. Then, we show the results of spike sorting surrogate data with these optimal parameters.

6.1. Parameter Scans

Each spike sorting algorithm was subject to a detailed parameter scan over the space spanned by the parameters defined in 2.2 and 2.3. We performed the scan in two passes. In the first pass, we chose sparse values for each parameter so as to determine which subsets of parameters significantly affect the spike sorting result. We call this subset the *primary parameters*, while the rest are called *secondary parameters*. For the second phase, we performed a more fine-grained search for each primary parameter while keeping the secondary parameters at constant, optimal values. This approach allowed us to eliminate the confounding effect of the secondary parameters on our performance metrics.

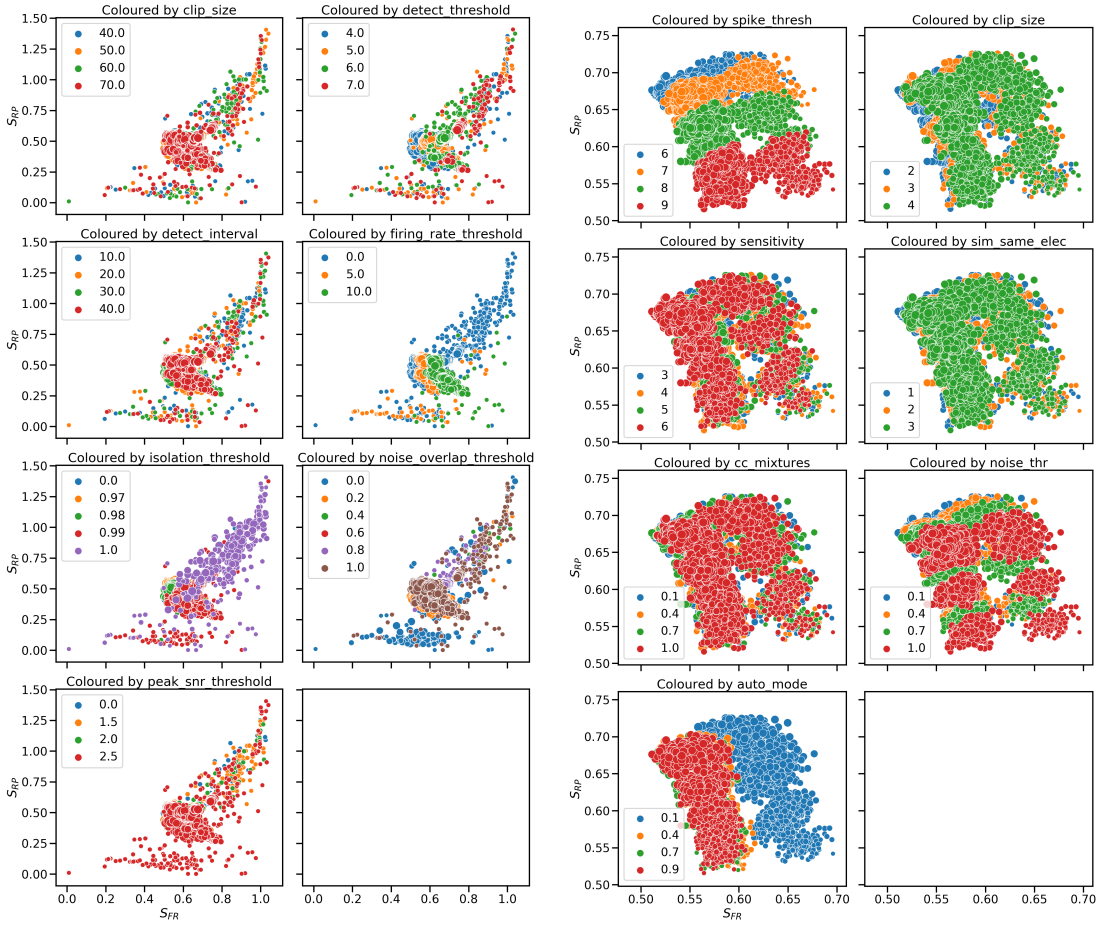
6.1.1. MountainSort

For MountainSort, we chose the parameter values tabulated in Table 6.1. In the first pass, by varying each parameter over the specified values, we cover a total of 18,432 parameter sets. From the results of each parameter set, we obtain multiple matched pairs across all 96 electrodes. For each parameter set, we calculate the mean S_{FR} value, mean S_{RP} value and the total number of retrieved GTUs, summarized in Figure 6.1a. We observe the following:

1. Only the parameters `detection_threshold`, `firing_rate_threshold`, `isolation_threshold` and `noise_overlap_threshold` appear to have an effect on the spike sorting performance, since the colours corresponding to the different parameter values have a discernible structure only in those plots.

2. We cannot determine any parameter set which lies close to the point of optimum and retrieves a relatively large number of GTUs, because all points towards the lower left corner of each plot have a small size.

3. Certain parameter sets (those in the top right corner of each plot) retrieve a relatively small number of units and also perform poorly.



(a) MountainSort **(b)** Spyking Circus

Figure 6.1.: Mean S_{FR} vs. mean S_{RP}, coloured by the various parameter values for both MountainSort (a) and Spyking Circus (b). Each single point represents the mean S_{FR} and mean S_{RP} values of a single parameter set, averaged over all matched pairs discovered across all electrodes. Within a panel, each plot contains points at the same positions, however the size colours and sizes of the points vary across the plots. The size of each point indicates the relative number of GTUs retrieved by that parameter set (hence, larger points indicate a better spike sorting). The colour indicates the value of the parameter being varied in each plot. This plot gives an overview of the classification performance of both spike sorters. Note the difference in the x- and y-limits between (a) and (b).

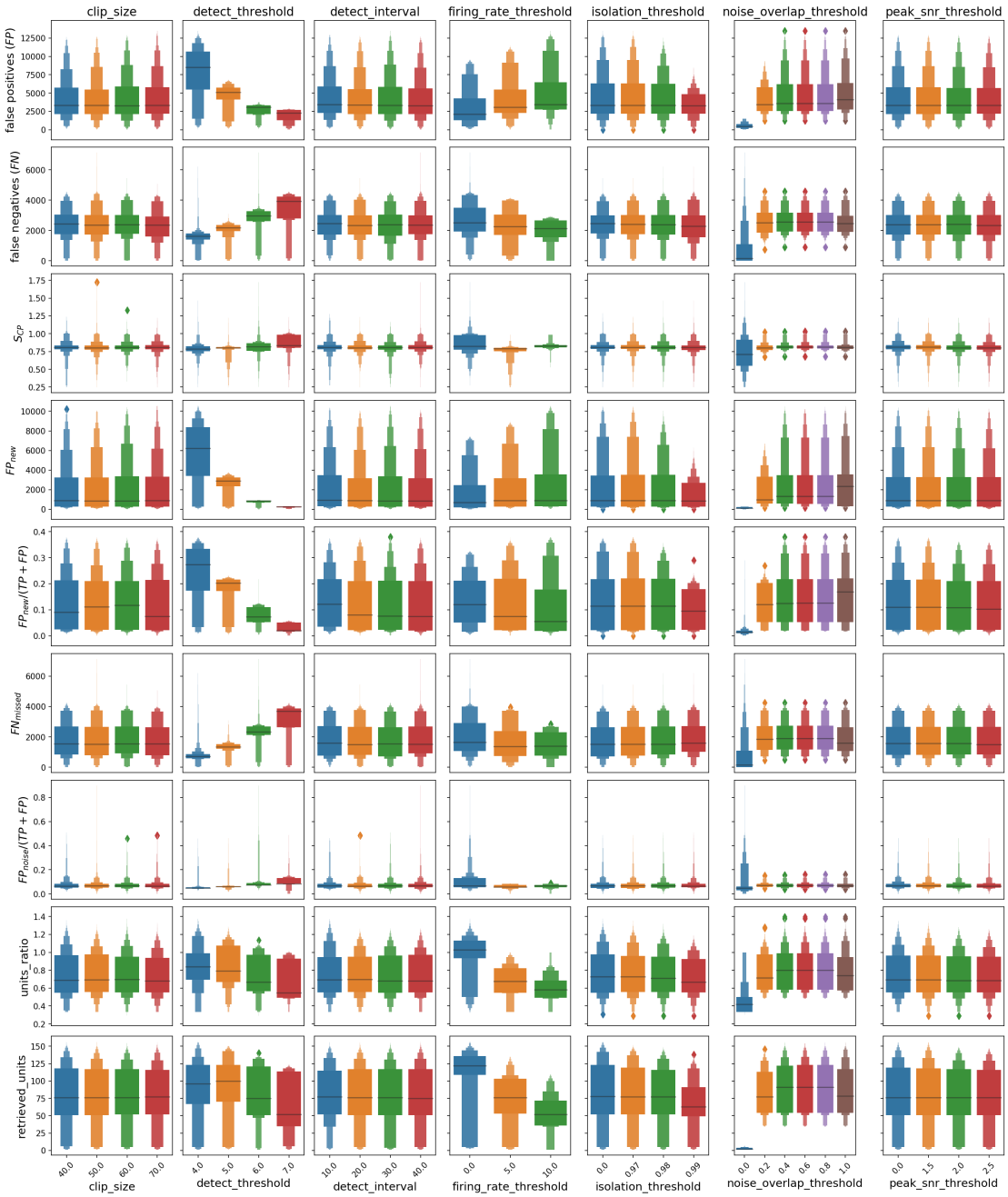


Figure 6.2.: Letter value plots showing distributions of various metrics for individual values of each parameter of MountainSort during the first pass of the parameter scan.

Parameter Name	Parameter Type	First Pass	Second Pass
`detect_threshold`	primary	4, 5, 6, 7	4, 5, 6, 7
`detect_interval`	secondary	10, 20, 30, 40	10
`clip_size`	secondary	40, 50, 60, 70	40
`firing_rate_threshold`	secondary	0.0, 5.0, 10.0	0.0
`isolation_threshold`	primary	0.0, 0.97, 0.98, 0.99	0.97, 0.975, 0.98, 0.985, 0.99, 0.995
`noise_overlap_threshold`	primary	0.0, 0.2, 0.4, 0.6, 0.8, 1.0	0.1, 0.2, 0.3, 0.4, 0.5
`peak_snr_threshold`	secondary	0.0, 1.5, 2.0, 2.5	1.5

Table 6.1.: MountainSort parameter values used for the first and second passes of the parameter scan.

The classification performance scores alone, however, are not sufficient to select good parameter sets for spike sorting, and we need to observe the effect of the individual parameters on more aspects of the spike sorting result. To this end, we make use of letter value plots (see Appendix A for a brief introduction on how to read letter value plots). Figure 6.2 shows the variation of different performance metrics with individual values of spike sorting parameters. The letter value plots show distributions of the values of each performance metric over all matched pairs in the dataset. They provide a detailed overview of the spike sorting performance across multiple spike sorting parameters. Several inferences can be drawn from this figure:

1. In line with the observations from Figure 6.1a, only the parameters `detect _threshold`, `isolation_threshold`, `firing_rate_threshold` and `noise_overlap_threshold` have an effect on the spike sorting quality, whereas the parameters `clip_size`, `detect_inteval` and `peak_snr_threshold` do not seem to affect it. This is evident by the way the distributions of the performance metrics change when the corresponding spike sorting parameter is changed.

2. The parameter `detect_threshold` affects the spike sorting performance most significantly. For a `detect_threshold` of 4, a typical matched pair tends to have an unacceptably large number of false positives. We do not exclude that value in the second pass of the parameter scan, though, to see if the number of false positives decreases when other confounding parameters are removed.

3. The parameter `noise_overlap_threshold` appears to plateau in its performance above a value of 0.4, which we will use as an upper bound for the second pass of our parameter scan.

4. The parameter `isolation_threshold` affects the spike sorting performance only with values close to 1.0, so we will use only those values in the second pass.

5. Certain parameters have an intuitive explanation, such as `firing_rate_threshold` (minimum required firing rate for each unit) and `peak_snr_threshold` (minimum required SNR for each unit). They are used in the final curation phase of the algorithm and only play a role in filtering out units which do not match the specified criteria. Instead of determining optimal parameters for these, we use values typically used by experimentalists working on this dataset. These can and should be adjusted to the dataset being worked on since the typical firing rate and SNR of a unit is likely to change based on the source and quality of the data.

To summarize, we determine values for the second pass of the parameter scan, given in Table 6.1, by analysing the letter value plots in Figure 6.2. The results for the second pass, wherein we covered a total of 120 parameter sets, are shown using the strip plots in Figure 6.3a. Strip plots give a more accurate representation of the distribution of the underlying data when using fewer parameter sets, due the relative sparseness of the data. We infer the following from the strip plots.

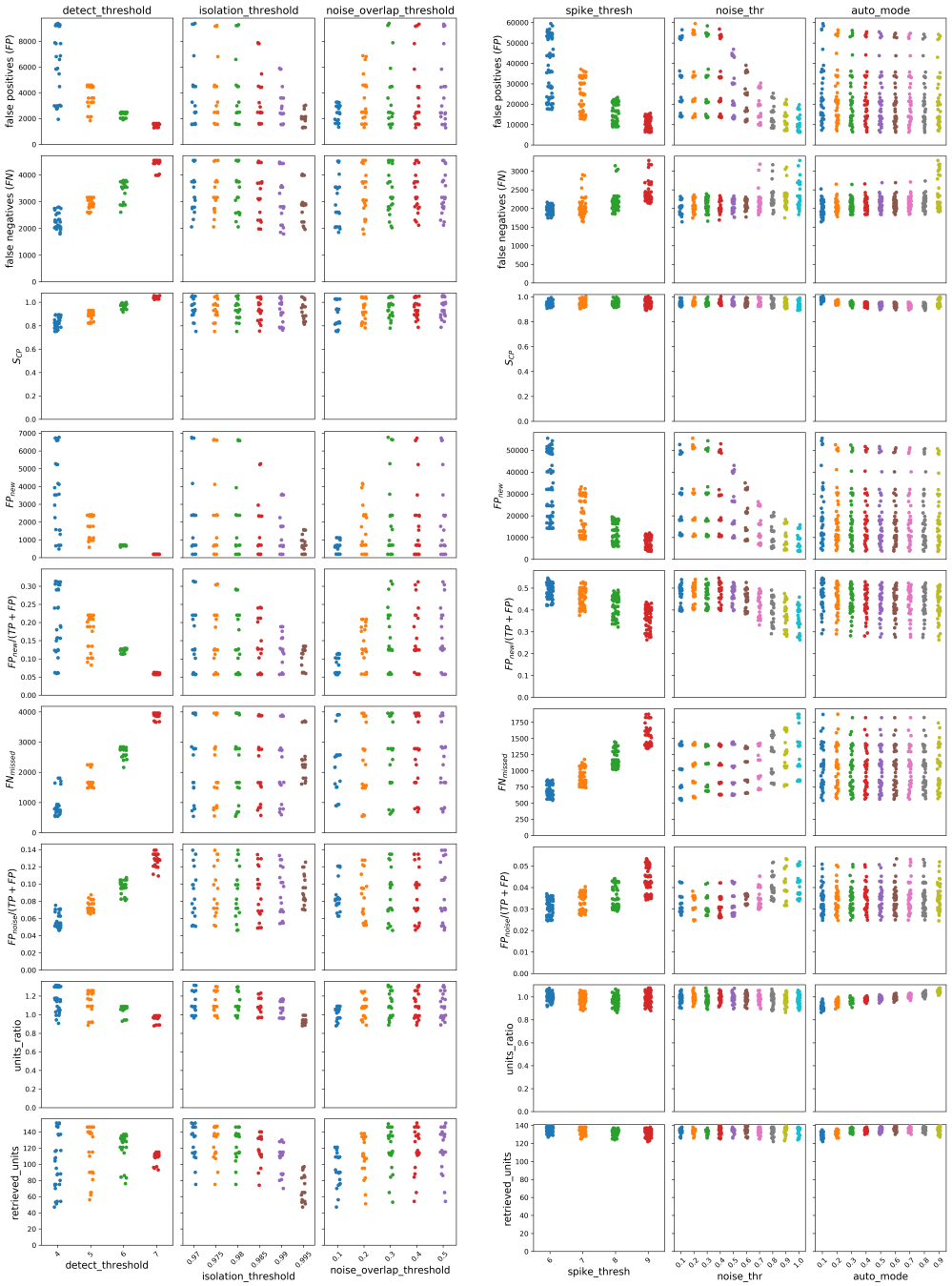
1. The `detect_threshold` parameter strongly affects the quality of spike sorting. A value of 4 for this parameter yields large numbers of FP and FN events, in line with our observations from the first pass. A value of 5, however, shows a good balance between the number of retrieved units (last row in Figure 6.3a) and classification performance (third row in Figure 6.3a). Thus, we set a value of 5 as the optimal `detect_threshold`. Higher values lead to a more conservative sorting.

2. The `isolation_threshold` parameter shows a marked, albeit detrimental, effect on the sorting performance for values above 0.985. Thus, we set the value of 0.985 to be optimal for this parameter. Higher values of this parameter reduce the numbers of FP and FN, along with a relatively large decrease in the number of retrieved units, implying a more conserative performance. Lower values lead to a slightly more liberal performance. There is no evident effect of this parameter on the S_{CP}.

3. The `noise_overlap_threshold` parameter shows a marked change in performance above a value of 0.1 (see FP, FP_{new} and retrieved_units) and then plateaus in its performance. For a value of 0.1, the algorithm retrieves a smaller number of units but with a correspondingly smaller number of FP events. We, however, set a value of 0.2 as optimal in order to find a balance between the number of units retrieved and the number of FN and FP events. Lower values for the `noise_overlap_threshold` may be used for a more conservative spike sorting, whereas higher values would yield a slightly more liberal performance.

4. Overall, we observe a sparse structure in the results of the second pass, wherein points are clustered around certain values for each metric, rather than forming a continuum.

Additional figures which support our observations are shown in the Appendix B. To summarize, we list parameters selected for different scenarios in Table 6.2.



(a) MountainSort **(b)** Spyking Circus

Figure 6.3.: Strip plots showing distributions of various metrics for individual values of each parameter of MountainSort during the second pass of the parameter scan.

Parameter Name	Optimal	Liberal	Conservative
`detect_threshold`	5	4	6
`detect_interval`	10	10	10
`clip_size`	40	40	40
`firing_rate_threshold`	0.0	0.0	0.0
`isolation_threshold`	0.985	0.97	0.99
`noise_overlap_threshold`	0.2	0.3	0.1
`peak_snr_threshold`	0.0	0.0	0.0

Table 6.2.: Selected parameter sets for MountainSort. Optimal set represents a balance between liberal and conservative sorting.

Parameter Name	Parameter Type	First Pass	Second Pass
`spike_thresh`	primary	6, 7, 8, 9	6, 7, 8, 9
`clip_size`	secondary	2, 3, 4	2
`sensitivity`	secondary	3, 4, 5, 6	6
`sim_same_elec`	secondary	1, 2, 3	3
`cc_mixtures`	secondary	0.1, 0.4, 0.7, 1.0	0.1
`noise_thr`	primary	0.1, 0.4, 0.7, 1.0	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
`auto_mode`	primary	0.1, 0.4, 0.7, 0.9	0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9

Table 6.3.: Spyking Circus parameter values used for the first and second passes of the parameter scan.

6.1.2. Spyking Circus

We use the same procedure for the parameter scans for Spyking Circus as we did for MountainSort. The parameter values we use for spyking circus are tabulated in Table 6.3. In the first pass, we cover a total of 9,216 parameter sets, and analyze all matched pairs across all 96 electrodes for each parameter set. We calculate the mean S_{FR}, mean S_{RP} and the total number of retrieved units for each parameter set and generate the plots shown in Figure 6.1b. From these, we observe the following.

1. The parameters `spike_thresh`, `noise_thr` and `auto_mode` affect the quality of the sorting significantly, while the other four parameters do not seem to have an appreciable effect on the sorting performance. This can be seen by the movement of the cluster of points corresponding to different parameter values across the S_{FR}-S_{RP} space.

2. All parameter sets are localized in a relatively small region of the S_{FR}-S_{RP} space. This implies that although the algorithm is highly parameterised, the performance is not very sensitive to parameter changes.

3. There is limited variation in the relative sizes of all points, which suggests that the algorithm retrieves about the same number of GTUs irrespective of the underlying parameter set used.

We now make use of letter value plots to consolidate information from other performance metrics for the results from the first pass of the parameter scan. These are shown in Figure 6.4, from which we infer the following.

1. The parameters `spike_thresh`, `noise_thr` and `auto_mode` are the only ones which seem to affect the spike sorting results. Any changes in the values of the other four parameters – `clip_size`, `sim_same_elec`, `sensitivity` and `cc_mixtures` – do not affect the performance metrics significantly.

2. The parameter `spike_thresh` has the largest effect on all performance metrics, with the S_{CP} improving slightly as the parameter value increases. For a parameter value of 6, the number of FP events is on the order of tens of thousands and a typical matched pair has approx. 40% FP_{new} events. In comparison, for a parameter value of 9, there is a three-fold decrease in the number of FP events (median values from approx. 22000 to approx. 7000) and only a 1.7-fold increase in the number of FN_{missed} events (median values from approx. 1000 to approx. 1700). We use the same values of this parameter for the second pass of the parameter scan in order to remove the confounding effect of secondary parameters.

   - The total number of FN events remains almost the same for different values of `spike_thresh`. This is further explored in Figure 6.5, where we show the mean number of events belonging to different subcategories of FP and FN events for different values of `spike_thresh`. The number of missed FN events increase and the number of misclassified FN events decrease with increasing `spike_thresh`, keeping the overall number of FN constant. The number of FP events decreases with `spike_thresh` mainly due to fewer newly detected spikes.

3. The parameters `noise_thr` and `auto_mode`, both, show interesting trends in their performance metrics. The overall performance, characterized by the S_{CP}, is better for higher values of each parameter. A change in the parameter `auto_mode` from a value of 0.1 to 0.4 leads to a sharp improvement in performance. A similar trend is observed for `noise_thr` between values of 0.7 and 1.0, wherein the number of FP events sharply decreases along with a modest increase in FN events. For both these parameters, we probe finer values in the second pass of the parameter scan.

4. The parameter `sensitivity` has a seemingly subtle effect on the overall performance (tendency to find less FP events for higher `sensitivity` values). In order to limit the parameter space covered in the second pass, we set a value of 6 to be optimal.

In summary, we have determined the set of parameter values to be explored in the second pass, given in Table 6.3. The results of the second pass, in which 360 parameter sets are covered, are shown in Figure 6.3b using strip plots. From these, we infer the following.

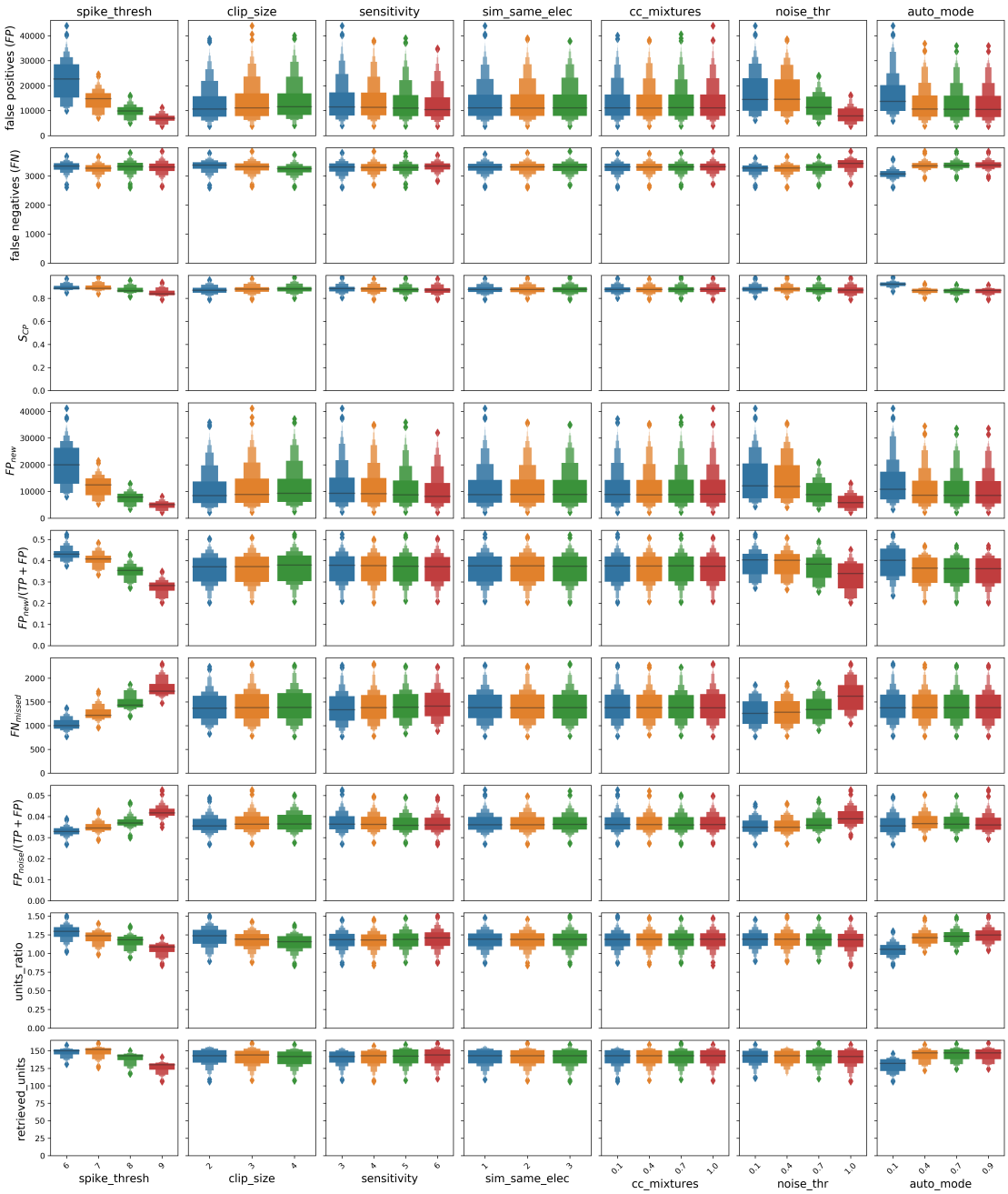
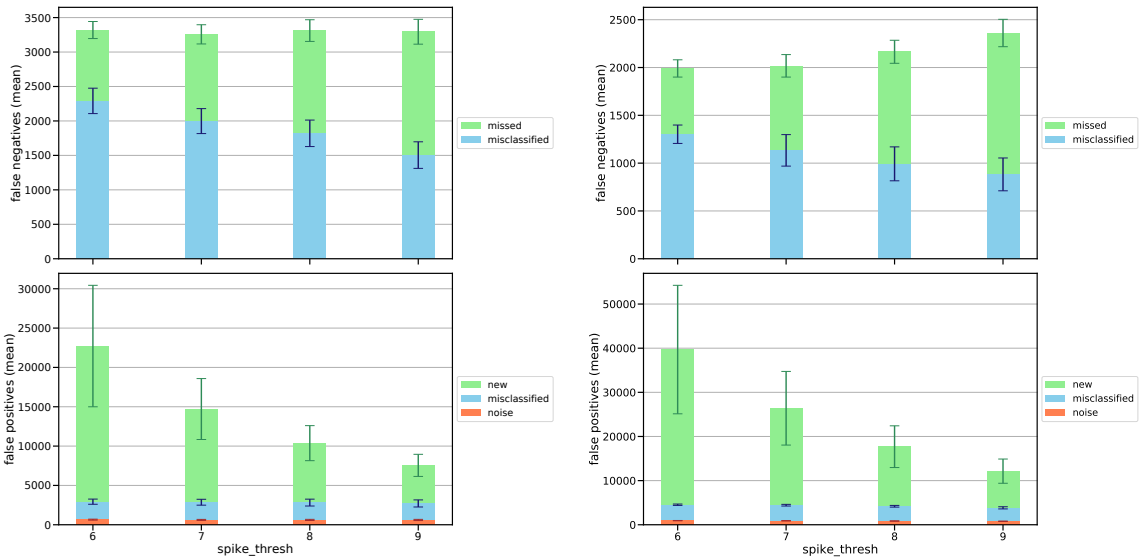


Figure 6.4.: Letter value plots showing distributions of various metrics for individual values of each parameter of Spyking Circus during the first pass of the parameter scan.



(a) First Pass **(b)** Second Pass

Figure 6.5.: Mean number of events for all subcategories of FP events (bottom row) and FN events (top row) for different values of `spike_thresh` during (a) the first pass and (b) the second pass of the parameter scan. In the first pass (a), the total number of FN events remains almost constant with an increase in `spike_thresh` due to an increase in FN_{missed} and a decrease in $FN_{misclassified}$ events. At the same time, the number of FP_{new} events decreases drastically, with other subcategories of FP events remaining largely unchanged. In the second pass (b), the number of FN events increases slightly, although the FN_{missed} and $FN_{misclassified}$ events follow a similar trend as in (a). The FP events follow the same trend as in (a).

Parameter Name	Optimal	Liberal	Conservative
`clip_size`	2	2	2
`spike_thresh`	7	6	8
`sim_same_elec`	3	3	3
`sensitivity`	6	6	6
`cc_mixtures`	0.1	0.1	0.1
`noise_thr`	0.9	0.6	1.0
`auto_mode`	0.8	0.8	0.8

Table 6.4.: Selected parameter sets for Spyking Circus. Optimal set represents a balance between liberal and conservative sorting.

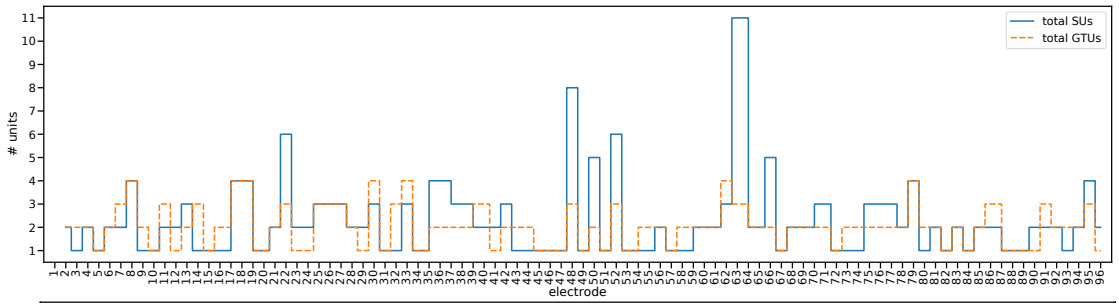
1. The parameter `spike_thresh` has the largest effect on the spike sorting performance. For a value of 6, the number of FP events varies considerably (between approx. 18000 events to approx. 60000 events), owing to the large variations induced by the different values of `noise_thr`. The number of FP events decreases considerably with an increase in the `spike_thresh` parameter value, along with a modest increase in the number of FN events. The number of retrived GTUs remains largely constant across the different parameter values. Taking all of this into consideration, we choose a value of 7 to be optimal for this parameter. Higher values would yield increasingly conservative results.

2. The parameter `noise_thr`, when combined with small parameter values of `spike_thresh`, yields an unreasonably large number of FP events. For larger values of this parameter, though, the number of FP events is significantly reduced (however, this is still much larger than for any parameter set of MountainSort, see Figure 6.3a). For values above 0.6, there is a sudden rise in the number of FN events, however this is due only to the parameter sets with an `auto_mode` value of 0.9 (see right column of Figure 6.3b). `noise_thr` does not affect the number of retrieved GTUs. We choose 0.9 as the optimal value for this parameter to balance the FP and the FN events.

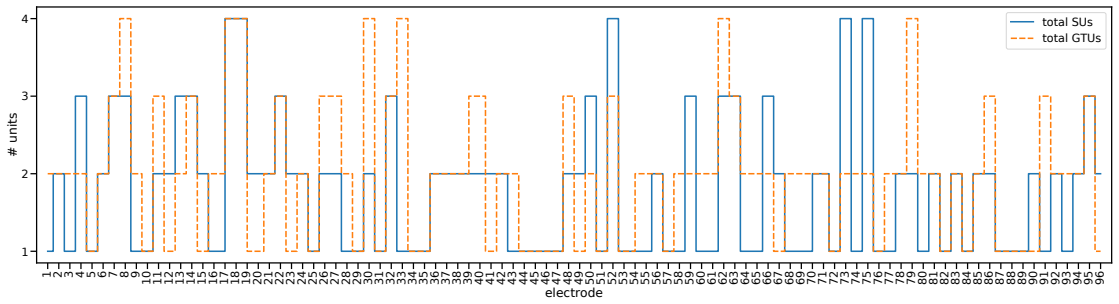
3. With increasing values of the parameter `auto_mode`, the performance steadily improves, until a value of 0.8. For a value of 0.9, the number of FN events increases suddenly, with other metrics showing no such sudden change. We, thus, choose 0.8 as the optimal value of this parameter.

4. Overall, the algorithm retrieves around the same number of GTUs irrespective of the parameter values used. Also, the performance score S_{CP} does not change appreciably with changes in parameter values. These inferences are in line with the observations made from Figure 6.1b.

Additional figures which support our observations are shown in the Appendix B. We list the chosen parameter values for Spyking Circus in Table 6.4.



(a) MountainSort



(b) Spyking Circus

Figure 6.6.: Number of units found on each electrode. The number of SUs (blue) and the number of GTUs (orange) found on each electrode are shown for the two spike sorting algorithms. Both algorithms were run using optimal parameters. Notably, MountainSort finds up to 11 SUs on certain electrodes (electrodes 48, 52, 63). Spyking Circus, on the other hand, never finds more than 4 units.

6.1.3. Evaluation of Optimal Parameter Sets

Now that we have determined optimal parameter sets for both spike sorting algorithms in the sense that we explained above, we look a little deeper into the results obtained using these parameters.

Spyking Circus retrieved 138 GTUs and MountainSort retrieved 134 GTUs, from a total of 175 GTUs. Figure 6.6 shows the distribution of units across electrodes. The algorithms do not always find as many units on each electrode as there are in the ground truth. On certain channels, MountainSort finds upto 15 SUs – much more than on any GT channel. This is likely due to the algorithm's tendency to include large and recurring events which may not arise from spiking activity. This information is represented differently in Figure 6.7, where we see that despite having found no units on a comparable number of electrodes, MountainSort found many more units in total than the ground truth. Many of the extra units are accounted for by single electrodes containing large numbers of units (electrodes 48, 50, 52, 63, 66). Spyking Circus finds units on most electrodes – sometimes even on those where the ground truth contained none – hinting at an algorithm that readily accepts even small amplitude events into units. While excess

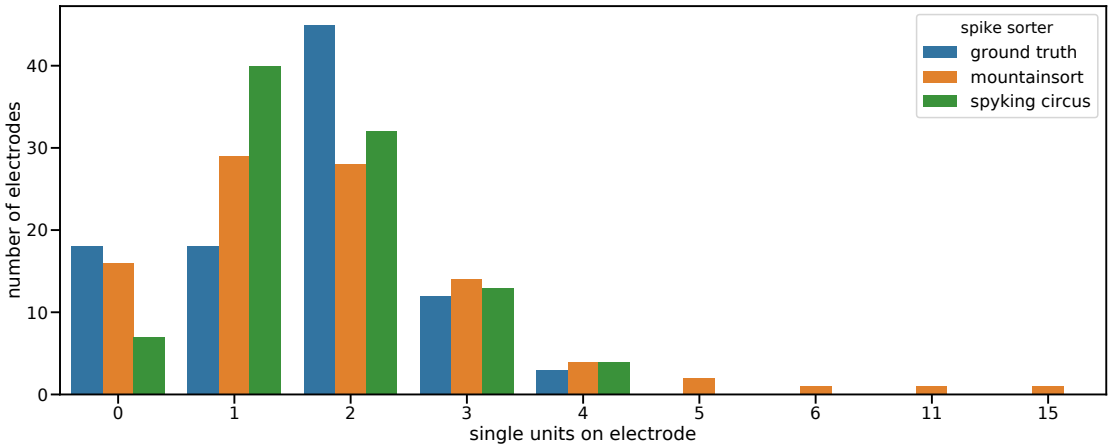


Figure 6.7.: Number of channels with a given number of sorted units, for the ground truth, MountainSort and Spyking Circus. The plot describes how many electrodes (y-axis) were found to contain a given number of units (x-axis) by a certain spike sorter (in different colours). Total number of units found – ground truth: 156, Spyking Circus: 159, MountainSort: 185. Spyking Circus does not find any units on 7 electrodes and MountainSort does not find any units on 16 electrodes, compared to 18 in the ground truth. Only MountainSort shows a tendency to find more than 4 units on multiple electrodes.

clusters can be manually removed, this hampers an automated workflow and hints at a greedy algorithm. Spyking Circus finds exactly one unit on 40 electrodes, compared to 18 electrodes in the ground truth. This is also seen in Figure 6.6b.

We investigate in more detail the classification performance of the algorithms on each electrode in Figure 6.8. We find that overall, Spyking Circus performs better than MountainSort when we consider only the classification performance. Spyking Circus finds units with a mean $S_{CP} < 0.25$ on 9 electrodes, whereas MountainSort's units meet that criterion only on 2 electrodes. Further, Spyking Circus finds units with a mean $S_{CP} > 0.5$ on 46 electrodes, whereas MountainSort's units meet that criterion only on 34 electrodes (ignoring electrodes where the respective algorithm found no units). The mean S_{CP} scores across all electrodes where the algorithms found units are relatively high – MountainSort: 0.48, Spyking Circus: 0.50 – hinting at a less than satisfactory performance from both algorithms. Panel 2 of the figure shows a stark difference between the two algorithms, wherein Spyking Circus accepts a large number of FP events compared to MountainSort. In panel 4, we see that Spyking Circus has a tendency for fewer FN events than MountainSort. Specifically, it finds less FN events than MountainSort on 49 electrodes out of 80 where both algorithms found units.

We compare the distributions of SNR values for units found by both algorithms in Figure 6.9. Units found by Spyking Circus have significantly lower SNR values than those in the ground truth or by MountainSort. Units found by MountainSort have comparable SNR values as the ground truth. This suggests that Spyking Circus yields

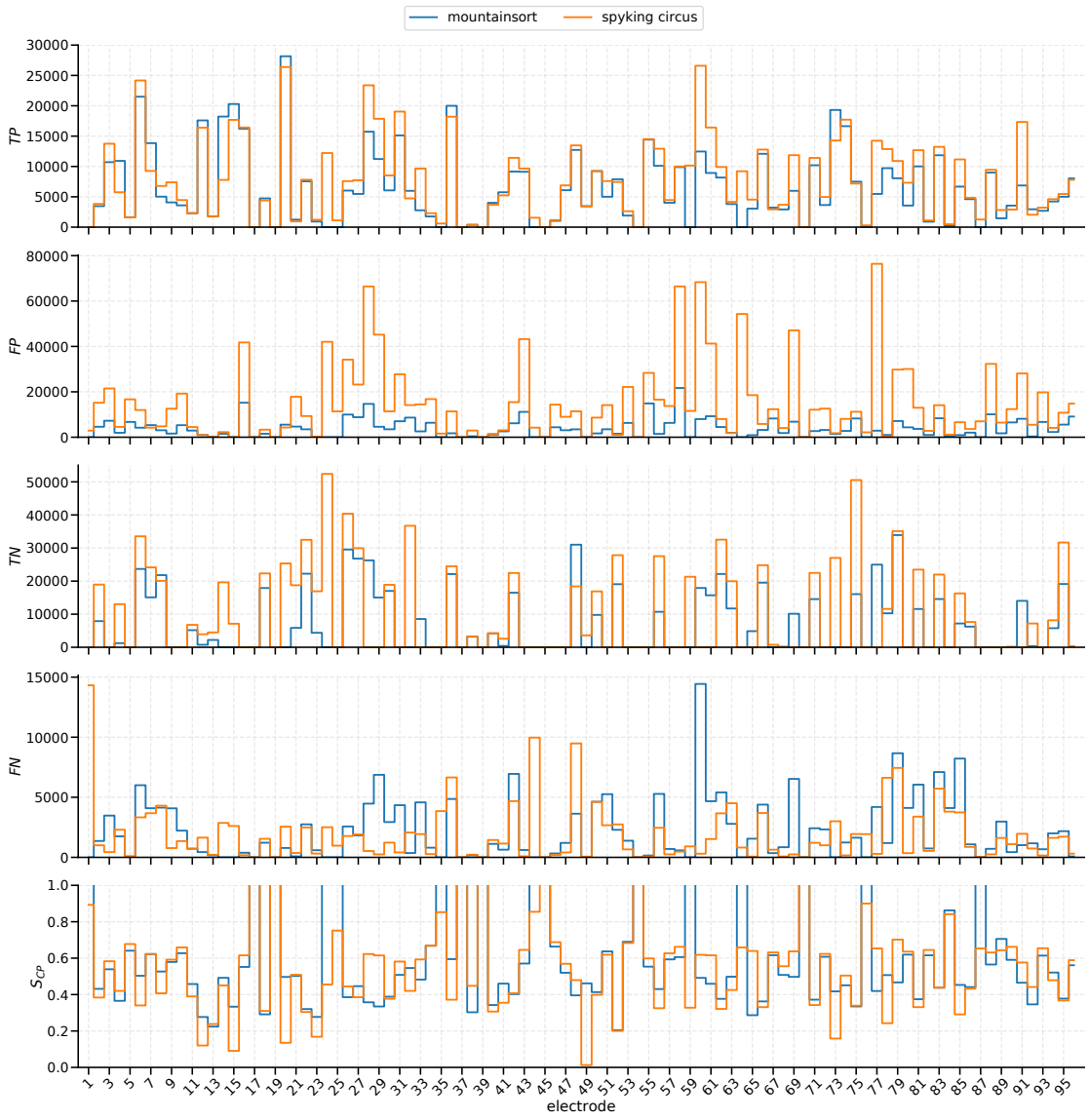


Figure 6.8.: Comparison of the classification performance of the two spike sorting algorithms. Panel 1 (top row) to panel 5 (bottom row) show different aspects of the classification performance for Spyking Circus (orange in all panels) and MountainSort (blue in all panels). Panels 1, 2, 3 and 4 show the mean number of TP, FP, TN and FN events on each electrode, respectively. Panel 5 shows the mean S_{CP} values. The mean values are calculated by averaging over the values of all matched pairs on each electrode. In Panel 5, on whichever electrode the algorithm did not find a unit (refer to figure 6.6), we set the S_{CP} score such that it exceeds the upper limit of the plot.

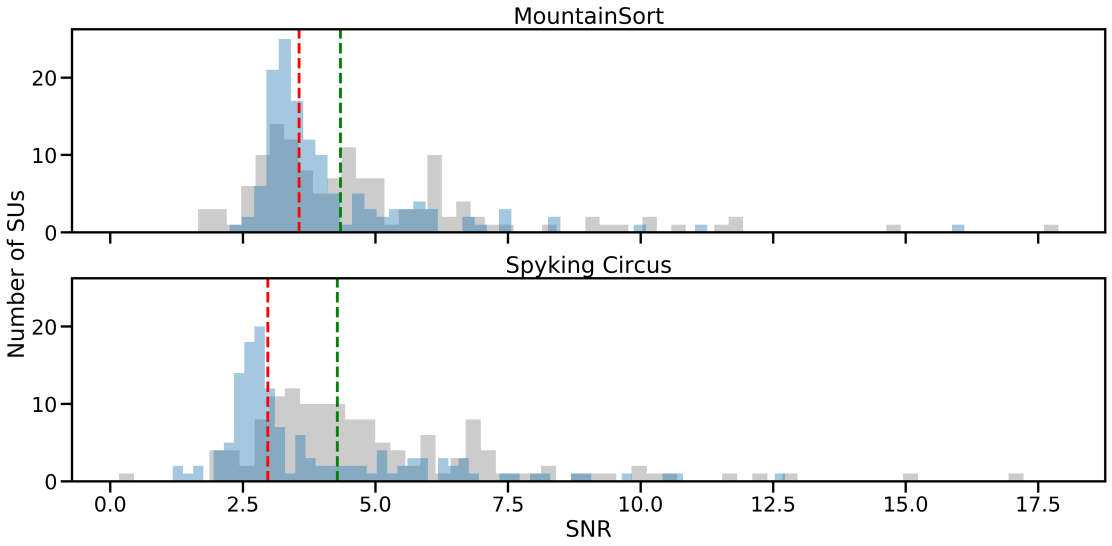


Figure 6.9.: Comparison of the distribution of SNR values. In each plot, the distribution of the SNR values for units found by the automatic algorithm is shown in blue (median shown in red) and the distribution of the SNR values of the GTUs matched by the algorithm is shown in grey (median shown in green). Spyking Circus yields units with lower SNR values than MountainSort.

units of lower quality than MountainSort.

In order to better understand the nature of the differences between the two algorithms, we created overview figures on a per-channel basis for each algorithm (see Figure 6.10 for an example). observations are corroborated by the overview Figure 6.11. Also, the excess units for MountainSort on electrode 48 (Figures 6.12 and 6.13) have a low firing rate and can be removed by setting the `firing_rate_threshold` parameter to a higher value.

Figure 6.14 compares the results on an electrode containing units with overlapping events and a large difference in amplitude. While both algorithms correctly identify the large unit, only Spyking Circus is able to retrieve one of the smaller units to a certain extent, highlighting the fact that Spyking Circus is able to find small units in the presence of larger units, despite overlaps. It should also be noted, however, that the smaller unit was only partially retrieved.

6.2. Performance on Surrogate Datasets

After having determined optimal parameter sets for each algorithm, we test their applicability on artificially generated surrogate datasets.

We generate datasets by the methods described in Chapter 3. The configuration for each channel is done individually (as per Table 6.5) and 200 realisations of each surrogate dataset are created, such that background signals which were extracted from real data

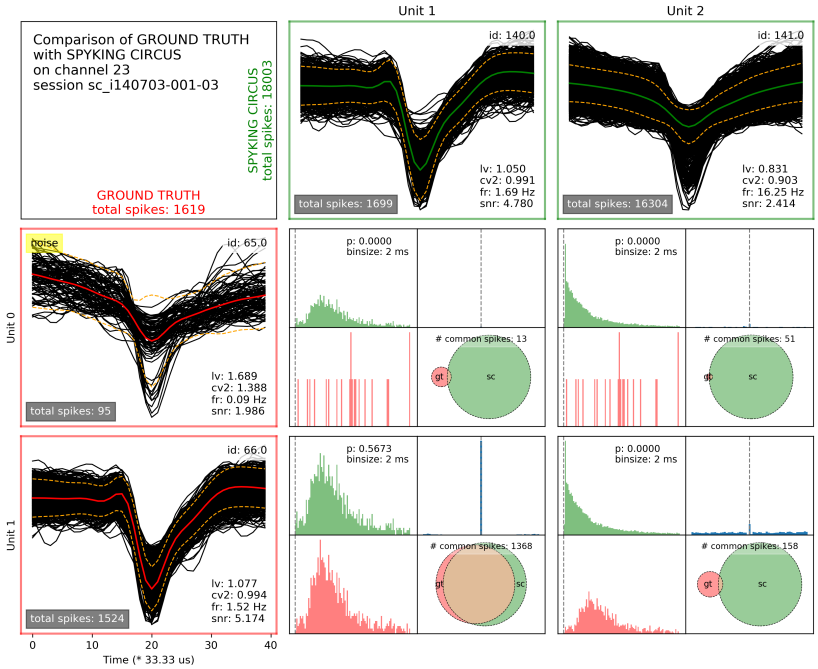
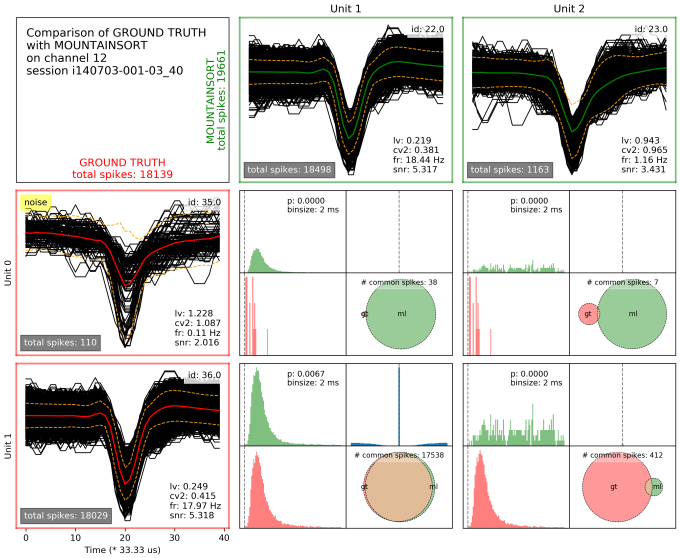
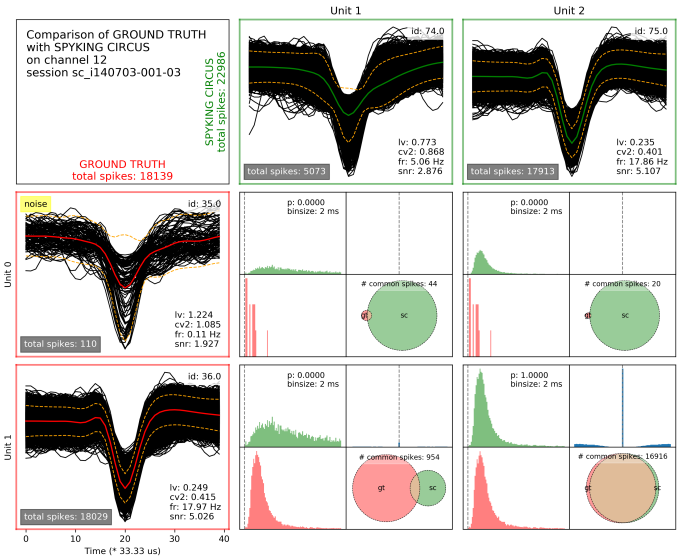


Figure 6.10.: Electrode overview figure to visually compare the results of an automatic algorithm on a single electrode to ground truth. Each figure has a tabular structure, with cells, at the intersection of rows and columns, containing plot elements that display a comparison of one GTU to one SU. In a figure, all plot elements coloured red correspond to the ground truth, whereas those coloured green correspond to the spike sorting algorithm. The first cell (top left) shows general information regarding the figure, along with the total number of spikes found on that electrode in the ground truth/sorting algorithm. Each row below the first row corresponds to one GTU. Each column to the right of the first column corresponds to one SU. For each SU/GTU, the first cell in the column/row shows the corresponding waveforms (plotted in black), along with their mean (in green/red) and 2 SDs around the mean (in dashed orange). Additional information about the unit is displayed in the corners – firing rate, CV_2 (Holt et al., 1996), LV (Shinomoto et al., 2003), SNR (Hatsopoulos et al., 2007) and the number of spikes. A label indicating whether a GTU was classified as "noise" or "mua" by the human expert is also shown where applicable. Each of the remaining cells compares the corresponding GTU (row) and SU (column), and is divided into four boxes. The lower left box shows the inter-spike interval histogram (ISIH) of the corresponding GTU, while the upper left box shows the ISI of the corresponding SU (both binned at 2ms). The upper left box also displays the p-value for the 2-sample Kolmogorov-Smirnov test comparing the two ISIHs, with a higher p-value suggesting more similar distributions. The lower right box contains a Venn diagram representing the overlap of spike times between the GTU and the SU (overlapping spikes represented in a khaki colour). The sizes of the circles are proportional to the number of spikes in the units. The upper right box shows the cross-correlation histogram between the spike trains of the GTU and the SU (binned at 1ms).



(a) MountainSort



(b) Spyking Circus

Figure 6.11.: Overview figures for electrode 12 for automatic spike sorting by (a) MountainSort and (b) Spyking Circus. See figure 6.10 for a description of the figure. Both algorithms were able to find most of events belonging to the single unit on this electrode. This is evident by the large overlap in the Venn diagrams for both spike sorters. However, both algorithms found one additional unit containing a large number of events.

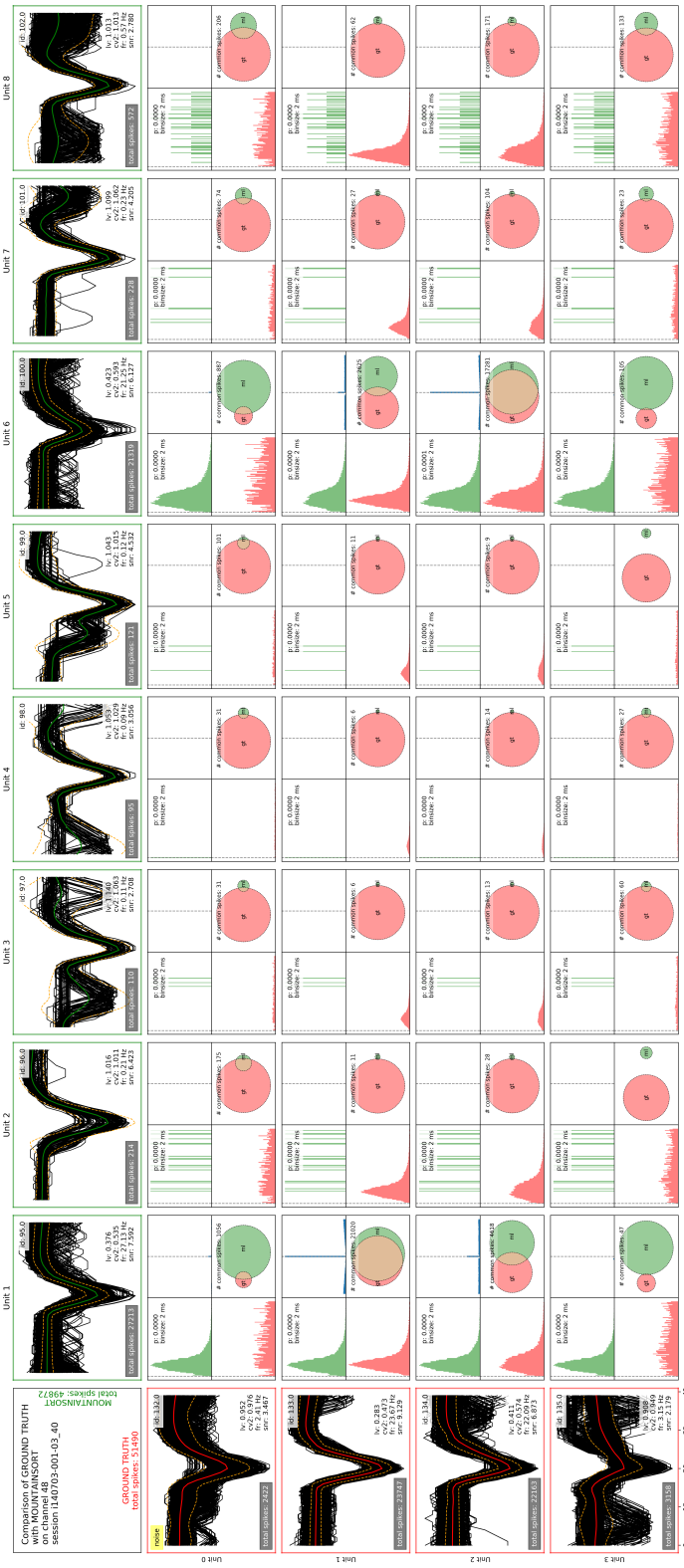


Figure 6.12: Overview figure for MountainSort's performance on elecrode 48. The algorithm found an unusually large number of units on this electrode. The units with no overlap with GTUs have large amplitudes, low firing rates, and a large overlaps with the noise cluster. This suggests a high sensitivity of the algorithm to even small clusters of events. Unit ids 95 and 100 are in good agreement with units 133 and 134 of the ground truth, respectively. If the `firing_rate_threshold` parameter for MountainSort is set to a slightly higher value, all the excess units on this electrode for MountainSort can be removed, leading to a better performance.

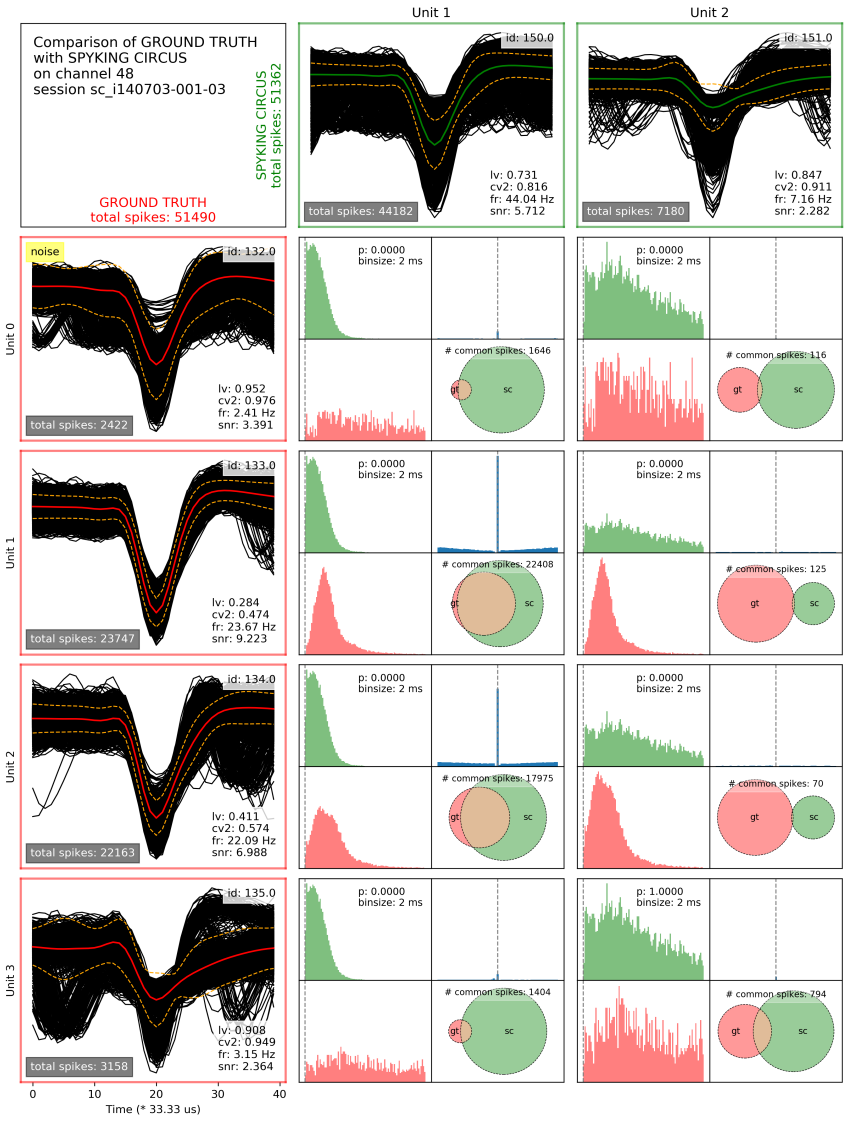
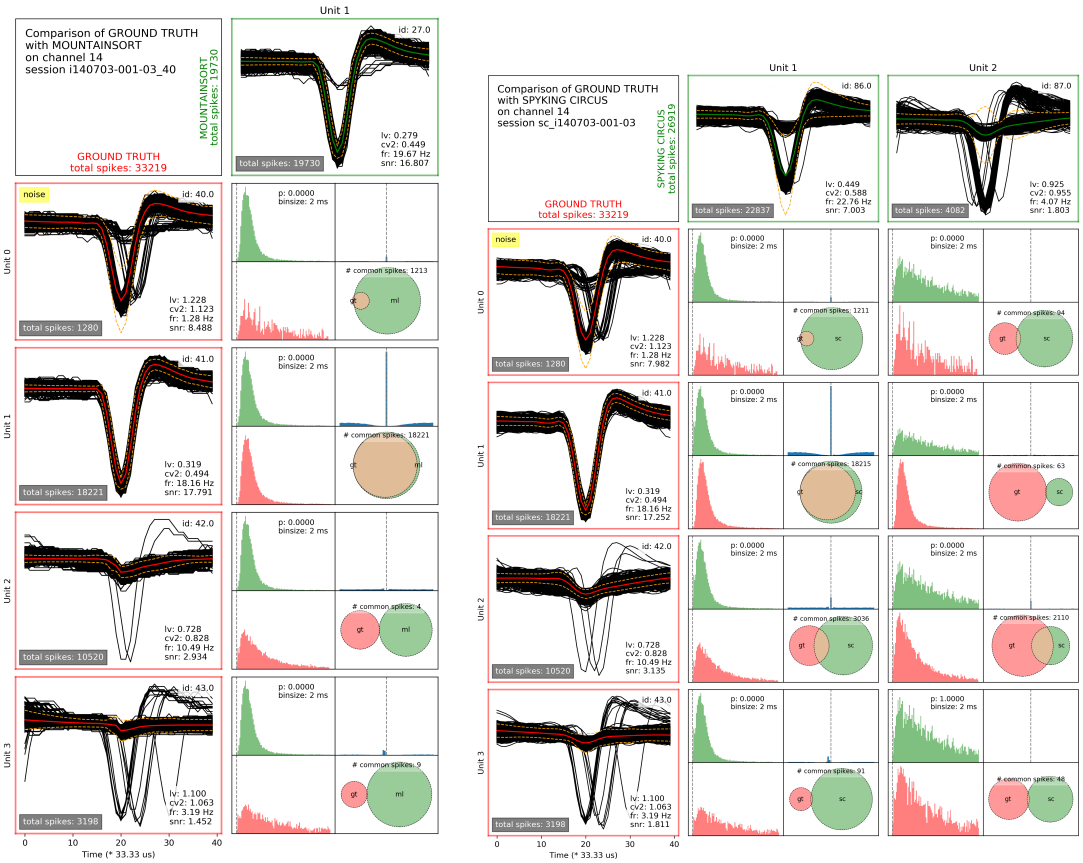


Figure 6.13.: Overview figure for Spyking Circus' performance on electrode 48, to be compared with MountainSort's performance in figure 6.12. Spyking Circus finds only 2 units in comparison to the 8 found by MountainSort. However, Spyking Circus merges the GTUs 133 and 134 into one unit (id 150) and creates a new unit with largely new events which don't exist in the ground truth.



(a) MountainSort **(b)** Spyking Circus

Figure 6.14.: Overview figures for electrode 14 for automatic spike sorting by (a) MountainSort and (b) Spyking Circus. This electrode contains units with a large difference in amplitude and overlapping events. Both algorithms are able to retrieve the large GTU (id 41). Spyking Circus is able to retrieve a portion of the smaller GTU (id 42). However, it is not able to find all of the spikes. This could be improved by using a more liberal set of parameters for Spyking Circus.

elec.	#units	λ_1	λ_2	a_1	a_2	$\Delta\lambda$	ϕ	elec.	#units	λ_1	λ_2	a_1	a_2	$\Delta\lambda$	ϕ
1	1	0	--	8	--	--	--	34	2	0.5	0.8	8	8	0.3	--
2	1	0.1	--	8	--	--	--	35	2	0.6	0.9	8	8	0.3	--
3	1	0.2	--	8	--	--	--	36	2	0.7	1	8	8	0.3	--
4	1	0.3	--	8	--	--	--	37	2	0	0.4	8	8	0.4	--
5	1	0.4	--	8	--	--	--	38	2	0.1	0.5	8	8	0.4	--
6	1	0.5	--	8	--	--	--	39	2	0.2	0.6	8	8	0.4	--
7	1	0.6	--	8	--	--	--	40	2	0.3	0.7	8	8	0.4	--
8	1	0.7	--	8	--	--	--	41	2	0.4	0.8	8	8	0.4	--
9	1	0.8	--	8	--	--	--	42	2	0.5	0.9	8	8	0.4	--
10	1	0.9	--	8	--	--	--	43	2	0.6	1	8	8	0.4	--
11	1	1	--	8	--	--	--	44	2	0	0.5	8	8	0.5	--
12	1	1	--	4	--	--	--	45	2	0.1	0.6	8	8	0.5	--
13	1	1	--	5	--	--	--	46	2	0.2	0.7	8	8	0.5	--
14	1	1	--	6	--	--	--	47	2	0.3	0.8	8	8	0.5	--
15	1	1	--	7	--	--	--	48	2	0.4	0.9	8	8	0.5	--
16	1	1	--	8	--	--	--	49	2	0.5	1	8	8	0.5	--
17	1	1	--	9	--	--	--	50	2	0	0.6	8	8	0.6	--
18	1	1	--	10	--	--	--	51	2	0.1	0.7	8	8	0.6	--
19	2	0	0.1	8	8	0.1	--	52	2	0.2	0.8	8	8	0.6	--
20	2	0.1	0.2	8	8	0.1	--	53	2	0.3	0.9	8	8	0.6	--
21	2	0.2	0.3	8	8	0.1	--	54	2	0.4	1	8	8	0.6	--
22	2	0.3	0.4	8	8	0.1	--	55	2	0	0.7	8	8	0.7	--
23	2	0.4	0.5	8	8	0.1	--	56	2	0.1	0.8	8	8	0.7	--
24	2	0.5	0.6	8	8	0.1	--	57	2	0.2	0.9	8	8	0.7	--
25	2	0.6	0.7	8	8	0.1	--	58	2	0.3	1	8	8	0.7	--
26	2	0.7	0.8	8	8	0.1	--	59	2	0	0.8	8	8	0.8	--
27	2	0.8	0.9	8	8	0.1	--	60	2	0.1	0.9	8	8	0.8	--
28	2	0.9	1	8	8	0.1	--	61	2	0.2	1	8	8	0.8	--
65	2	0	0.2	8	8	0.2	--	62	2	0	0.9	8	8	0.9	--
66	2	0.1	0.3	8	8	0.2	--	63	2	0.1	1	8	8	0.9	--
67	2	0.2	0.4	8	8	0.2	--	64	2	0	1	8	8	1	--
68	2	0.3	0.5	8	8	0.2	--	86	2	0	1	8	10	--	0.1
69	2	0.4	0.6	8	8	0.2	--	87	2	0	1	8	10	--	0.2
70	2	0.5	0.7	8	8	0.2	--	88	2	0	1	8	10	--	0.3
71	2	0.6	0.8	8	8	0.2	--	89	2	0	1	8	10	--	0.4
72	2	0.7	0.9	8	8	0.2	--	90	2	0	1	8	10	--	0.5
73	2	0.8	1	8	8	0.2	--	91	2	0	1	8	10	--	0.6
29	2	0	0.3	8	8	0.3	--	92	2	0	1	8	10	--	0.7
30	2	0.1	0.4	8	8	0.3	--	93	2	0	1	8	10	--	0.8
31	2	0.2	0.5	8	8	0.3	--	94	2	0	1	8	10	--	0.9
32	2	0.3	0.6	8	8	0.3	--	95	2	0	1	8	10	--	1
33	2	0.4	0.7	8	8	0.3	--								

Table 6.5.: Configuration of electrodes in each surrogate dataset. λ_1 and λ_2 are the λ values of two units (for electrodes with two units on them). a_1 and a_2 correspond to the scaling factors of those two units. Electrodes with overlapping units were not used for studying the effect of $\Delta\lambda$. ϕ values denote the overlap fraction for electrodes with two overlapping units.

are randomly assigned to the electrodes in each realisation. The firing rate for all units is kept constant at 5 Hz and the duration of each dataset is fixed at 300 seconds. This yields, on average, 1500 events per GTU. Both algorithms are supplied with the same realisations as input. We then study the effect of the different data generation parameters on spike sorting performance.

6.2.1. Effect of changing λ

The λ value affects the shape of the waveform, but not its amplitude. We vary this parameter to test if the algorithms show a preference for a certain waveform shape. We expect the algorithms to be agnostic of the specific spike shape. We insert a single unit on each electrode with a specific λ value, keeping other parameters constant (refer to Table 6.5 – electrodes 1 to 11). figure 6.15a shows the results of the spike sorting, averaged across the 200 realisations.

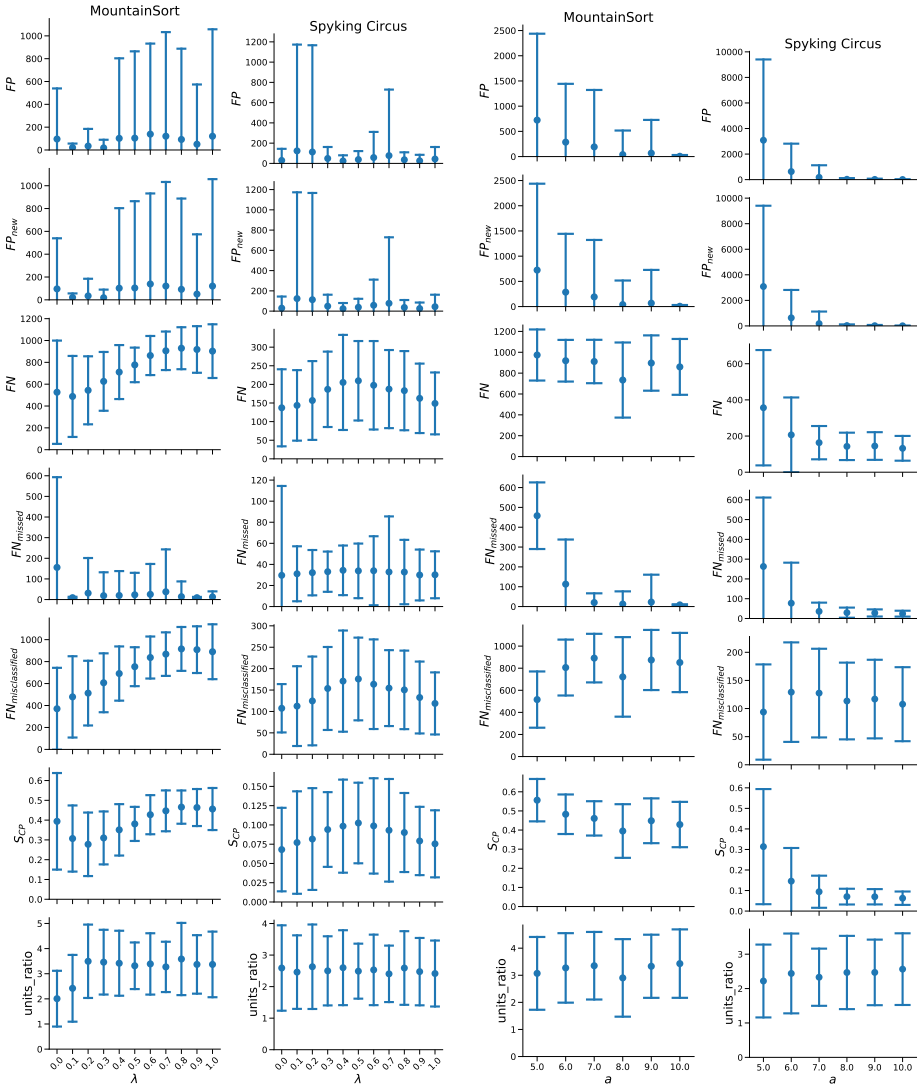
We observe that Spyking Circus remains largely agnostic to the shape of the inserted spike, since none of the performance metrics vary considerably with a change in λ. However, MountainSort shows a preference for units with λ values between 0.1 and 0.4, given that the S_{CP} score drops down to below 0.75 for those values. Moreover, the number of FN events steadily increases with increasing λ, largely due to the $FN_{\text{misclassified}}$ events. No such preference was shown by Spyking Circus and it performed significantly better, with S_{CP} scores averaging around 0.2.

6.2.2. Effect of changing a

The scaling factor a affects the amplitude of the inserted waveforms and is measured in units of standard deviations around the mean of the filtered background signal. We vary this parameter to determine how the amplitude of the waveforms affects the algorithms' performance. We expect the algorithms' performance to increase with increasing values of a, and we also expect a large number of missed events for low values of a. We insert a single unit on each electrode with a specific scaling factor, keeping other parameters constant (refer to Table 6.5 – electrodes 12 to 18). Figure 6.15b compares the performance of the two algorithms, averaged across 200 realisations.

We see that both algorithms' performance improves as we increase the scaling factor. Spyking Circus performs almost optimally for large values of a (S_{CP} score is close to 0), although it finds, on average, twice as many units as we insert. This implies that Spyking Circus has no trouble finding large units.

MountainSort, on the other hand, performs poorly even when it comes to finding a single, large unit. For small scaling factors, the number of FN events suggests that the inserted spikes were of a lower amplitude than the detection threshold for the algorithm. However, the number of FN events does not decrease substantially even for larger scaling factors. We can see that the algorithm does detect most of the events (owing to almost 0 FN_{missed} events for larger values of a), but has a strong tendency to misclassify them (owing to large number of $FN_{\text{misclassified}}$ events). The algorithm also tends to find, on average, 3 times as many SUs as GTUs.



(a) Varying λ **(b)** Varying a

Figure 6.15.: Spike sorting performance of MountainSort and Spyking Circus on surrogate data averaged over 200 realisations, while (a) varying λ and (b) varying a. The error bars denote the standard deviations around the mean. The λ value should ideally not affect the quality of spike sorting since it only determines the shape of the waveform. However, MountainSort shoes a decrease in performance with an increase in λ. Spyking Circus shows no clear preference for λ. The scaling factor a controls the amplitude of the inserted waveforms, and thus, should affect the spike sorting performance. This holds true for both algorithms, with the sorting performance drastically improving with increasing a.

6.2.3. Effect of changing $\Delta\lambda$

In the previous two cases, we considered cases where a single unit was inserted into the background signal. Now, we study the algorithms' ability to retrieve two units from an electrode which were inserted with different values of λ. Each case is characterised by the difference in the λ values of the inserted units – which we refer to as $\Delta\lambda$. We expect the algorithms' performance to improve with increasing $\Delta\lambda$. The two inserted units have the same amplitude relative to the filtered background signal (refer to Table 6.5 – electrodes 19 to 73). The performance of the algorithms is shown in Figure 6.16a.

Both algorithms perform poorly for small values of $\Delta\lambda$, however only Spyking Circus shows an improvement in performance as $\Delta\lambda$ is increased. MountainSort shows an almost consistent performance with a high S_{CP} score of 1.1, on average, implying a poor spike sorting. The main reason for this is the large number of $FN_{\text{misclassified}}$ events, suggesting that the algorithm is incapable of correctly retrieving the inserted GTUs. In the case of Spyking Circus, the performance improves until a $\Delta\lambda$ value of 0.3, after which it plateaus at a S_{CP} score of approx. 0.5. Looking at the `units_ratio` metrics, it also find fewer units than MountainSort.

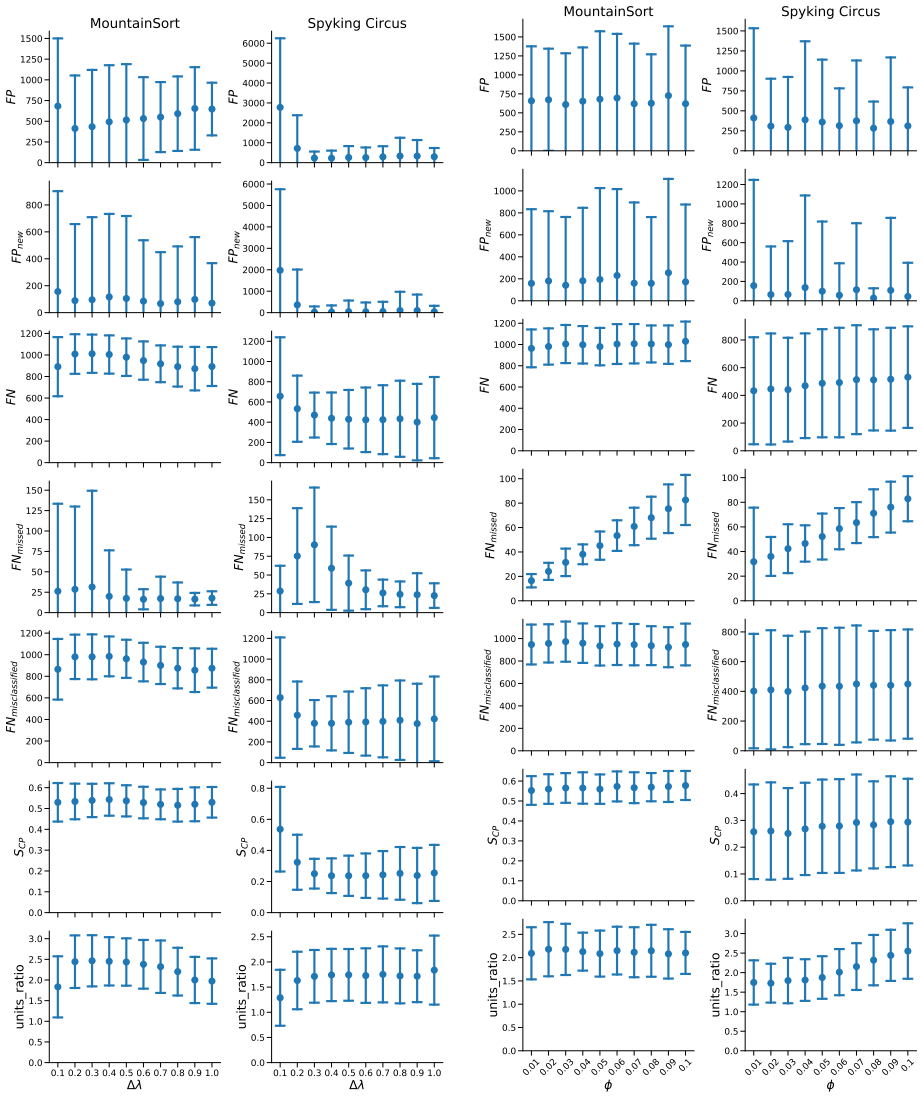
6.2.4. Effect of changing ϕ

Lastly, we introduce two units on each electrode with a certain known fraction of overlapping events – characterized by the overlapping fraction, ϕ. The λ and a parameters are kept constant for each unit (refer to Table 6.5 – electrodes 74 to 83). Given that MountainSort does not have any mechanism by which it can retrieve overlapping events, we expect it to perform poorly. Spyking Circus, on the other hand, has the ability to distinguish overlapping units. Hence, we expect Spyking Circus to perform better than MountainSort for this scenario. The performance of the two algorithms in such a scenario is compared in Figure 6.16b.

As expected, Spyking Circus has a lower S_{CP} score by almost a factor of 2, despite having a much larger number of FP_{new} events. This is largely attributable to the fewer $FN_{\text{misclassified}}$ for Spyking Circus. The S_{CP} for Spyking Circus worsens only slightly with increasing overlaps. Interestingly, the `units_ratio` increases from approx. 1.5 to approx. 2.0, with increasing values of a, suggesting that the algorithm splits up highly overlapping units to create new ones. MountainSort performs very poorly, owing to the large number of FN events, and it's performance is not affected by the overlap fraction.

6.3. Overview

The spike sorting algorithms we analysed are able to successfully sort our ground truth dataset over a range of parameters, showing a large variability in performance across parameter sets (Figures 6.2 and 6.4). We determine optimal parameter sets (Tables 6.2 and 6.4) for each algorithm by comparing their performance to ground truth and study the results from these parameter sets in detail. Overall, Spyking Circus exhibits a preference for FP errors, while MountainSort is more tolerant towards FN errors.



(a) Effect of varying $\Delta\lambda$ **(b)** Effect of varying ϕ

Figure 6.16.: Spike sorting performance of MountainSort and Spyking Circus on surrogate data averaged over 200 realisations, while (a) varying $\Delta\lambda$ and (b) varying ϕ. The error bars denote the standard deviations around the mean. An increase in the $\Delta\lambda$ value should improve the spike sorting performance, but this is seen only for Spyking Circus (decreased S_{CP}). Increase in the overlap fraction ϕ causes a steady decrease in the performance for both algorithms, suggesting that neither of the algorithms is optimized to detect overlapping units.

With optimal parameters, Spyking Circus shows a large variation across electrodes, with exceptional performance on certain electrodes and very poor performance on others. It is able to detect overlapping units to some extent and is more sensitive to low-amplitude GTUs. It also, however, has a tendency to produce more noisy units (Figure 6.9). MountainSort is more consistent in its performance across electrodes, however, it performs exceptionally well only on 2 electrodes. It typically cannot isolate smaller amplitude units in the presence of larger ones, but yields less noisy units (Figure 6.9). On average, however, they exhibit a similar performance.

The performance of both algorithms is markedly different on surrogate datasets. Spyking Circus outperforms Mountainsort in all respects, showing no sensitivity to waveform shape and a good retrieval of single units. The performance for both algorithms worsens in more difficult scenarios (with multiple units and with overlapping waveforms), however, Spyking Circus always performs better. Appendix C contains overview figures of selected electrodes from one realisation of surrogate data which shed more light on this matter. This is also discussed further in Chapter 7.

Overall, we do not find a clear advantage for either algorithm. Both algorithms have their drawbacks and advantages, and the choice of the spike sorting algorithm should depend on the requirements of downstream analyses.

7. Summary and Discussion

7.1. Summary

In this thesis, we devloped an analysis framework and extensible pipelines to compare spike sorting algorithms to ground truth, and used them to compare two spike sorting algorithms – MountainSort and Spyking Circus. Two different kinds of ground truth were used for the comparison. One was obtained at the end of a careful manual spike sorting performed by a human expert. The second was synthetically generated surrogate data, for which we used elements of real data.

Both spike sorting algorithms rely on multiple spike sorting parameters (described in Chapter 2). We determined optimal parameter sets for each algorithm by carrying out large parameter scans for which we made use of automated pipelines, which are described in Chapter 4. We then evaluated the performance of the algorithms with optimal parameter sets on surrogate data, and studied the effect of different data generation parameters on the quality of spike sorting. The evaluation was done using metrics, such as the classification performance score (S_{CP}), which we developed in Chapter 5.

While both algorithms were able to successfully spike sort real and surrogate datasets, the results differed considerably. Spyking Circus showed a large tolerance for FP events, whereas MountainSort preferred to have more FN events – revealing a more liberal approach for the former and a more conservative approach for the latter. MountainSort created units with higher signal-to-noise ratios than Spyking Circus, but also showed a tendency to create more units than the ground truth. Spyking Circus was able to retrive more units and events from the ground truth, but the units it created had large numbers of FP events. However, Spyking Circus performed significantly better on surrogate data than MountainSort, yielding near-perfect results with certain data generation parameters.

The choice of spike sorting algorithms for the reader should depend on the kind of analysis which the sorting results will be used for. Based on our findings, for analyses unaffected by large false positives, Spyking Circus would be ideal. For example, the unitary event analysis method to detect significant spike synchronization (Grün et al., 2002a,b) is negatively affected by FP errors (Pazienti and Grün, 2006). On the other hand, analyses requiring units with high signal-to-noise ratios and small number of false positives, MountainSort would be the right choice.

7.2. Discussion

In this thesis, we have attempted to address a growing challenge in electrophysiological data analysis – that of automatically and accurately spike sorting large datasets in rea-

sonable time. We created scalable automated spike sorting pipelines with the intent of making spike sorting a reproducible procedure. Additionally, we present an approach to scan the complex space of parameters for automatic spike sorting algorithms in order to extract sets of parameters which are optimal for a given dataset. However, as with any analysis method, this approach is not without its limitations. Over the next few pages, we discuss the limitations of our approach and also discuss details which were beyond the scope of the present study.

Performance Metrics As a part of this approach, we developed performance metrics, such as the classification performance score, S_{CP} (equation (5.15)), and the conservativeness scores, C_{RP} (equation (5.17)) and C_{FR} (equation (5.16)). This was done in an attempt to yield single scores on the basis of which one can evaluate the quality of spike sorting. They are calculated for every pair of units matched across the ground truth and spike sorting result, and summarize the precision (equation (5.1)), recall (equation (5.2)) and fallout (equation (5.3)) for a given match.

Interpretation of the Classification Performance Score The S_{CP} score reduces information stored in precision-recall and fallout-recall curves. These curves, in turn, rely on the TP, FP, FN and TN events of the corresponding matched pair. In literature, when comparing the performance of a spike sorting algorithm to ground truth, the results are typically evaluated solely on these four values (Pachitariu et al., 2016; Chung et al., 2017; Chaure et al., 2018; Yger et al., 2018). In our case, however, we decided to reduce the information stored in these values to a single score to make the comparison of multiple parameter sets tractable.

Figure 7.1 shows how the S_{CP} score is affected by and can be related to the TP, FP, FN and TN events. The interpretation of the score varies with the total positive and negative events in a certain match. For example, with an equal number of negatve and positive events, a classification with 20% FN and 20% FP errors will be scored at around 0.2. However, with 10 times more negative events than positive events, the same classification percentages will yield a score of around 0.4. This aspect of the score could be remedied by normalizing it to the number of positive and negative events in future work.

We observe that for each matched pair in our dataset, we obtain many more negative events (FP and TN) than positive events (TP and FN) Figure 6.8. This is due to the fact that we consider all events belonging to other units on the electrode as TN events. Thus, the last row of heatmaps in Figure 7.1 should used to interpret the score values we present in this thesis.

Limitations of the Classification Performance Score The S_{CP} score is calculated for each pair of matched units on a given electrode. It rewards TP and TN events and penalizes FP and FN events (by maximizing precision and recall, and minimizing fallout). However, there is no mechanism by which it penalizes excess or missing units on an electrode. As an example, consider figure 7.2a, where we show an overview figure

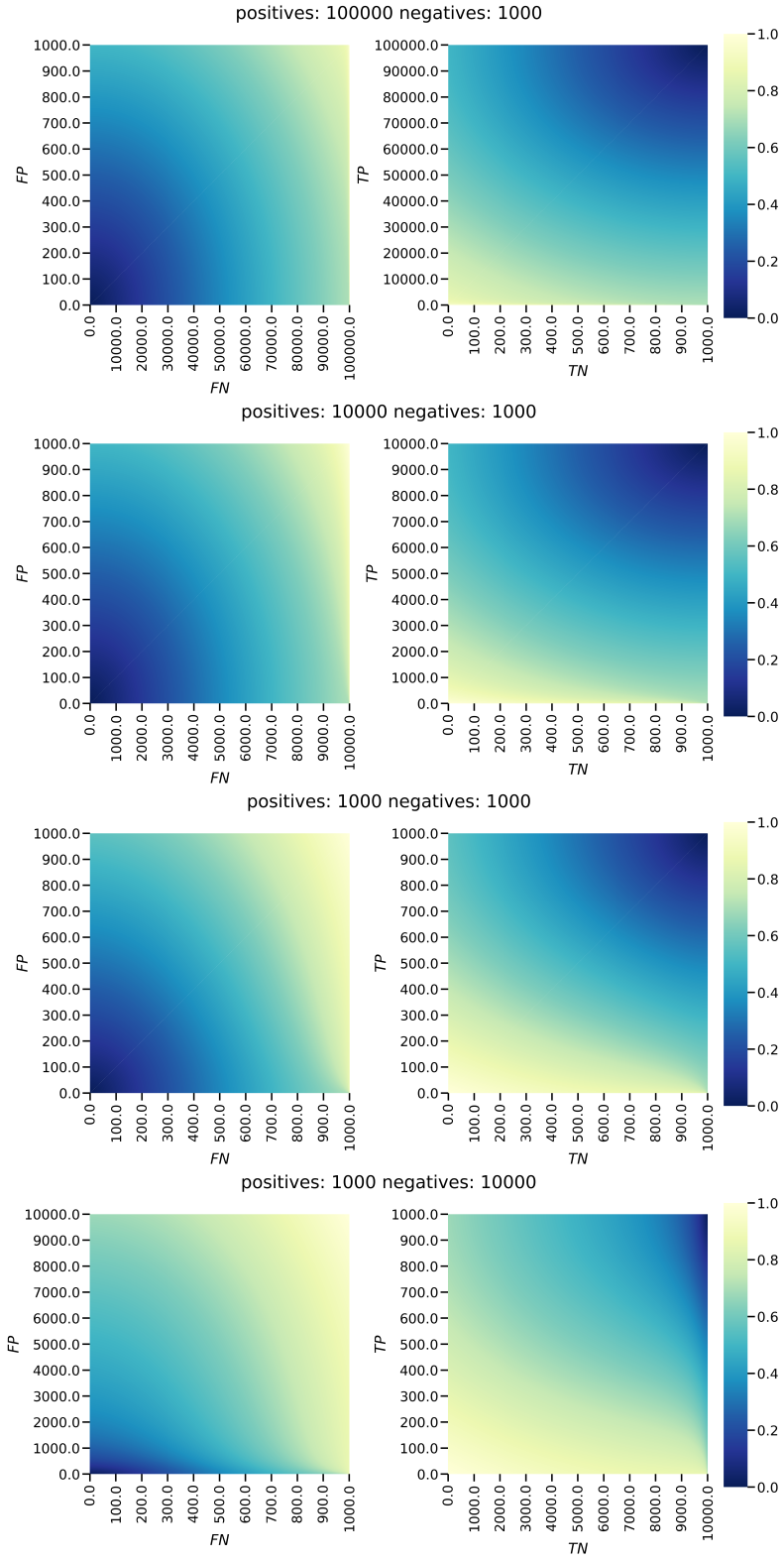
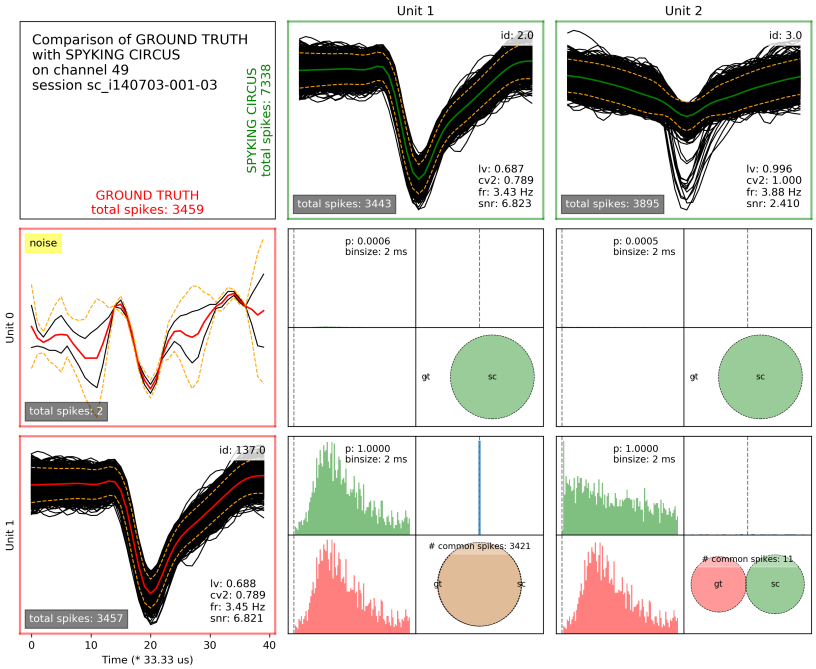
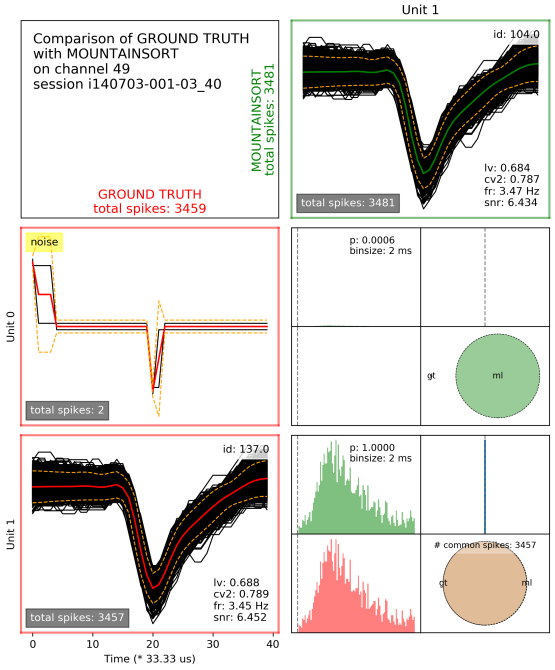


Figure 7.1: Reference heatmaps to interpret values of S_{CP}. Each row of heatmaps spans the possible values of TP, TN, FP and FN events for hypothetical classifications involving different numbers of positive events ($TP + FN$) and negative events ($FP + TN$) in the GT. Both heatmaps in each row are complementary to each other, since the number of TP events complement the number of FN events when the number of positive events is constant. Similarly, the number of TP events complement the number of FN events when the number of negative events is constant. The performance of a classification as measured by the S_{CP} score is, thus, subject to the relative number of positive and negative events.



(a) Spyking Circus



(b) MountainSort

Figure 7.2.: Overview figure for electrode 49 of surrogate data for (a) Spyking Circus and (b) MountainSort. The excess unit found on electrode 49 by Spyking Cirucs is not penalized, and instead lowers the S_{CP} score. On the same electrode, the lack of excess units in the SR leads to an increase in the S_{CP} score for MountainSort, even though it objectively performs better on the electrode than Spyking Circus.

for the comparison of ground truth to results from Spyking Circus on electrode 49. The GTU id 137 matches almost perfectly with SU id 2. However, since there is only one GTU on the electrode, the second SU (id 3) is not matched with any GTU and is thus not accounted for or penalized. In fact, the existence of the this unit adds to the number of $TN_{classified}$ events for the GTU-SU matched pair on the electrode and reduces its fallout score - thus, boosting the overall performance score of the algorithm on the electrode. In other words, on any electrode where an algorithm: 1. find more units than the ground truth, and 2. finds completely new events in the excess units the algorithm is not sufficiently penalized.

On the same electrode, the performance of MountainSort is penalized due to the lack of excess units. Consider Figure 7.2b, where GTU id 137 matches nearly perfectly with SU id 104. However, the S_{CP} score for the matched pair is close to 0.5 (compare this to the value of almost 0.0 for Spyking Circus in Figure 6.8). This occurs because the S_{CP} score for each matched pair relies on its fallout score (defined as $F = {}^{FP}/_{FP+TN}$), which in turn depends on the the number of FP and TN events. In the example above, the number of TN events is 2 (owing to the 2 events in the GTN that the algorithm successfully rejected) and the number of FP events is 24 (which are the excess events in the SU which do not exist in the GTU – FP_{new}). This yields a relatively large fallout score of 0.92 which greatly reduces the S_{CP} score.

These errors could be rectified by excluding all new events when calculating performance metrics (i.e. no TN_{new} and FP_{new} subcategories). This would, however, fail to reward the algorithm for correctly classifiying detected events into SUs which match well with GTUs. We instead provide the `units_ratio` metrics, which can be used to flag parameter sets which perform well but create many units.

Variability in the size of GTN units The noise clusters on each electrode have a large variability in the number of events they contain. Since these units are included in the calculation of precision (FP_{noise}) and fallout (FP_{noise} and TN_{noise}) for all matched pairs on an electrode, the performance scores are affected by the size of the GTN on each electrode. While not incorrect, this introduces a variability which depends on the number of events that were classified as noise in the ground truth.

Evaluation of spike sorting independent of the ground truth Throughout this thesis, we rely on the existence of a ground truth to evaluate the performance of the chosen algorithms. Ground truth is, however, not usually available for a typical electrophysiological experiment due to reasons of practicality. There has been recent work (Schmitzer-Torbert et al., 2005; Joshua et al., 2007; Neymotin et al., 2011; Barnett et al., 2016) on developing metrics which evaluate the quality of spike sorting in the absence of ground truth, making use of metrics such as the isolation score and estimating the number of FP and FN events based on the spike sorted data. Such an approach circumvents the drawbacks of using manual spike sorting or surrogate data as ground truth, discussed below. Therefore, a natural extension to the pipelines presented in Chapter 4 is to include these measures in the evaluation process.

Determination of matched pairs of units To determine matched pairs of units on a given electrode, we use the f_{1_0}-score for each pair of units. This measure ignores the TN events and all events belonging to the GTN on the electrode. Additionally, it doubly rewards the number of TP events (owing to the coefficient of 2 for the TP events in equation (5.11)). We chose this to remove the effect of large numbers of TN events present on electrodes. There is no standard measure used in literature for such comparisons and the choices include recall, f_1-score and accuracy. Some articles (Chung et al., 2017) also use confusion matrices to perform unit-to-unit comparisons, however, these are useful when comparing individual spike sorting results and become impractical when performing multiple comparisons to determine optimal parameter sets.

Comparison of spike times The spike times of events in our ground truth were flagged at the times of threshold crossing, whereas the spike times obtained from the two spike sorting algorithms were times of waveform minima. This difference in the spike times required us to align the spikes from the sorting results. We have described the approach we take to align these spike times in 5.1.2.1. While this approach works for non-overlapping units, it could cause a mis-assignment of spike times when the spike waveforms are overlapping (Figure 7.3). This is likely to happen when the larger of the overlapping waveforms has its waveform minimum after the minimum of the first. In such a case, the ground truth spike time of the first spike may get aligned to the waveform minimum of the second spike. It is important to note, however, that this would occur only in cases where overlapping spikes have been detected in the ground truth, as is the case with our surrogate data. The simplest way to mitigate this error would be to shorten the window in which we search for the minimum of the spike. The window should, however, not be set too small, otherwise ground truth spikes which actually correspond to certain sorted spikes will not be aligned.

Ground truth data We use two kinds of ground truth datasets to compare the results from automatic spike sorting algorithms. Neither of them, however, can be considered to be true ground truth, and we discuss the reasons here.

Manual spike sorting as ground truth As was already mentioned in Chapter 1, the result of manual spike sorting varies significantly depending on who is performing it (variability from expert to expert) and is not consistent even when the same expert performs the spike sorting at different times on the same data (Wood et al., 2004b). This variability stems primarily from human error in judgment, and cannot be eliminated. However, the scientific progress made in the past decades using electrophysiological data has always relied on such data and resulted in important findings for the field and for neuroscience at large. Thus, although not infallible, manual spike sorting is a valid approximation of the ground truth, and any spike sorting algorithm must be able to compare to the results obtained from manual spike sorting.

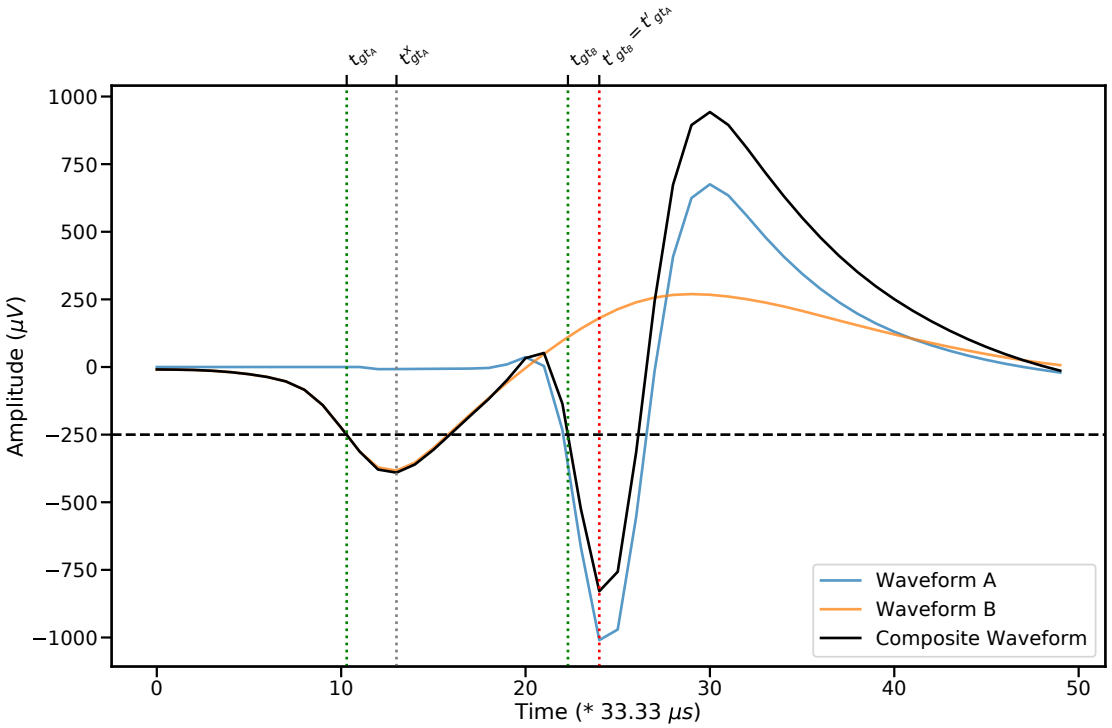


Figure 7.3.: Illustrative example of the mis-alignment of spike times caused due to overlapping waveforms. A single waveform of unit A (blue) is shown to overlap with a single waveform of unit B (orange) to create the composite waveform (black). The detect threshold for ground truth is shown by the horizontal dashed line at an amplitude of -250 μV. The intersection of the detect threshold with the composite waveform is shown indicated by the dotted green vertical lines at t_{gt_A} and t_{gt_B} for waveforms A and B, respectively. The dotted grey vertical line indicates the correct position for the corrected GT spike time for waveform A (at $t^x_{gt_A}$) and the dotted red vertical line indicates the time stamp for the actual aligned GT spike times for both waveforms (at t'_{gt_B}). Any events flagged by a spike sorting algorithm at time $t^x_{gt_A}$ would be marked as a FP_{new} event.

Surrogate data as ground truth The idea behind generating surrogate data is to have datasets where the absolute ground truth is known. For surrogate electrophysiological data, this would mean having complete knowledge of the true identities of waveforms and their corresponding spike times. Such data could be generated by simulating the electrical signals measured by an electrode, requiring accurate models of the shape of extracellular action potentials at different locations relative to a single neuron and of the superposition of such action potentials from large populations of neurons. This is achieved, under certain limitations, by the likes of Neuron [1], LFPy [2] and VisaPy (Hagen et al., 2015). However, these simulations are computationally expensive and heavily parametrised, and provide a level of detail that is not required to evaluate spike sorting algorithms. Instead, the approach we use here is often referred to as the "hybrid" approach (Carlson and Carin, 2019), wherein we use background signals from real experimental data and superimpose waveforms at known times. As Carlson and Carin (2019) aptly state in their work, "[...] synthetic data do not have to perfectly simulate a biophysical system to be useful; if a pipeline struggles with controlled synthetic data, similar struggles should be expected with real data".

However, our approach has limitations which we must point out. We obtain background signals for surrogate data by subtracting mean waveforms of units obtained at the end of spike sorting from the corresponding events in the recorded signals, so as to remove the primary component of the spike while retaining the residual noise. This heavily relies on the quality of the spike sorting performed, since subtracting the mean waveform would only be useful if the units have a relatively high SNR and contain few overlapping events. Additionally, the detection threshold used for manual spike sorting should be low enough to detect events which an automatic algorithm would find using a low detection threshold of its own. In other words, the manual spike sorting should account for as many spikes in the real data as possible, and assign them to as clean clusters as possible, in order to ensure that the resulting background signal has no events large enough for an automatic algorithm to detect. Ideally, the background signals we obtain should contain no neuronal activity (spikes) at all. However, given the limitations of manual spike sorting, there are a significant number of spikes which remain undetected or wrongly sorted in the manual process. These spikes are then detected by the automatic spike sorting algorithms and assigned to a certain cluster, and are counted as false positive events since they were not detected during manual spike sorting. This could be a potential reason for both spike sorting algorithms to have `unit_ratio` > 1.

An alternative to manually spike sorting data would be to use an automatic spike sorting algorithm with a low detection threshold, along with other parameters which yield a large number of well-defined units (which is, in essence, what we desire from the manual spike sorting). This warrants additional checks, though. For example, if algorithm A is used to remove real neuronal activity and generate background signals for a dataset which will be sorted with algorithm B, we must also test the results with the roles of the algorithsm reversed, so as to account for the effect of each algorithm at each

[1] https://github.com/neuronsimulator/nrn
[2] https://github.com/LFPy/LFPy

step.

Performance on surrogate datasets The results shown in 6.2 and Appendix C show a significant difference in the performance of Spyking Circus and MountainSort on surrogate datasets. Spyking Circus performs almost optimally when single units are present on a channel, and shows acceptable results on channels with two units – with and without overlap. The results of MountainSort, on the other hand, are practically unusable due to the large `unit_ratio` and large S_{CP} scores in almost all scenarios (Figure 6.15 and Figure 6.16). This is particularly surprising since MountainSort performed objectively better on real data (Figure 6.8). This drop in performance on surrogate datasets seems to stem from the algorithm's tendency to create more units. We suspect this is a result of strong over-clustering during the ISO-SPLIT step 2.2.3 that the algorithm is unable to correct for in subsequent steps. However, this ought to be investigated in future work.

Speed and efficiency of automatic spike sorting When motivating automatic spike sorting algorithms in Chapter 1, we described the typical manual spike sorting workflow and gave estimates for the time taken at every step of manual spike sorting. It is worthwhile stating here that both automatic spike sorting algorithms studied here were able to spike sort the data in 15-30 minutes, which is 4- to 8-fold decrease in the total time taken to sort a single dataset. MountainSort, in particular, was almost twice as fast as Spyking Circus. The algorithms were using on compute clusters with 24 CPUs per compute node, of which the algorithms used 12 CPUs per sorting task. Both algorithms were also optimized for memory usage, and the only limitation with the memory was the total size of the original dataset. The efficient parallelization of both algorithms enabled this large increase in speed and efficiency (no statistics to support these claims).

Packaging and distribution of pipelines In Chapter 4, we developed pipelines for the automatized and scalable execution of spike sorting algorithms. Each of these pipelines exists as an individual Snakemake file, along with its accompanying configuration and parameter files. There are multiple rules common to these pipelines which could be merged into a single file. Also, even though snakemake pipelines are easy to share, it would be more practical to make them available as distributable packages, possibly as Python packages. This would allow for more systematic versioning of the pipelines and easier bug reporting and troubleshooting.

The work we have presented in this thesis is by no means complete. However, we do believe it sets the groundwork for further studies which attempt to evaluate spike sorting algorithms. Our approach of using parameter scans and performance scores in combination with automated pipelines makes such analyses fast, efficient, reproducible and intuitive, and we hope future work utilizes this and improves on it.

8. Outlook

Automatic spike sorting will play a crucial role in the future of electrophysiological data analysis. With increasingly large and dense multielectrode arrays being used in experiments, it becomes imperative that algorithms and data workflows are developed that scale with the massive increase in recorded data (Einevoll et al., 2012). The work we have done in this thesis is a small step in that direction and we discuss here ideas on ways in which our work could be extended.

A good first step would be to use the developed analysis framework to analyze more datasets, especially from different experiments. The pipelines we present in this thesis are catered towards datasets obtained from microelectrode arrays with a large electrode pitch, such as the Utah Array, and there are several such datasets available in literature (Mizuseki et al., 2014; Senzai and Buzsáki, 2017). In the same spirit, the pipelines in this thesis could be generalized to be compatible with datasets of any kind. In order to accomplish this, one would have to modify only the scripts which (i) consolidate spike sorting results to a common data format for comparison, and (ii) perform the comparison between spike sorting results and ground truth. All other components of our pipelines are already generalizable to any dataset. This is facilitated by the fact the pipelines make use of the Neo data framework (Garcia et al., 2014) which provides a common data model and support for a large number of file formats for electrophysiological data.

A logical extension to our work would be to include more automatic spike sorting algorithms into the analysis. The literature is replete with different algorithms which tackle spike sorting in novel ways. Most notably, the algorithms introduced by Rossant et al. (2016) and Pachitariu et al. (2016) seem promising candidates. Each of these spike sorting algorithms, however, is designed with specific datasets in mind on which it performs optimally. It should not be expected that any one algorithm is superior to the rest in all scenarios.

To that end, we would like to highlight a recent work by Buccino et al. (2019), wherein they introduce a toolkit called SpikeInterface. It brings together many different spike sorting algorithms (including the two covered in this thesis) under one common interface, allowing the user to choose which algorithm to use for analysing a dataset. We believe this is a step in the right direction, since it removes the overhead that comes with installing and running different pieces of software, provides a unified interface for any spike sorting algorithm and greatly aids in the goal towards a reproducible spike sorting workflow. SpikeInterface accepts a variety of data formats as input, converts them into the data format required by the spike sorting algorithm of the user's choice, and returns results in a common data format. This is also the principle approach of the spike sorting pipelines developed in this thesis, and we will investigate if these developments can be coupled to the statistical analysis performed by our pipelines.

We created our pipelines with the intent of integrating them into automated data anlysis workflows being developed at the Institute of Neuroscience and Medicine – 6 (INM-6), at the Forschungszentrum Jülich. These existing workflows are designed using snakemake, allowing for a smoother integration with our pipelines. These workflows automatize various pre-processing steps for datasets recorded during long-term experiments and reduce the latency between data acquisition and analysis, allowing researchers to focus more on the analysis rather than the pre-processing. Spike sorting is a major pre-processing stage for any spike-time based analysis, requiring significant time and effort. Our pipelines will be an important addition to this workflow. Their design as Snakemake rules makes them easy to integrate in the process.

In Chapter 7, we discuss how the metrics used in this thesis rely on a ground truth and how it would be useful to introduce metrics which evaluate the sorters' performance in the absence of ground truth. Once such metrics are introduced into our analysis framework, they could be used in conjunction with our parameter scan pipeline to determine an independent set of optimal parameters for each recorded dataset. This should yield better spike sorting results than ones obtained by using a common set of parameters.

Spike sorting yields knowledge of the precise spike timing of distinct neurons in the vicinity of implanted electrodes. This precise spike timing can be used to detect correlations between behaviour and spiking activity (Vaadia et al., 1995; Riehle et al., 1997; Hatsopoulos et al., 1998), between the LFP and spiking activity (Denker et al., 2011), and even between the spiking activity of individual neurons in the form of spatio-temporal patterns (Torre et al., 2013, 2016). These correlations rely heavily on the quality of spike sorting (Pazienti and Grün, 2006). The usability of automatic spike sorting algorithms can be evaluated by looking for such correlations in automatically spike sorted data and comparing the results with those obtained from manually spike sorted data.

Automatic spike sorting pipelines could also help speed up other kinds of analyses. The shape of the waveforms of single units identified during spike sorting is stereotypical and results from a combination of the electrical properties of the extracellular medium and the distance of the neuron from the electrode (Gold et al., 2006). For datasets recorded using chronically implanted microelectrode arrays as a part of long-term experiements, there is evidence to suggest that single units can be tracked from one recording session to the next on the basis of their waveform shape and firing activity (Dickey et al., 2009; Fraser and Schwartz, 2012; Tolias et al., 2007), in effect enabling the tracking of putative neurons over the course of multiple days/weeks.

This type of a study would require multiple recorded datasets to be spike sorted. If done manually, this would be restrictively time consuming and labour intensive. However, with the help of automatic spike sorting pipelines, this procedure could be sped up significantly and even parallelized. A related challenge in single unit tracking has to do with the fact that the shape of a single unit's waveform depends on the spike sorting, and the variability inherent in manual spike sorting (Wood et al., 2004b) brings the validity of such analyses into question. If automatic spike sorting algorithms are used instead, the same spike sorting parameters could be used on all datasets being studied, thereby removing any external sources of variability. Moreover, parameter scans could be performed on multiple datasets using our pipelines to determine how spike sorting

parameters affect the results of single unit tracking.

In summary, the pipelines and metrics we have introduced in this thesis have a wide range of applications and we hope they are used and improved by anyone interested in automatizing their spike sorting workflows.

A. Interpreting Letter Plots

In Chapter 5, to present and evaluate the results of the first pass of our parameter scans, we made use of letter plots (Hofmann et al., 2011). They are graphical representation of the distribution of data samples and were introduced as an alternative to box plots and violin plots. They rely on the concept of letter values (described below). They give a more accurate representation of the tails of the underlying distribution than box plots, without making any density estimations on the samples (thus representing the true samples, unlike in violin plots).

Letter Values Given n samples $x_{(1)}, x_{(2)}, \ldots, x_{(n)}$, the k^{th} order statistic is defined as the k^{th} smallest sample. These order statistics can then be written as $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$, where $X_{(k)} = x_{(j)}$ is the k^{th} smallest sample of the distribution, $x_{(j)}$, which is at an arbitrary position j in the original distribution of samples. Of these, the median of the distribution of samples appears at the d_1^{th}-order, or at a depth of d_1, where $d_1 = \frac{(1+n)}{2}$. Letter values are order statistics which appear at defined depths in the distribution of samples, with the first letter value always being the median. Successive letter values appear at $d_i = \frac{(1+d_{i-1})}{2}$. For fractional values of d_i, the mean of the two adjacent order statistics, $X_{\lceil d_i \rceil}$ and $X_{\lfloor d_i \rfloor}$, is used. The letter values demarcate the proportion of the underlying samples contained in the tail of the distribution. For example, the second letter values demarcate 25% of the samples in the tail (*quartiles*), the third letter values demarcate 12.5% of the samples in the tail (*octiles*), and so on.

Figure A.1 shows a comparison between box plots, violin plots, strip plots and letter plots on a sample dataset describing the weight (in carats) of 53,940 diamonds for different qualities of the diamond cut. In the panel titled "Letter Plots", for each type of cut, letter values are shown as boxes of decreasing width around the median (shown by a black horizontal lines). The width of the boxes has no relevance and is used only to emphasize the difference between the letter values. The boxes are shaded (from a darker shade for smaller letter values to a lighter shade for larger ones) to represent the relative density of data points contained within those boxes. Outliers from the distribution are marked separately, just as in box plots.

The first three panels contain distributions over the same data, but using different graphical representations. Box plots reveal the same information at the centre of the distributions as the letter plots, but do not describe the tails of the distribution as well and also generate more outliers. Violin plots show a more detailed variation in the density, however, they do not represent actual data samples and the smoothness of the curves can be misleading. The white dots with black bars inside the violin plots are the same as the box plots (without the outliers marked, and the white dot representing the median). Strip plots are an almost direct representation of the underlying data and are

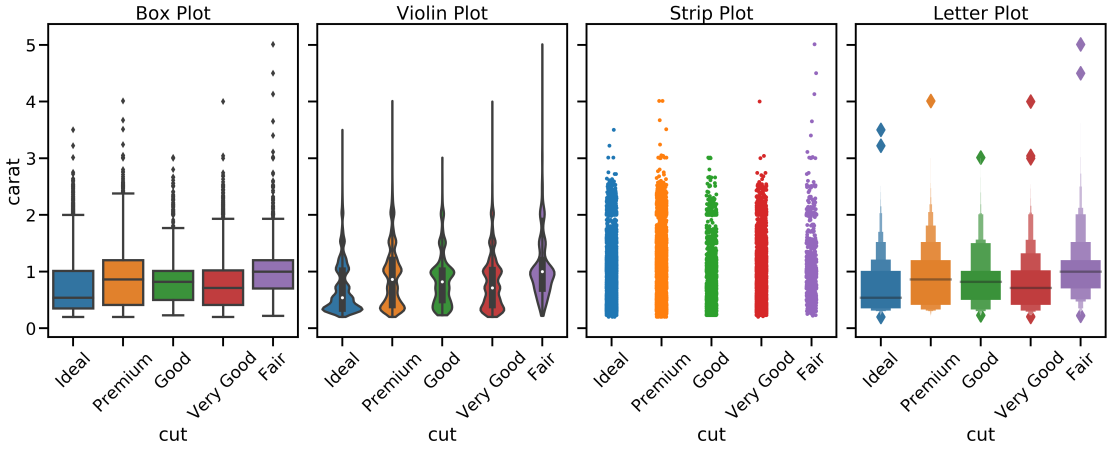


Figure A.1.: Comparison of box plots, violin plots, strip plots and letter value plots on an example dataset. The four panels of the figure show the distribution of the weight (in carats) of 53,940 diamonds for different qualities of the diamond cut ('ideal', 'premium', 'good', 'very good' and 'fair'). Each panel shows the data using a different graphical representation. From left to right, these are: box plots, violin plots, strip plots and letter values plots.

only useful when there are fewer data samples. We use strip plots for the second pass of our parameter scans.

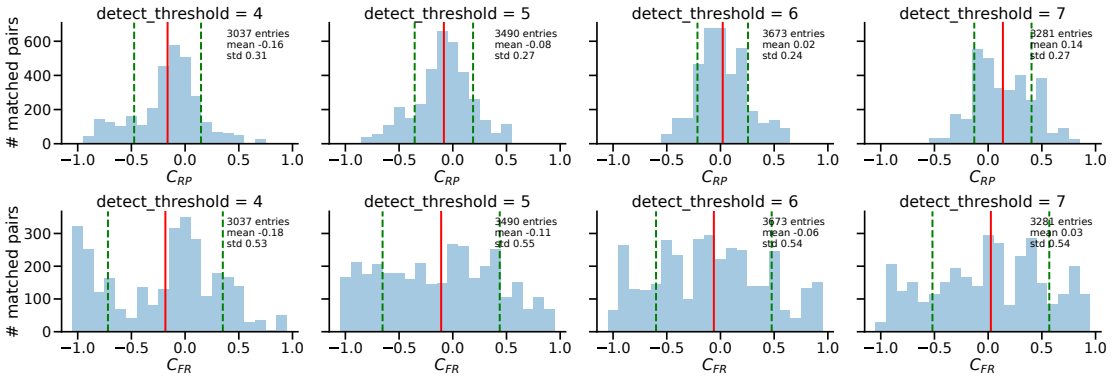
B. Distributions of Conservativeness Scores

The C_{FR} and C_{RP} scores provide an intuition regarding the conservativeness of the classification. Positive values indicate a conservative spike sorting result, whereas negative values indicate a liberal spike sorting result. The results we showed and discussed in Chapter 6 were sufficient to deduce the conservativeness of the matches and we omitted the C_{FR} and C_{RP} distributions from that chapter. Instead, we show the distributions for the second pass of the parameter scan in Figure B.1 and Figure B.2.

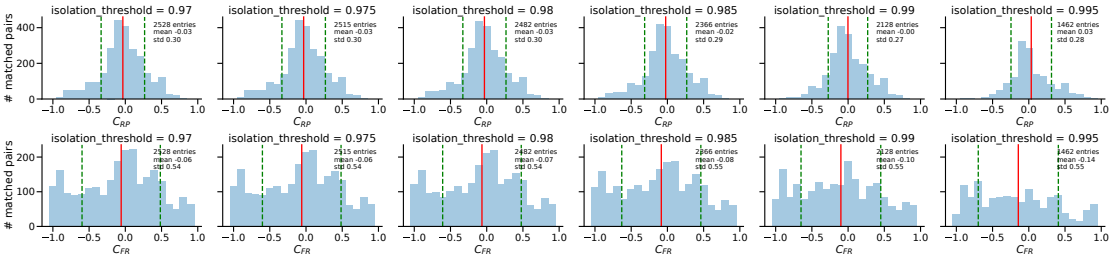
Each figure contains a subfigure for each parameter varied during the second pass of the parameter scan. In each subfigure, the top rows of plots show distributions of C_{RP} for different values of that parameter, whereas the bottom rows of plots show distributions of C_{FR}. For a given value of a spike sorting parameter, the distribution of conservativeness scores is calculated over all matched pairs which were found with that value of the parameter. This indicates how the parameter affects the overall distribution of conservativeness. A shift of the distribution to the left or the right with a change in parameter values would indicate a shift towards a more liberal or more conservative sorting, respectively. In each plot, the number of matched pairs found with the corresponding parameter value is shown, as a rough indication of the tendency of the algorithm to retrieve GTUs when that parameter value is used. The mean and the standard deviation are displayed and plotted as dashed vertical lines of red and green colour, respectively. These plots can be viewed in conjunction with Figure 6.3.

For MountainSort, an increase in the values of the `detect_threshold` parameter shifts the C_{RP} distribution to the right by a significant amount. Although a similar shift occurs for the mean values of C_{FR} distributions, the shape of the distributions do not show a discernible structure. The number of matched pairs peaks for `detect_threshold = 6`, however we choose `detect_threshold = 5` as optimal after considering other performance metrics. Increase in the values of `isolation_threshold` also cause a slight shift to the right for the C_{RP} distributions, although not significant. For higher values, though, the number of matched pairs decreases drastically, which is also visible by the size of the distribution for `isolation_threshold = 0.995`. The C_{FR} distributions do not show a consistent behaviour. A value of 0.985 affords a good balance between the conservativeness and the number of matched units. Lastly, for `noise_overlap_threshold`, we do not observe significant shifts in either distributions, although the number of matched pairs increases as you increase the parameter value. We choose `noise_overlap_threshold = 0.2` as optimal after considering other performance metrics.

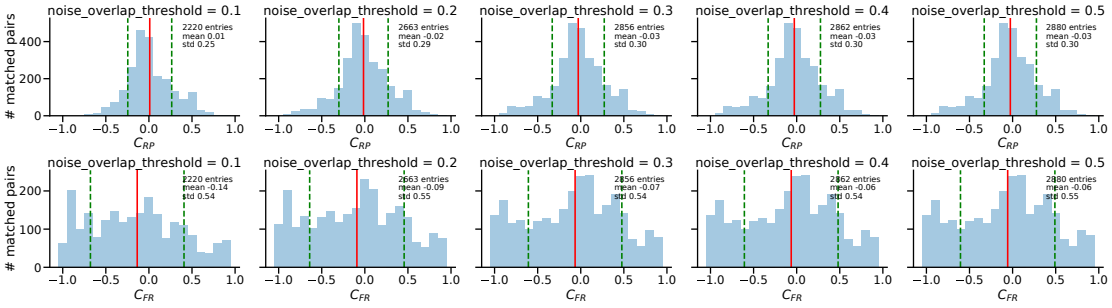
In the case of Spyking Circus, we use the same procedure of interpreting Figure B.2 to arrive at the list of optimal parameter values listed in Table 6.4.



(a) `detect_threshold`

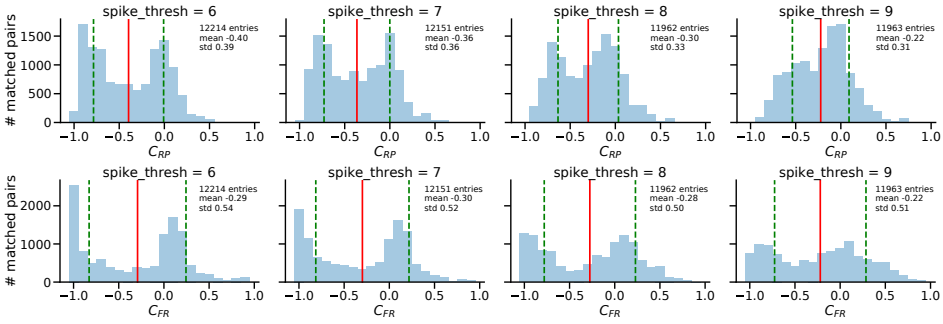


(b) `isolation_threshold`

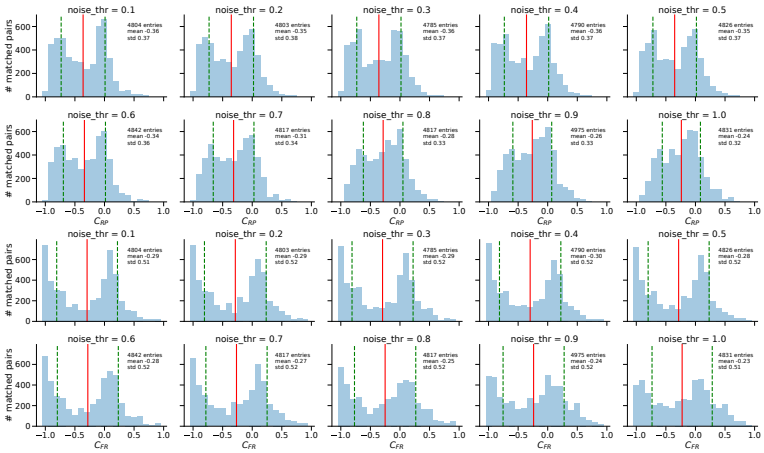


(c) `noise_overlap_threshold`

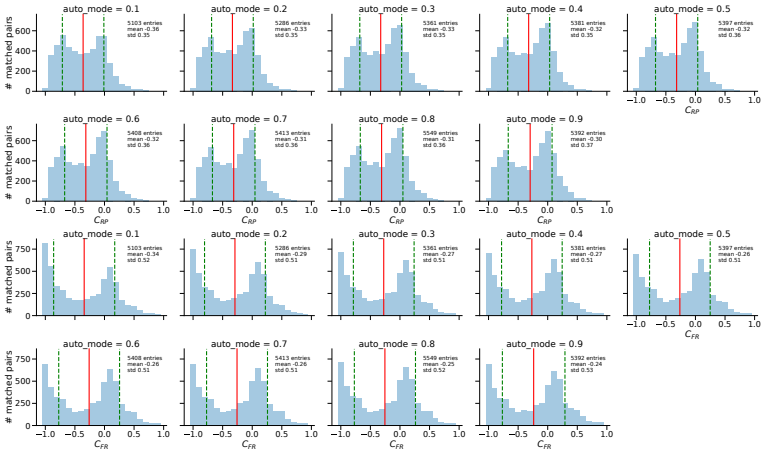
Figure B.1.: C_{RP} and C_{FR} **distributions for MountainSort parameters (a)** `detect_threshold` **(b)** `isolation_threshold` **(c)** `noise_overlap_threshold`**.**



(a) `spike_thresh`



(b) `noise_thr`

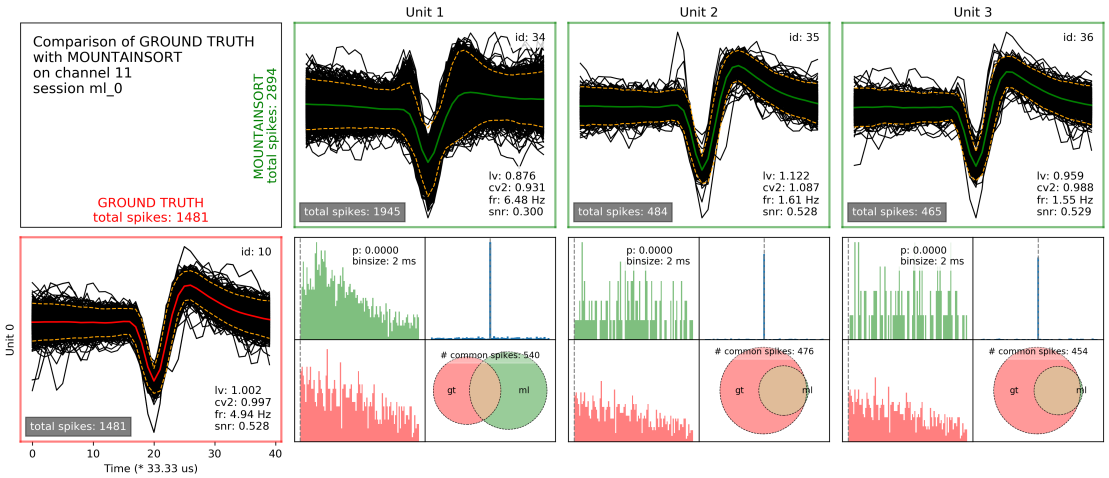


(c) `auto_mode`

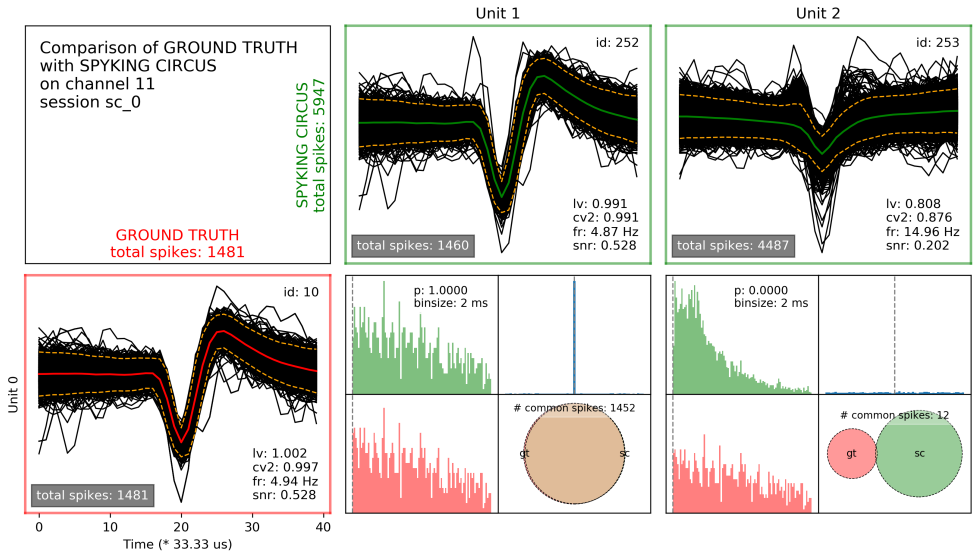
Figure B.2.: C_{RP} and C_{FR} distributions for Spyking Circus parameters (a) `spike_thresh` (b) `noise_thr` (c) `auto_mode`.

C. Overview Figures from Comparisons to Surrogate Data

In 6.2, we evaluated the performance of the two automatic spike sorting algorithms on surrogate data. We found a stark difference between their performances, with Spyking Circus performing better in every case scenario, on every performance metric. We corroborate those findings with examples of overview figures of chosen electrodes from one iteration of the analysis. Refer to figure captions for further explanations, and to Figure 6.10 to interpret the overview figures.

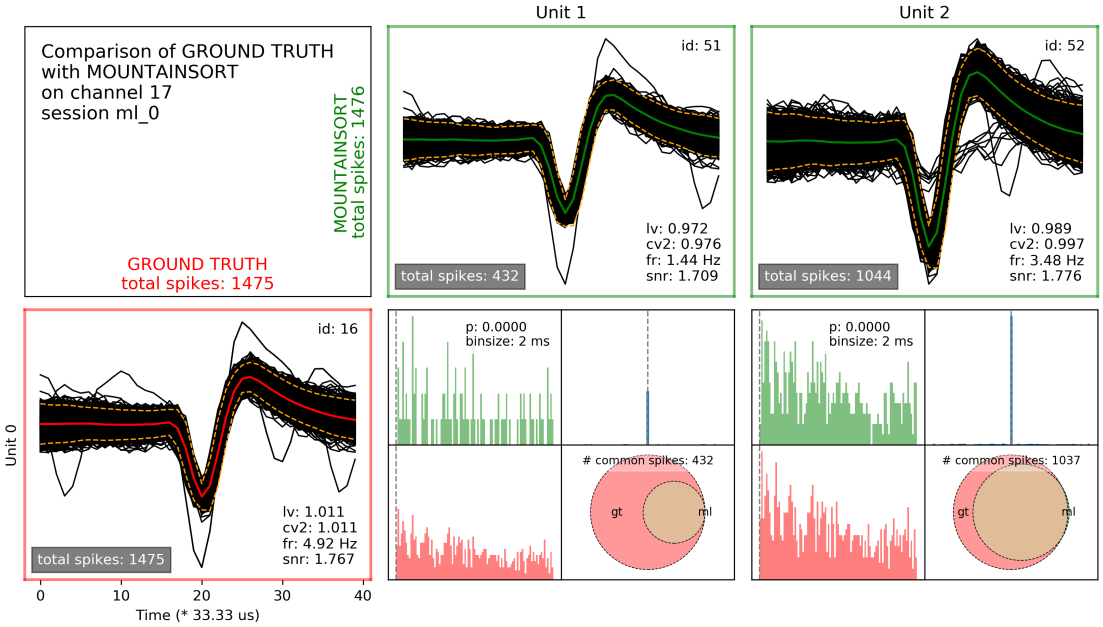


(a) MountainSort

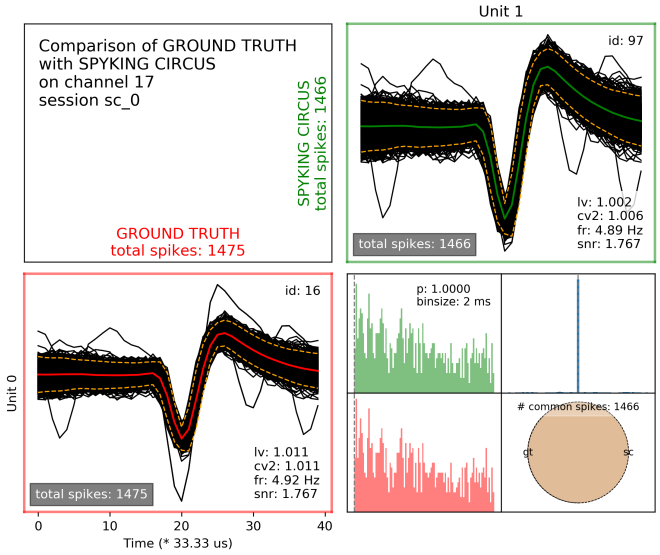


(b) Spyking Circus

Figure C.1.: Overview figure for electrode 11 of surrogate data for (a) MountainSort and (b) Spyking Circus. This relevant data generation parameters are: $\lambda = 1.0$ and $a = 8$. Mountainsort splits up the single GTU into three SUs, with one of the SUs containing many contaminating events (id 34). Due to the few FP events, SU id 35 is matched to the GTU, however, the three SUs combined were able to retrieve almost all GT events (540+476+454=1070). Spyking Circus was able to retrieve almost the entire GTU perfectly in SU id 252. The second SU (id 253) seems to contain the same events as in id 34 of MountainSort, suggesting both algorithms found the same contaminants in addition to the true surrogate events.

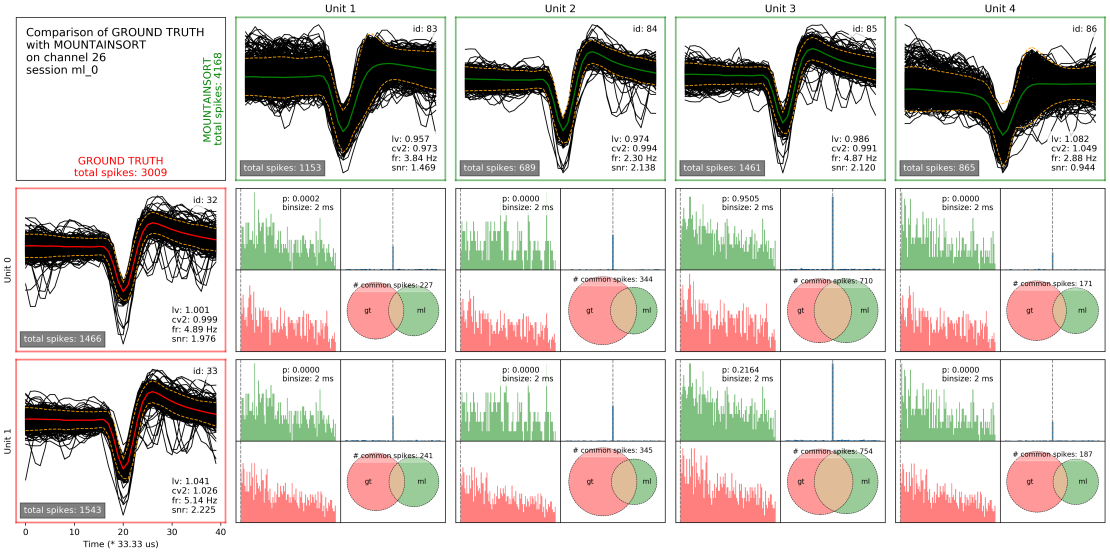


(a) MountainSort

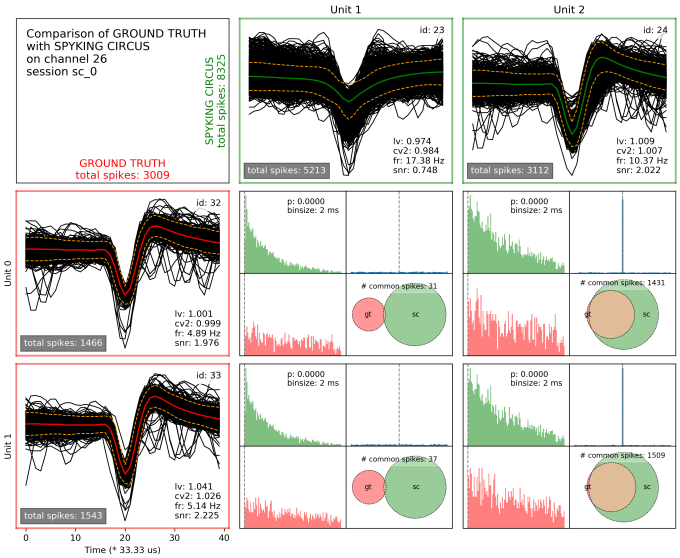


(b) Spyking Circus

Figure C.2.: Overview figures for electrode 17 of surrogate data for (a) MountainSort and (b) Spyking Circus. The relevant data generation parameters are: $a = 9$ and $\lambda = 1.0$. MountainSort splits the single GTU into two SUs, such that the sum of the overlaps from each case accounts for almost all spikes of the GTU (432+1037=1469). On the other hand, Spyking Circus isolates the entire unit almost perfectly. The surrogate GTU has a fairly large amplitude, being 9 SDs larger than the mean of the filtered signal. The waveforms amplitudes might seem different for the SU and the GTU in the case of Spyking Circus, but this is an artefact of autoscaling by the plotting package.

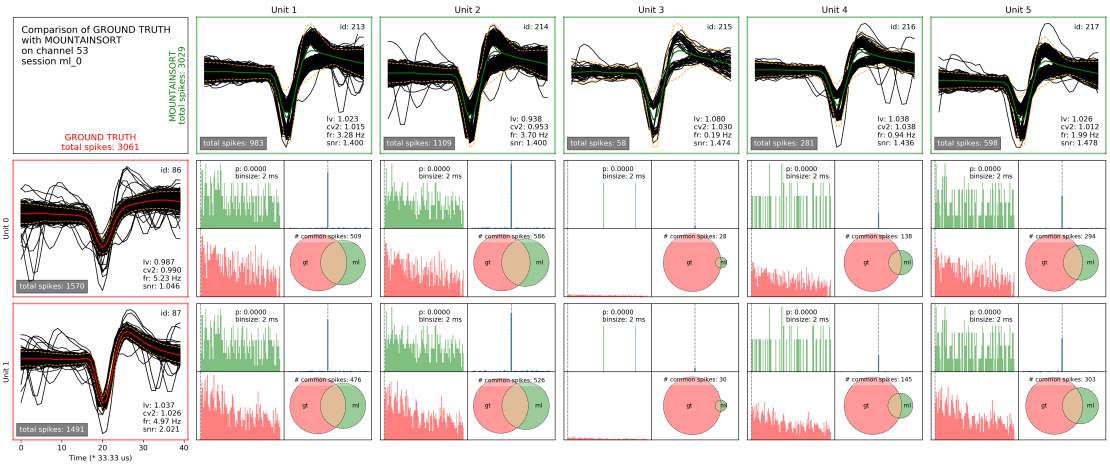


(a) MountainSort

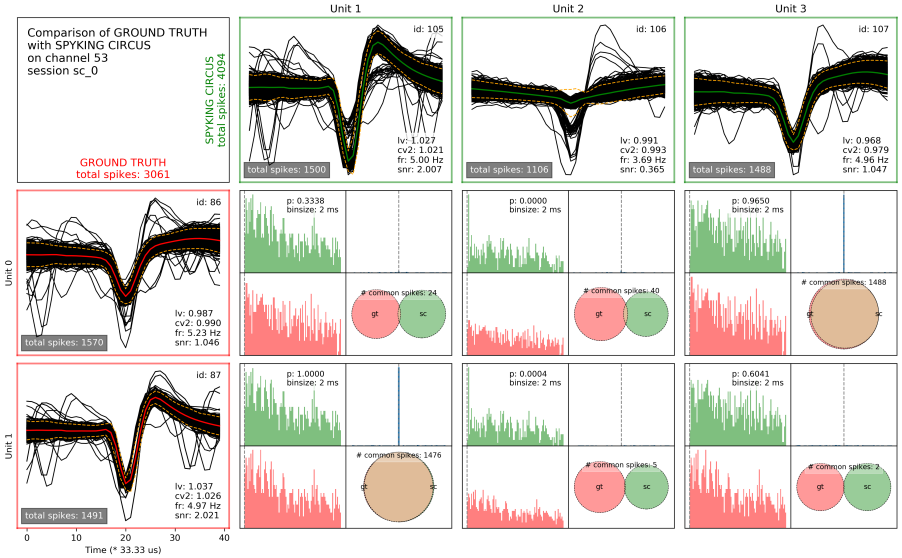


(b) Spyking Circus

Figure C.3.: Overview figures for electrode 26 of surrogate data for (a) MountainSort and (b) Spyking Circus. The relevant configuration parameters are: $\lambda_1 = 0.7$, $\lambda_2 = 0.8$, $\Delta\lambda = 0.1$, $a_1 = a_2 = 8$. The $\Delta\lambda$ for this electrode is too small for either algorithms to be able to effectively segregate the GTUs. However, both algorithms tackle this differently. MountainSort splits the elements of the two GTUs over four SUs in a seemingly arbitrary manner, mixing them with other contaminant spikes. This severely affects the S_{CP} score. Spyking Circus behaves more predictably and merges both GTUs into one large SU (id 24) and creates a separate cluster (id 23) with contaminant spikes.

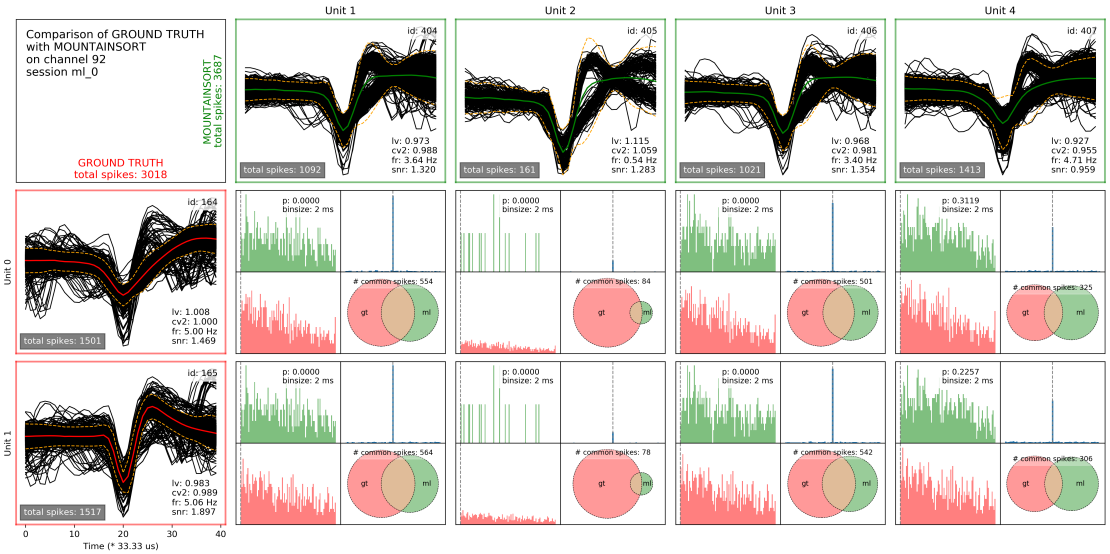


(a) MountainSort

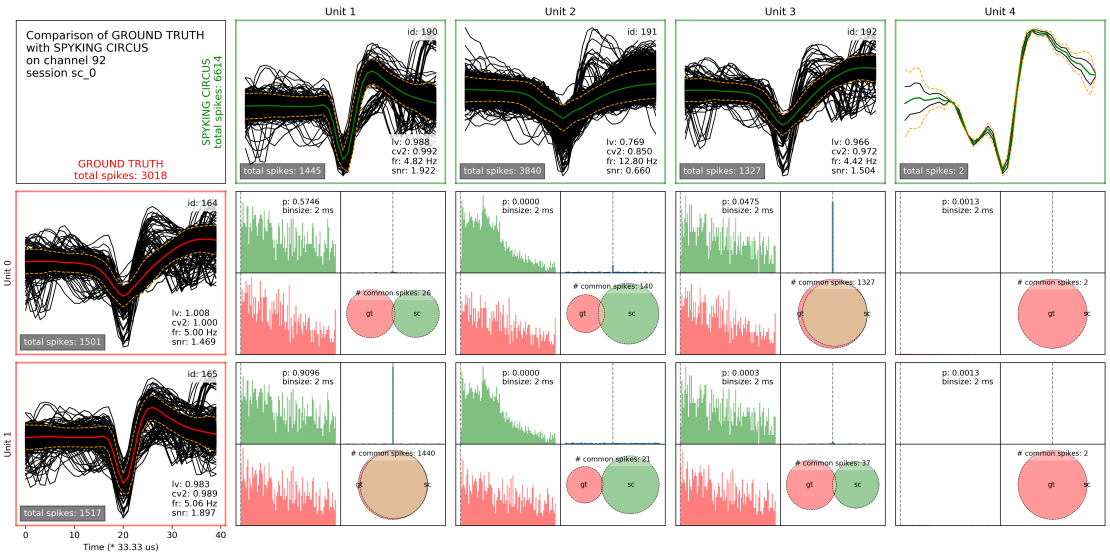


(b) Spyking Circus

Figure C.4.: Overview figures for electrode 53 of surrogate data for (a) MountainSort and (b) Spyking Circus. The relevant data generation parameters are: $\lambda_1 = 0.3$, $\lambda_2 = 0.9$, $\Delta\lambda = 0.6$ and $a_1 = a_2 = 8$. In contrast to figure C.3, this electrode has a larger $\Delta\lambda$, implying an "easier" task for sorting algorithms. MounatinSort, however, does not seem to benefit from the larger $\Delta\lambda$ and splits the two GTUs across five SUs, yielding very high S_{CP} score. This is does not seem to be an effect of contaminant spikes, since the algorithm finds fewer events in all (3029) as compared to the GT (3061). Spyking Circus achieves near-perfect performance on the same electrode, creating SU ids 105 and 107 corresponding to GTU ids 86 and 87. It also finds a contaminant unit (id 106) with minimal overlaps with GTUs.



(a) MountainSort



(b) Spyking Circus

Figure C.5.: Overview figures for electrode 92 of surrogate data for (a) MountainSort and (b) Spyking Circus. The relevant data generation parameters are: $\lambda_1 = 0.0$, $\lambda_2 = 1.0$, $a_1 = 8$, $a_2 = 10$ and $\phi = 0.7$. Note that the GTUs have 7% overlapping spikes. MountainSort split the two GTUs across four SUs and found a few contaminant spikes in the process (owing to the difference in the total number of spikes). This bad performance might not be a result of overlapping spikes, since the algorithm also performed poorly on electrodes without any overlaps (figure C.3a). Spyking Circus was able to segregate the two GTUs well, with SU ids 190 and 192 matching GTU ids 165 and 164, respectively. It found a unit of mostly contaminant spikes (id 191) and an additional cluster (unit 4) with 2 events which could potentially belong to either of the two GTUs.

Bibliography

Abeles M (1991) *Corticonics: Neural Circuits of the Cerebral Cortex* Cambridge University Press, Cambridge.

Abeles M, Goldstein MH (1977) Multispike Train Analysis. Proc. IEEE 65:762–773.

Barnett AH, Magland JF, Greengard LF (2016) Validation of neural spike sorting algorithms without ground-truth information. Journal of Neuroscience Methods 264:65–77.

Blanche TJ, Spacek MA, Hetke JF, Swindale NV (2005) Polytrodes: High-Density Silicon Electrode Arrays for Large-Scale Multiunit Recording. Journal of Neurophysiology 93:2987–3000.

Bressler SL (1995) Large-scale cortical networks and cognition.

Brochier T, Zehl L, Hao Y, Duret M, Sprenger J, Denker M, Grün S, Riehle A (2018) Massively parallel recordings in macaque motor cortex during an instructed delayed reach-to-grasp task. Scientific Data 5:180055.

Brown EN, Kaas RE, Mitra PP (2004) Multiple neural spike train data analysis: state-of-the-art and future challenges. Nat. Neurosci. 7:456–461.

Buccino AP, Hurwitz CL, Magland J, Garcia S, Siegle JH, Hurwitz R, Hennig MH (2019) SpikeInterface, a unified framework for spike sorting. bioRxiv p. 796599.

Buzsáki G (2004) Large-scale recording of neuronal ensembles. Nature neuroscience 7:446.

Carlson D, Carin L (2019) Continuing progress of spike sorting in the era of big data. Current Opinion in Neurobiology 55:90–96.

Chaure FJ, Rey HG, Quian Quiroga R (2018) A novel and fully automatic spike-sorting implementation with variable number of features. Journal of Neurophysiology 120:1859–1871.

Chung JE, Magland JF, Barnett AH, Tolosa VM, Tooker AC, Lee KY, Shah KG, Felix SH, Frank LM, Greengard LF (2017) A fully automated approach to spike sorting. Neuron 95:1381–1394.e6.

Denker M, Roux S, Lindén H, Diesmann M, Riehle A, Grün S (2011) The local field potential reflects surplus spike synchrony. Cereb. Cortex 21:2681–2695.

Dickey AS, Suminski A, Amit Y, Hatsopoulos NG (2009) Single-Unit Stability Using Chronically Implanted Multielectrode Arrays. Journal of Neurophysiology 102:1331–1339.

Einevoll GT, Franke F, Hagen E, Pouzat C, Harris KD (2012) Towards reliable spike-train recordings from thousands of neurons with multielectrodes. Curr. Opin. Neurobiol. 22:11–17.

Fawcett T (2006) An introduction to ROC analysis. Pattern Recognition Letters 27:861–874.

Fee MS, Mitra PP, Kleinfeld D (1996) Automatic sorting of multiple unit neuronal signals in the presence of anisotropic and non-Gaussian variability. J. Neurosci. Methods 69:175–88.

Franke F, Pröpper R, Alle H, Meier P, Geiger JRP, Obermayer K, Munk MHJ (2015) Spike sorting of synchronous spikes from local neuron ensembles. Journal of Neurophysiology 114:2535–2549.

Fraser GW, Schwartz AB (2012) Recording from the same neurons chronically in motor cortex. Journal of Neurophysiology 107:1970–1978.

Garcia S, Guarino D, Jaillet F, Jennings TR, Pröpper R, Rautenberg PL, Rodgers C, Sobolev A, Wachtler T, Yger P et al. (2014) Neo: an object model for handling electrophysiology data in multiple formats. Frontiers in neuroinformatics 8:10.

Garcia S, Pouzat C (2016) Tris des clous.

Gerstein GL, Clark WA (1964) Simultaneous studies of firing patterns in several neurons. Science 143:1325–1327.

Gold C, Henze DA, Koch C, Buzsáki G (2006) On the origin of the extracellular action potential waveform: a modeling study. J Neurophysiol 95:3113–3128.

Gray C, Maldonado P, Wilson M, McNaughton B (1995) Tetrodes markedly improve the reliability and yield of multiple single-unit isolation from multi-unit recordings in cat striate cortex. J. Neurosci. Methods 1–2:43–54.

Gross GW (2011) Multielectrode arrays. Scholarpedia 6:5749.

Gross G, Rieske E, Kreutzberg G, Meyer A (1977) A new fixed-array multi-microelectrode system designed for long-term monitoring of extracellular single unit neuronal activity in vitro. Neuroscience Letters 6:101–105.

Grün S, Diesmann M, Aertsen A (2002a) 'Unitary Events' in multiple single-neuron spiking activity. I. Detection and significance. Neural Comput. 14:43–80.

Grün S, Diesmann M, Aertsen A (2002b) 'Unitary Events' in multiple single-neuron spiking activity. II. Non-Stationary data. Neural Comput. 14:81–119.

Grün S, Rotter S, editors (2010) *Analysis of Parallel Spike Trains* Springer.

Hagen E, Ness TV, Khosrowshahi A, Sørensen C, Fyhn M, Hafting T, Franke F, Einevoll GT (2015) ViSAPy: A python tool for biophysics-based generation of virtual spiking activity for evaluation of spike-sorting algorithms. J. Neurosci. Meth. 245:182–204.

Harris K, Henze D, Csicsvari J, Hirase H, Buszàki G (2000) Accuracy of tetrode spike separation as determined by simultaneous intracellular and extracellular measurements. J. Neurophysiol. 84(1):401–414.

Hatsopoulos N, Ojakangas C, Paninski L, Donoghue J (1998) Information about movement direction obtained from synchronous activity of motor cortical neurons. Proc. Natl. Acad. Sci. USA 26:15706–15711.

Hatsopoulos NG, Xu Q, Amit Y (2007) Encoding of movement fragments in the motor cortex. J. Neurosci. 27:5105–5114.

He H, Garcia EA (2009) Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering 21:1263–1284.

Hemberger M, Shein-Idelson M, Pammer L, Laurent G (2019) Reliable Sequential Activation of Neural Assemblies by Single Pyramidal Cells in a Three-Layered Cortex. Neuron 104:353–369.e5.

Henze D, Borhegyi Z, Csicsvari JM A, Harris K, Buzsaki G (2000) Intracellular features predicted by extracellular recordings in the hippocampus in vivo. J. Neurophysiol. 1:390–400.

Hofmann H, Kafadar K, Wickham H (2011) Letter-value plots: Boxplots for large data Technical report, had.co.nz.

Holt GR, Softky WR, Koch C, Douglas RJ (1996) Comparison of discharge variability *in vitro* and in *in vivo* in cat visual cortex neurons. J. Neurophysiol. 75:1806–1814.

INCF Data Sharing Program (2016) Nix: Neuroscience information exchange.

Joshua M, Elias S, Levine O, Bergman H (2007) Quantifying the isolation quality of extracellularly recorded action potentials. Journal of Neuroscience Methods 163:267–282.

Köster J, Rahmann S (2012) Snakemake—a scalable bioinformatics workflow engine. Bioinformatics 28:2520–2522.

Lee J, Carlson D, Shokri H, Yao W, Goetz G, Hagen E, Batty E, Chichilnisky E, Einevoll G, Paninski L (2017) YASS: Yet another spike sorter .

Lefebvre B, Yger P, Marre O (2016) Recent progress in multi-electrode spike sorting methods. Journal of physiology, Paris 110:327–335.

Lewicki MS (1998) A review of methods for spike sorting: the detection and classification of neural action potentials. Network 9:R53–78.

Llinas R (2008) Neuron. Scholarpedia 3:1490 revision #91571.

Maynard EM, Nordhausen CT, Normann RA (1997) The Utah intracortical electrode array: A recording structure for potential brain-computer interfaces. EEG Clin. Neurophysiol. 102:228–239.

McNaughton BL, O'Keefe J, Barnes CA (1983) The stereotrode: A new technique for simultaneous isolation of several single units in the central nervous system from multiple unit records. Journal of Neuroscience Methods 8:391–397.

Mizuseki K, Diba K, Pastalkova E, Teeters J, Sirota A, Buzsáki G (2014) Neurosharing: large-scale data sets (spike, LFP) recorded from the hippocampal-entorhinal system in behaving rats pp. 1–11.

Neymotin SA, Lytton WW, Olypher AV, Fenton AA (2011) Measuring the quality of neuronal identification in ensemble recordings. Journal of Neuroscience 31:16398–16409.

Pachitariu M, Steinmetz NA, Kadir SN, Carandini M, Harris KD (2016) Fast and accurate spike sorting of high-channel count probes with KiloSort.

Pazienti A, Grün S (2006) Robustness of the significance of spike synchrony with respect to sorting errors. J. Comput. Neurosci. 21:329–342.

Pedreira C, Martinez J, Ison MJ, Quian Quiroga R (2012) How many neurons can we see with current spike sorting algorithms? Journal of Neuroscience Methods 211:58–65.

Pouzat C, Mazor O, Laurent G (2002) Using noise signature to optimize spike-sorting and to assess neuronal classification quality. J. Neurosci. Methods 122:43–57.

Powers DMW (2011) Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation 2:37–63.

Quiroga RQ, Nadasdy Z, Ben-Shaul Y (2004) Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. Neural Comput. 16:1661–87.

Rey HG, Pedreira C, Quian Quiroga R (2015) Past, present and future of spike sorting techniques. Brain Research Bulletin 119:106–117.

Riehle A, Grün S, Diesmann M, Aertsen A (1997) Spike synchronization and rate modulation differentially involved in motor cortical function. Science 278:1950–1953.

Rossant C, Kadir SN, Goodman DF, Schulman J, Hunter ML, Saleem AB, Grosmark A, Belluscio M, Denfield GH, Ecker AS, Tolias AS, Solomon S, Buzski G, Carandini M, Harris KD (2016) Spike sorting for large, dense electrode arrays. Nature Neuroscience 19:634–641.

Saito T, Rehmsmeier M (2015) The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. PLOS ONE 10:e0118432.

Schmidt EM (1984) Computer separation of multi-unit neuroelectric data: a review. J. Neurosci. 12:95–111.

Schmitzer-Torbert N, Jackson J, Henze D, Harris K, Redish AD (2005) Quantitative measures of cluster quality for use in extracellular recordings. Neuroscience 131:1–11.

Senzai Y, Buzsáki G (2017) Physiological Properties and Behavioral Correlates of Hippocampal Granule Cells and Mossy Cells. Neuron 93:691–704.e5.

Shew WL, Bellay T, Plenz D (2010) Simultaneous multi-electrode array recording and two-photon calcium imaging of neural activity. Journal of Neuroscience Methods 192:75–82.

Shinomoto S, Shima K, Tanji J (2003) Differences in spiking patterns among cortical neurons. Neural Comput. 15:2823–2842.

Shoham S, O'Connor DH, Segev R (2006) How silent is the brain: is there a "dark matter" problem in neuroscience? J. Comp. Phys. 192:777–784.

Stark E, Abeles M (2007) Predicting Movement from Multiunit Activity. Journal of Neuroscience 27:8387–8394.

Sukiban J, Voges N, Dembek TA, Pauli R, Visser-Vandewalle V, Denker M, Weber I, Timmermann L, Grün S (2019) Evaluation of spike sorting algorithms: Application to human subthalamic nucleus recordings and simulations 414:168–185.

Supèr H, Roelfsema PR (2005) Chronic multiunit recordings in behaving animals: Advantages and limitations.

Takahashi S, Anzai Y, Sakurai Y (2003a) Automatic sorting for multi-neuronal activity recorded with tetrodes in the presence of overlapping spikes. J. Neurophysiol. 89:2245–2258.

Takahashi S, Anzai Y, Sakurai Y (2003b) A new approach to spike sorting for multi-neuronal activities recorded with a tetrode–how ICA can be practical. Neurosci Res 46:265–272.

Tolias AS, Ecker AS, Siapas AG, Hoenselaar A, Keliris GA, Logothetis NK (2007) Recording Chronically From the Same Neurons in Awake, Behaving Primates. Journal of Neurophysiology 98:3780–3790.

Torre E, Picado-Muiño D, Denker M, Borgelt C, Grün S (2013) Statistical evaluation of synchronous spike patterns extracted by frequent item set mining. Frontiers in computational neuroscience 7:132.

Torre E, Quaglio P, Denker M, Brochier T, Riehle A, Grün S (2016) Synchronous spike patterns in macaque motor cortex during an instructed-delay reach-to-grasp task. Journal of Neuroscience 36:8329–8340.

Tsai D, Sawyer D, Bradd A, Yuste R, Shepard KL (2017) A very large-scale microelectrode array for cellular-resolution electrophysiology. Nature Communications 8:1802.

Vaadia E, Haalman I, Abeles M, Bergman H, Prut Y, Slovin H, Aertsen A (1995) Dynamics of neuronal interactions in monkey cortex in relation to behavioural events. Nature 373:515–518.

Wood E, Fellows M, Donoghue JR, Black MJ (2004a) Automatic spike sorting for neural decoding In *The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Vol. 2, pp. 4009–4012.

Wood F, Black MJ, Vargas-Irwin C, Fellows M, Donoghue JP (2004b) On the variability of manual spike sorting. IEEE Trans Biomed Eng 51:912–918.

Wood F, Goldwater S, Black MJ (2006) A Non-Parametric Bayesian Approach to Spike Sorting In *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 1165–1168. IEEE.

Yger P, Spampinato GL, Esposito E, Lefebvre B, Deny S, Gardella C, Stimberg M, Jetter F, Zeck G, Picaud S, Duebel J, Marre O (2018) A spike sorting toolbox for up to thousands of electrodes validated with ground truth recordings in vitro and in vivo. eLife 7:1–23.