Nano Trees:
Machine Learning Enhanced
Signal Processing of Nanopore Data

by

Deekshant Wadhwa

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the degree of
Master of Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Examining Committee

The following served on the Examining Committee for this thesis.

External Member: Dr. Richard Naud
Associate Professor
Department of Cellular and Molecular Medicine
Brain and Mind Research Institute (BMRI)
University of Ottawa

Internal Member(s): Dr. Tom Cesari
Assistant Professor
School of Electrical Engineering and Computer Science
University of Ottawa

Supervisor(s): Dr. Paula Branco
Professor, School of Electrical Engineering & Computer Science
University of Ottawa

Dr. Vincent Tabard-Cossa
Professor, Department of Physics
University of Ottawa

Declaration of Authorship

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University of Ottawa regulations concerning plagiarism, including those regarding consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

I would like to disclose that several sections of this thesis, including text, images, datasets, and results, are identical to those found in my upcoming paper [84] on the same algorithm and a patent [83]. Specifically, these sections pertain to the core methodology, experimental setup, and primary findings of my research. This overlap is intentional and is a result of the well-defined nature of my research into the development and explanation of this algorithm. Therefore, all such content is not separately referenced within this thesis.

However, this thesis also includes several unique sections that are not part of the forthcoming paper. These new sections provide additional context, in-depth analysis, and technical explanations from the perspective of a non-expert in the field of bio-science or nanoscience that further elucidate the research and its implications.

I affirm that all duplicated content is included with the full awareness and approval of the relevant parties. This ensures that my thesis maintains academic integrity and meets the standards set forth by the University of Ottawa.

Abstract

Solid-state nanopores are molecular-size holes in ultra-thin dielectric membranes used for single-molecule sensing. Nanopore sensors are operated by immersing them in a salt solution and applying an electric potential to drive the electrophoretic motion of ions and the capture of biomolecules. In a solid-state nanopore experiment, as a biomolecule passes through the nanopore, it produces fluctuations in the electrical current measured across the nanopore. These fluctuations can be used to determine various properties of the biomolecule and the nature of this translocation. As the complexity of these experiments increases, analysis of the resulting electrical signals to determine biomolecular details becomes a challenge. Piecewise constant function fitting is an important part of this analysis which is used to extract sublevel details from these signals. In an ideal world with perfect measuring devices, sublevel transitions would be instantaneous and perfectly captured without any noise in the signals. However, in practical scenarios, measurement imperfections and noise necessitate sophisticated fitting techniques to accurately interpret the sublevel transitions and deduce the underlying biomolecular properties.

Current techniques for this task perform poorly when fitting this data prompting the development of more robust and efficient methods. In this thesis, I address this challenge through a novel Nano Trees algorithm for fitting piecewise constant functions. Nano Trees leverages machine learning algorithms to provide accurate fits to the noisy piecewise constant data characteristic of nanopore signals, producing accurate fits on events as short as twice the response time of the measurement system. I analyze the performance of our algorithm on several real and synthetic datasets. These results produced using Nano Trees underscore the generalizability and accuracy of this approach in the regime of fast molecular event fitting.

Finally, I provide several approaches to understanding and fine-tuning various hyperparameters of Nano Trees. These include debugging procedures, flowcharts, isolated hyperparameter tuning instructions, default values for hyperparameters, and more. As the algorithm is novel and complex, I also discuss various automation and implementation strategies aimed at improving its practical applicability, usability, and overall effectiveness in real-world settings.

Acknowledgements

This thesis would not have been possible without the invaluable contributions of several individuals. From the initial conception during the coop to the final draft, their expertise and guidance were crucial in bringing this work to completion. I am deeply grateful to my supervisors, colleagues at the T.-Cossa Lab, and the Cyber Range Research Group, for their time, knowledge, and encouragement throughout this journey.

Thank you, Dr. Branco and Dr. Tabard-Cossa, for this opportunity and your exceptional supervision. Your insightful feedbacks and unwavering support have been instrumental in shaping this thesis. Your expertise and encouragement have not only enhanced the quality of this work but have also greatly contributed to my personal and professional growth. I am deeply thankful for the invaluable learning experience you have provided.

Thank you, Kyle, for your mentorship, your willingness to engage in numerous discussions even on my craziest ideas and your invaluable support in various aspects of my journey. Your guidance has been crucial in navigating both the challenges and opportunities I've encountered. I also extend my gratitude to my colleagues at the T.-Cossa Lab for numerous discussions, brainstorming and bridging the gap between the intricacies of nanoscience and my computer science background in this interdisciplinary research.

Finally, and most importantly, I want to express my deepest gratitude to my parents for their constant encouragement, and belief in me throughout this process. Despite the distance, their motivation and love have been a source of strength throughout this endeavour. And my brother, Deepanshu, for always being a pillar of support, offering help on everything I needed, whenever I needed it, and keeping me grounded and focused. Their combined influence and understanding have greatly contributed to the success of this thesis.

Additional Considerations

I would like to extend my gratitude to T.-Cossa Lab. I am deeply appreciative of their invaluable support, including the provisioning of software resources and data, which greatly facilitated this research.

Special thanks are due to Dr. Zachary Roelen and Philipp Mensing, who generously provided the datasets used in Section 5.2 and Section 5.3 respectively, and in several figures throughout this thesis, to illustrate various aspects related to the data and findings. Their datasets were essential for the analysis and validation of the methodologies discussed.

Additionally, Zach and Phil contributed as blind reviewers, meticulously labelling the results on their own datasets for both Nano Trees and CUSUM+ fits. Their expertise, as the original authors of the datasets, ensured the accuracy and relevance of the labels. This contributed to the precision of the corresponding interpretations, thus validating the results and enhancing the overall robustness of the findings.

Finally, I would like to thank Dr. Kyle Briggs for generating and validating the synthetic data used in Section 5.1, which was crucial for testing and validating the proposed algorithm.

Table of Contents

List of Tables

List of Figures

Abbreviations

R^2 Score Coefficient of Determination xvii, 105, 106

AdaBoost Adaptive Boosting xiv, 23, 25, 31–33, 56, 105, 114

ADEPT Adaptive Time-Series Analysis 18, 19

CUSUM Cumulative sum 6, 16–18

DBSCAN Density-Based Spatial Clustering of Applications with Noise 20

DNA Deoxyribonucleic acid xii, xiv, xvi, xviii, 7, 16, 32, 72, 77–80

dsDNA Double Stranded DNA xiv, xvi, 32, 72, 73, 77

MAE Mean Absolute Error 105

MSE Mean Squared Error xv, xvii, 54–56, 105, 106

OS Operating System 52

RC Resistance Capacitance 3, 19

RMSE Root Mean Squared Error 105

Chapter 1

Introduction

Signals with values changing with time are all around us. Amount of current flowing through a circuit, fMRI images of the brain changing in response to different stimuli [49,54], fluctuations in stock market prices [27,62,77], or changes in current flow when a biomolecule passes through a nanopore [11, 20, 37, 39]. All these signals are mere data, numbers or groups of numbers whose values change with time. Extracting information from this raw data is useful because it could provide insights not just about the data but also about the underlying processes that generated the signals.

The data we focus on in this thesis is nanopore data. The specifics of this data are discussed in Chapter 2, but simply put nanopores are tiny holes, during an experiment a constant current is maintained across it (along with some inherent noise). When a biomolecule translocates it, which means it passes through the nanopore to exit on the other side, it produces some changes in the electrical current measured across the nanopore. These changes can be detected and measured to provide information about the biomolecule.

The issue with the data is that it is not perfect, it has several distortions and inconsis-

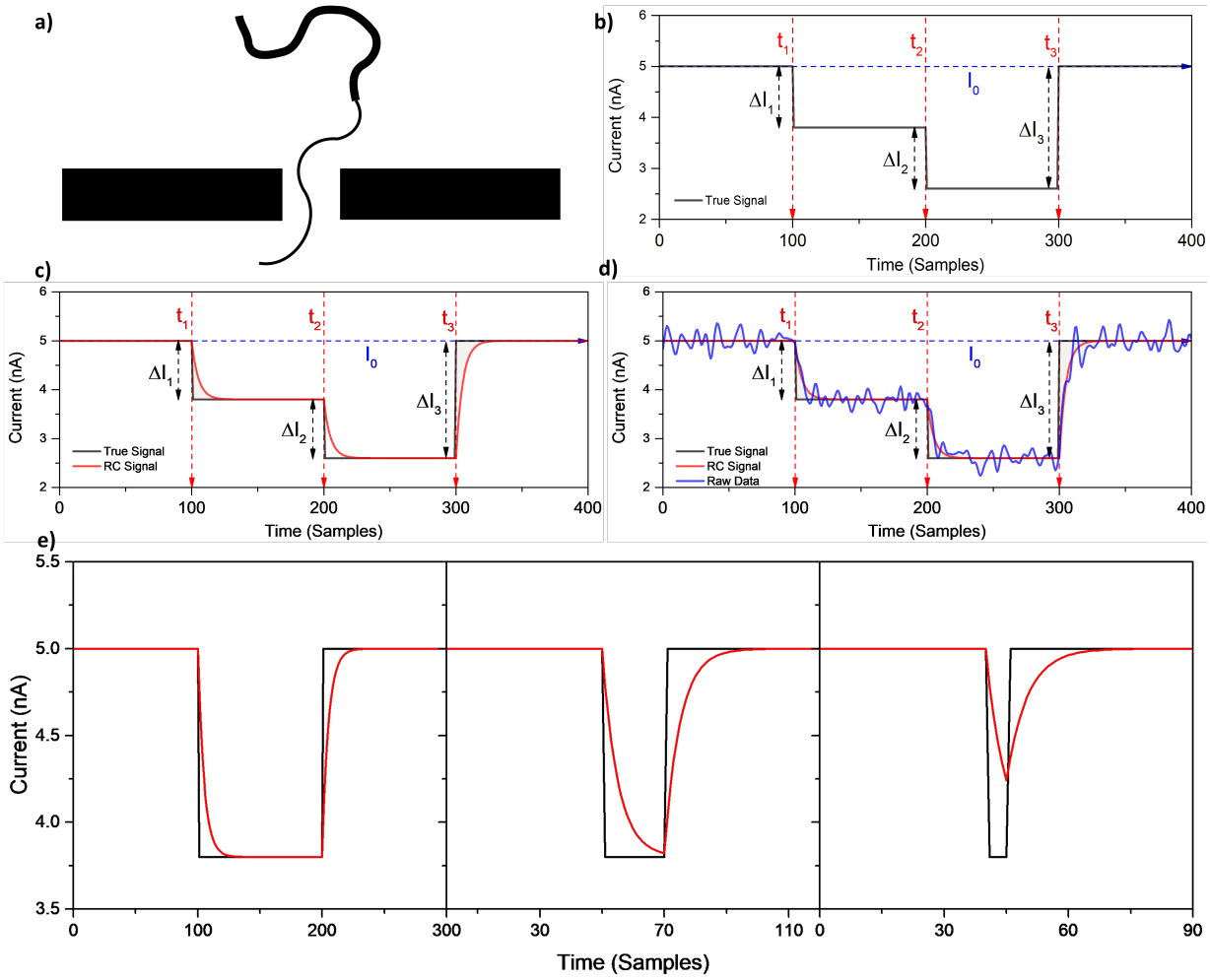

Figure 1.1: a) a schematic diagram of a molecule of varying thickness translocating a nanopore from top to bottom. b) The signal that would be produced by this molecule in the absence of noise and with infinite measurement bandwidth. c) The signal produced by this molecule considering finite bandwidth electronics, using the prediction from [11]. d) The signal that would be produced by this molecule overlaid with systemic noise. e) Examples of the level of distortion of a noiseless step function arising from bandwidth limitations as the duration of the translocation approaches the temporal resolution limits of the measurement system.

tencies. A representative example of a nanopore signal undergoing a step change between two molecular states is depicted in Figure 1.1. The imaginary molecule generating such a two-state signal is illustrated in Figure 1.1a, having its diameter increase halfway along its contour length. The true underlying signal that we hope to extract is well-approximated by a piecewise constant function as can be seen in the schematic example in Figure 1.1b. However, a truly instantaneous transition between states requires infinite bandwidth and finite response time. Bandwidth limitations imposed by measurement electronics, as well as any low-pass filtering applied, result in distortion of the signal. The case where the rise time is dictated by the Resistance Capacitance (RC) response of the measurement circuit is shown in Figure 1.1c.

The system is also subject to several sources of electrical noise that further distort the signal, and which are discussed in detail in other studies [78]. An example of the full raw signal (i.e., subject to noise, low-pass filtering, and bandwidth limitations) can be seen in Figure 1.1d. This distortion becomes especially problematic when the duration of an important feature of the signal approaches the response time of the measurement system or the bandwidth of the recording device, which causes the signal to be attenuated as shown in Figure 1.1e, and vanishing entirely into the noise for durations that are shorter than the system rise time. In simple terms, when a biomolecule or a segment of it translocates a nanopore too fast for the measuring device to accurately record the changes in current it produced then it is hard to interpret that signal recorded to extract the information of that biomolecule or that segment from this signal. These sudden changes or fast "transients" in the signal are short-lived and can sometimes get lost while interpreting the data.

Consequently, to accurately analyze nanopore data sets, the approach should first denoise the signal, then correct distortions arising from the finite bandwidth of the system, and finally evaluate the physical validity of the extracted sublevel structure. To enable

such an accurate decoding of sublevels all the way down to the bandwidth and hardware-imposed limitations of nanopore measurement, we present here a method of decoding and fitting sublevels to nanopore data, named Nano Trees. The Nano Trees framework can improve the fitting and characterization of nanopore signals that contain fast transient events, showing excellent fit accuracy down to twice the system response time. This allows for a more accurate extraction of even the short-lived transients. We also present several approaches for optimizing fitting parameters, which we anticipate will assist with the standardization of statistical analysis of complex nanopore signals across different experimental contexts.

1.1 Motivation

The primary motivation for building this algorithm was to create a unified fitting algorithm that is suitable for most nanopore data throughout all labs that work on creating and/or processing such data. There have been several attempts to build such an algorithm, as discussed in Section 3 but each method has some drawbacks. Nano Trees was built to be generalizable on any kind of data that requires piecewise constant function fitting and to overcome several drawbacks of other algorithms. We wanted the algorithm to be flexible enough to adapt to any kind of data, but be user-friendly to be easily applicable in extracting sublevels for any downstream research.

The downstream research applications were also another motivation to build this algorithm so as to have better input data for those purposes. It is hard to extract knowledge from the raw data retrieved from a nanopore experiment. We needed an algorithm to produce accurate fits to act as information from which knowledge could be easily extracted. The changes in sublevels correspond to physical attributes of dynamic biomolecules in

motion, and good fits can extrapolate several details about those biomolecules that are useful in identifying what they are, how they are folded, their interaction, and several other characteristics useful in real-world applications.

1.2 Contributions

In this thesis, we present several contributions to the field of nanopore data analysis. We introduce the Nano Trees algorithm, a novel approach designed to improve the fitting of piecewise constant functions to nanopore data. Nano Trees excels in fitting fast transients in data, a challenge that traditional methods often struggle with. Its flexibility in adapting to different datasets makes it a versatile and powerful tool for various applications in nanopore signal processing.

Our next contribution concerns the implementation of the Nano Trees algorithm using a modular design which is accomplished through the Fitting Blocks project. This approach allows individual components of the algorithm to be customized, replaced, and reused independently, facilitating easy updates and development of new algorithms. We have also created automation tools to streamline the process of tuning several hyperparameters used by Nano Trees, enhancing both the speed and accuracy of this algorithm.

Finally, as our last contribution, we will provide a comprehensive comparison of Nano Trees with the CUSUM+ method, evaluating its performance over several datasets, both real and synthetic. Our results demonstrate that Nano Trees not only surpasses CUSUM+ in capturing the underlying data structure but also excels in fitting short transients.

These contributions will be detailed in this thesis and have been documented through a research paper and a patent for the Nano Trees algorithm, underscoring their impact on

nanopore signal processing.

1.3 Organization

This thesis is organized into several chapters, each focusing on a distinct aspect of the research. We begin with a brief introduction to nanopores and signal generation using nanopores in Chapter 2. This chapter also covers piecewise constant function fitting and the challenges in fitting it on nanopore data. Chapter 3 introduces various algorithms and research in this field for fitting piecewise constant function on nanopore and other data, including Cumulative sum (CUSUM) algorithm which is one of the primary algorithms against which the results of Nano Trees are compared. The actual algorithm of Nano Trees is covered in Chapter 4 along with several automation. In Chapter 5 we introduce several synthetic and real nanopore datasets and analyze the performance of Nano Trees on them. Finally, Chapter 6 provides the main conclusions of this thesis and suggests improvements and directions for future research.

Chapter 2

Nanopores

This thesis revolves around fitting data produced by a nanopore, though the algorithm is generalizable to timeseries data in other domains as well. Nanopores are a field of study that covers topics ranging from their fabrication to applications of the technology, including DNA sequencing or "basecalling" [37], detection of biomarkers of disease in clinically relevant biofluids [39], and decoding of digital information encoded in molecular carriers [50, 97]. In this chapter, I introduce the physics of nanopores to give context for the example data used later. The chapter then discusses piecewise constant functions in detail, why they are important here, and the issues while fitting them to nanopore data.

2.1 What is a nanopore?

Nanopores are nanometer-scale holes similar in size to a single molecule of protein or DNA. Their characteristic size facilitates their use across numerous applications involving biomolecules. There are two main types of nanopores:

1. **Biological Nanopores** - These are naturally occurring protein structures present in biological membranes. Due to their atomic precision, they can be tuned to be highly specific to which molecules they allow to pass through. Some common examples of biological nanopores include Alpha-hemolysin [53, 55], Mycobacterium smegmatis porin A (MspA porin) [13] and Phi29 Connector channel [37].

2. **Solid-State Nanopores** - These are synthetic analogs of biological nanopores fabricated in a variety of materials such as Silicon Nitride [45,87,88], Aluminum Oxide [48], or Graphene [72, 91]. They are more versatile than biological nanopores in the sense that their size can easily be tuned to accommodate different targets but sacrifice native precision and specificity in exchange.

For the purpose of this thesis, only solid-state nanopores and the signals generated from solid-state nanopores are considered. However, the work done here can be generalized to any type of nanopore, or time-series data that can be described as a piecewise constant signal overlaid with noise and systemic distortion. Specifically, solid-state nanopores fabricated using dielectric breakdown, on a thin membrane of silicon nitride (SiNx) [45] are used for the experiments in this thesis, and for the purpose of this thesis, this nanopore will be referred to as simply "nanopore".

A nanopore can be thought of as a tiny hole through a thin membrane, such that it becomes the only path through which something transits the sheet, as shown in Figure 2.1. Figure 2.1 is also a representation of the pipe and sheet analogy used throughout this chapter to explain nanopores and their operation in simple terms. This analogy of sheet and pipe model consists of a sheet representing the silicon membrane on which a nanopore is fabricated, a pipe representing the nanopore itself, and a star as a placeholder for biomolecules translocating through it.

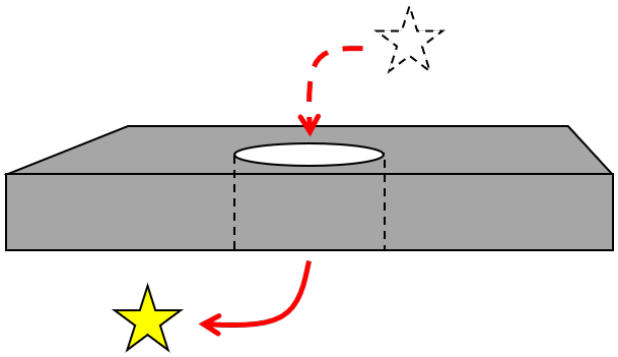

Figure 2.1: Pipe and Sheet analogy of nanopores depicting the one-sided top-to-bottom flow of any object through it.

2.2 How are signals generated from a nanopore?

When suspended in a conductive solution under an applied trans-membrane voltage, a nanopore sustains a steady ionic current, to which we refer as the baseline current. This flow of ions is transiently blocked by the translocation of a biomolecule through the nanopore. A nanopore signal then consists of a baseline current periodically interrupted by blockage events that encode information about the molecule that caused the blockage. The nanopore is first-order sensitive to the cross-sectional area of the segment of the molecule that is inside the pore at any given time.

Analogous to this, we can consider a flow of water in the sheet and pipe analogy to understand this signal generation as shown in Figure 2.2. Suppose a constant stream of water is passed through the pipe. In that case, the amount of water coming out of the other end can be considered analogous to the current flow through a nanopore when no biomolecule is translocating through it. Similarly, we can imagine passing an object along with the water flow through the pipe, that object will obstruct the flow of water thus varying the amount of water exiting the other end. This fluctuation in water flow is similar

to the fluctuations/signals a biomolecule produces while translocating the nanopore.

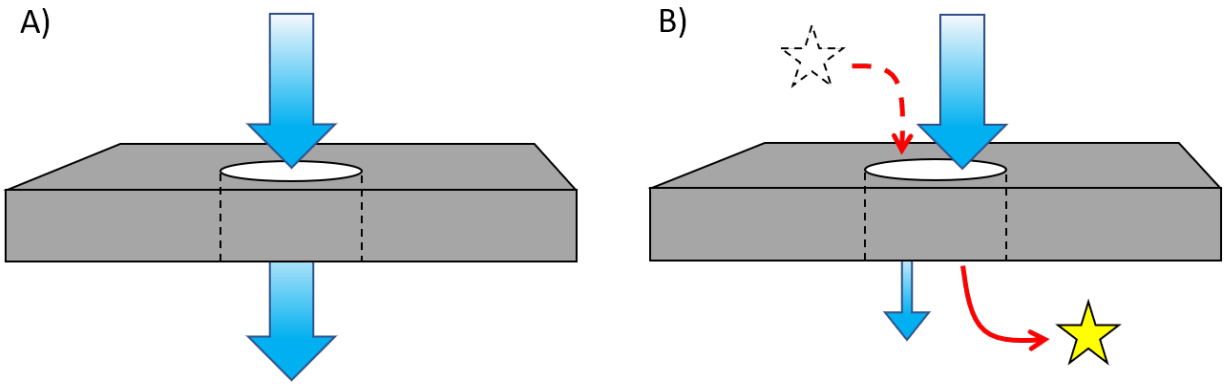

Figure 2.2: a) Pipe and sheet analogy for water flow. b) Pipe and sheet analogy for water flow along with an object.

2.3 Signal Processing

The depth of the fluctuations in current resulting from a biomolecule translocating through a nanopore correlates with the biomolecule's cross-section within and near the nanopore. By studying the duration and depth of the blockage, as well as the patterns in the current blockage signature, one can determine the physical properties of that molecule such as size, shape, orientation, folding, and branching [12, 75, 81, 85, 86, 94]. An example of the expected signal generation from a molecule that generates multiple different blockage states due to its conformation is shown in Figure 2.3 using a dummy biomolecule, nanopore, and a corresponding dummy signal.

Similarly, in the pipe and sheet model, this can be understood as how the water flow changes when objects of different sizes and shapes move through the pipes. Smaller objects allow more water to flow with them, while larger objects restrict the flow more due to their

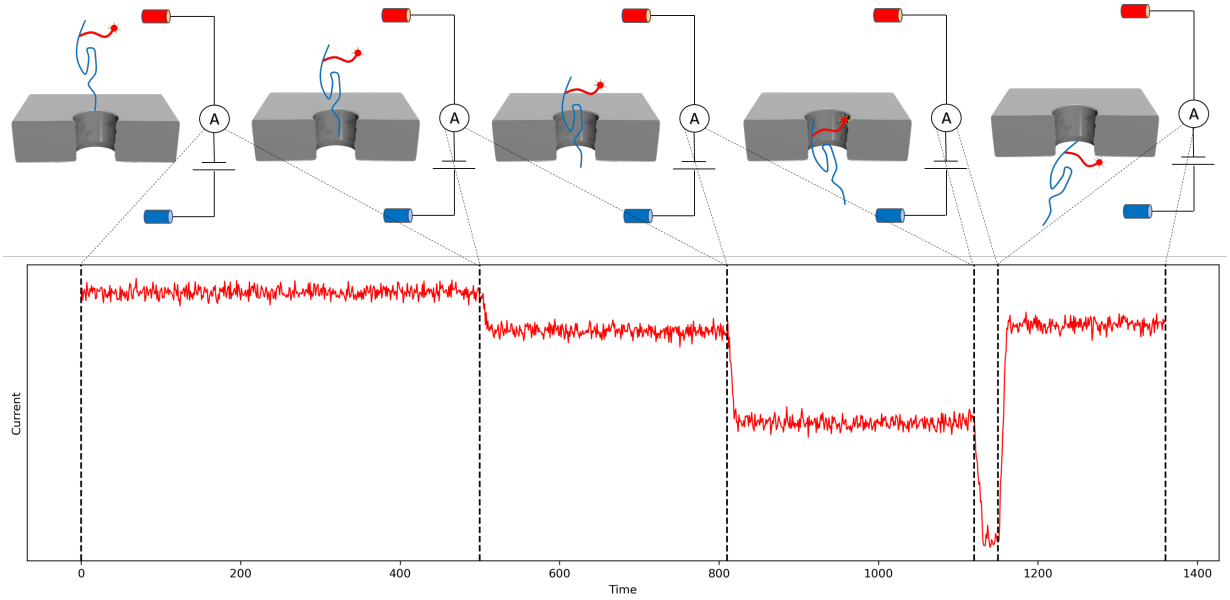

Figure 2.3: Pipe and Sheet analogy of nanopores depicting the one-sided top-to-bottom flow of any object through it.

size. Other factors like hydrodynamics, thermodynamics, pipe surface friction, material properties, and several other factors will contribute to the amount of water exiting on the other end at any instant, while an arbitrary object passes through the pipe. This creates a varying signal corresponding to the object flow and the amount of water exiting, with noise from various sources overlaying it, analogous to the signals generated by an actual biomolecule translocating an actual nanopore.

2.3.1 Piecewise Constant Function Fitting

The method used to extract relevant information about the biomolecule that caused the blockage is to fit a piecewise constant function to the data. A piecewise function $f(x)$ having n intervals, with a value of $a_1, a_2, ..., a_n$ can be defined as shown in Equation 2.1.

$$f(x) = \begin{cases} a_1 & \text{if } x > 0 \text{ and } x <= x_1 \\ a_2 & \text{if } x > x_1 \text{ and } x <= x_2 \\ ... & \\ a_n & \text{if } x > x_{n-1} \text{ and } x <= x_n \end{cases} \tag{2.1}$$

The position of the change points and the mean current value over that interval between them are the two key pieces of information that we can extract from a good fit. The position of the change points corresponds to the position of the starting and ending of sublevels and the value over that interval corresponds to the average residual current in that time duration. A typical piecewise constant function fit $f(i)$ on a nanopore data can hence be represented as shown in Equation 2.2.

$$f(i) = \begin{cases} i_1 \text{ or Baseline current} & \text{if } t > 0 \text{ and } t <= t_1 \\ i_2 \text{ or Sublevel 1 current} & \text{if } t > t_1 \text{ and } t <= t_2 \\ ... & \\ i_{n-1} \text{ or Last sublevel current} & \text{if } t > t_{n-2} \text{ and } t <= t_{n-1} \\ i_n \text{ or Baseline current} & \text{if } t > t_{n-1} \text{ and } t <= t_n \end{cases} \tag{2.2}$$

Here, the piecewise constant function $f(i)$ has n distinct interval over a range of 0 to t_{n-1}. The interval change occurs at $t_1, t_2, ..., t_{n-1}$ with the corresponding sublevel current of $i_1, i_2, ..., i_n$. The fit begins and ends with a baseline current, and has $n - 2$ distinct sublevels.

2.3.2 Issues in Fitting

Fitting a piecewise constant function to nanopore signals presents inherent complexities due to various factors. The primary difficulty lies in the undetermined nature of:

1. The number of discrete sublevels, or pieces, that should be present in any signal is unknown.

2. The background noise in the signal does not completely follow any commonly recognized statistical distribution (e.g., Gaussian, Poisson, or Uniform distribution).

3. The ground truth is mostly unknown, as the signals can in several cases become quite obfuscated.

4. The complex details of the additive noise sources arising from the interaction of the membrane material with the measurement electronics, capacitive coupling between the support structure and the input voltage, thermal noise, and 1/f-type noise arising from the discrete nature of the ions that make up the current signal [76, 79].

5. Shape of the transition between sublevels in nanopore signals which are not truly piecewise constant, instead experiencing systemic distortion due to the finite bandwidth of the measurement electronics [28] as shown in Figure 1.1c.

These first four challenges can mostly be addressed at the software level. Nano Trees does so by using Decision Trees for noise reduction and building sublevels on the go based on the user's hyperparameter choice. The last issue, however, requires more subtle consideration that takes into account the measurement context.

In an ideal situation, if the device recording the current took measurements continuously and had the ability to record instantaneous responses (i.e. had infinite bandwidth), then

the recorded signal would have sharp step changes. However, in real-world scenarios, measurement electronics cannot respond to changes in the signal instantaneously, instead responding on a timescale dictated by the interplay between the electrical resistance of the pore and the parasitic capacitance arising from the support structure, circuit junctions, and electrode-liquid interfaces [28].

Sublevels that last for a duration less than a small multiple of the response time of the measurement system suffer even more. This is because by the time the measurement system completely responds to the change in sublevel, another sublevel change occurs and the short-lasting sublevel never gets a chance to fully stabilize as shown in Figure 1.1e. Because our algorithms assume piecewise constant signals, these non-constant transients require special attention. If there were only long-lasting sublevels present, then the task of a fitting algorithm would have been to find the start and end of sublevels, which would be easily distinguishable, followed by taking a simple average of the steady state current between them. However due to this issue, the sublevel blockage is harder to estimate, and additional work is needed to determine the average current blockage during that sublevel.

Nano Tree solves these issues to an extent by using a Height Function (discussed in Section 4.2.1) that resolves sublevel height based on the type of sublevel. Sublevels are categorized based on how long they last and the current estimation is done separately for different types of sublevels. Nano Trees also utilize a backtracking algorithm to fix sublevel positions, height adjustments to further fine-tune current estimations, and several other tricks discussed in Chapter 4 to extract a piecewise constant function to get around most aforementioned issues with fitting a nanopore signal.

2.4 Summary

In this chapter, we discussed what a nanopore is and how signals are generated. Next, we describe fitting piecewise constant signals to noisy and distorted data with a piecewise constant ground truth. We concluded this section by detailing several issues encountered during this research, highlighting the need for improved methodologies and technologies to enhance the precision and reliability of nanopore signal analysis. In the following chapter, we will review related works in the field of fitting piecewise constant functions focusing on both nanopore applications and broader contexts beyond this field.

Chapter 3

Related Work

Nanopores are used for various molecular detection applications, including DNA sequencing [22], detection of biomarkers of disease in clinically relevant biofluids [8, 20, 39, 42, 68, 70], and decoding of digital information encoded in molecular carriers [8, 16]. However, the interpretation of these complex signals is a challenge, and the methods used to analyze and decode the rich information content therein remain a significant bottleneck. The problem of fitting noisy piecewise constant data is ubiquitous, appearing in many scientific fields aside from nanopore analysis. For example, the same problem appears in the analysis of anomalous network traffic [14, 74] and neuronal activity patterns [43, 80], to name just a few examples. Effective techniques to analyze this type of signal have the potential to be generally useful beyond just nanopore science.

In this chapter, we will discuss various techniques used to fit piecewise constant functions. We begin by discussing CUSUM+ [11, 28], a step change detection algorithm for nanopore signals built using CUSUM algorithm [25, 35, 36, 56]. This is one of the most commonly used algorithms in this field and is the one against which we compare our results in

Section 5. Over the years, this algorithm has been widely used by several researchers and is the go-to algorithm at T.-Cossa labs, where this research has been conducted. Next, we discuss other research in this field of fitting piecewise constant functions on nanopore data along with general methods to perform this fitting. The problem of extracting sudden changes is not unique to the field of nanopores; various specific and general algorithms across multiple disciplines have been developed to address this task.

3.1 CUSUM+

Nanopore analysis tasks involve the categorization of signals, for example, to recognize a rare target biomolecule signature from a complex mixture. Current analysis methods struggle to classify and identify molecules due to the fast kinetics of molecular passage compared to available measurement bandwidth and to their associated inability to accurately characterize fast transient signals [11, 28]. Currently, the need for nanopore data analysis is served by a varied patchwork of techniques, many of which are specific to a single experimental context, leading to differences in statistical treatment and making quantitative comparisons between labs challenging. There is, therefore, the need for a method that can be readily adjusted to work effectively across multiple molecular targets and nanopore types, to standardize reporting of results in the field. While numerous methods have been proposed over the years [3, 4, 28, 64], only a few can fit an arbitrary number of piecewise constant sublevels away from the baseline within a noisy and band-width-limited nanopore signal, and/or categorize events by type through recognition of patterns encoded in the sublevel structure.

A commonly used class of methods involves variations on the CUSUM algorithm [25, 35, 36, 56]. This algorithm [11, 25, 28] assumes that the signal to be analyzed is a piecewise

constant signal overlaid with Gaussian-distributed noise (though it can be generalized to other noise distributions [43]), and iteratively applies a modified t-test to each new data point to determine the likelihood that the local mean has undergone a step change of known magnitude. Several researchers have tried to repurpose CUSUM for nanopores sublevel extraction, in some way or the other. They work by using a technique for determining thresholds from the current sublevel on both sides and when that threshold is crossed a new sublevel is created. These studies [1, 21, 25, 35, 36, 56, 57, 61, 65, 90] struggle with resolving fast transients. Some attempts to address this issue by employing algorithms such as Adaptive Time-Series Analysis (ADEPT) (discussed in Section 3.2) to partially recover them to some extent, though challenges remain.

Forstater, Jacob H., et al. [28] implemented a version of CUSUM in their single-molecule analysis software focused on nanopore signal processing and called it CUSUM+ [11]. This method involves using statistical tests, such as the t-test, to compute the log-likelihood ratios for both positive and negative step changes in the signal. It then calculates these ratios for a positive or negative step change to assess the significance of detected changes, where a step change is flagged when either the positive or negative log-likelihood ratio surpasses a threshold. This threshold is dynamically determined by the software, based on several user-specified hyperparameters. These hyperparameters are tailored to the specific characteristics of the nanopore signals being analyzed and are hence difficult to set perfectly by an average user. The challenge lies in optimizing these parameters to balance sensitivity and specificity in step change detection. A poorly chosen threshold can lead to either missed detections of significant step changes or an excessive number of false positives, which can obscure meaningful results. These thresholds, being hard to tune, are also not easy to interpret. The complexity of nanopore signal characteristics and the variability in data can make it challenging to understand how changes in hyperparameters

affect detection outcomes. Moreover, these thresholds do not always correspond directly to physical or biological phenomena, which further complicates their interpretation. Instead, they are abstract metrics that are optimized based on statistical criteria rather than tangible physical attributes. Like most nanopore analysis frameworks, CUSUM+ performs poorly when fitting transients that are short compared to the response time of the measurement system [28], leading to miscalculations of sublevel duration and blockage depth when fitting very fast transients, or missing them entirely. Our work does not rely on step change likelihoods but instead iteratively finetunes the signal to look like a step signal and is hence able to fit transients smaller than that by other algorithms.

3.2 Other Research

An alternative to probabilistic sublevel fitting is to use an approximation to the transfer function of the measurement system to extract the underlying signal from the distorted measurement. The ADEPT algorithm [3, 4, 11, 28] fits a linear sum of exponential step functions over the event to determine the position of individual sublevels using standard nonlinear fitting techniques. This algorithm works under the assumption that the nanopore operates as a simple RC-equivalent circuit and that the rate-limiting factor that dominates the signal distortion is the response time of the measurement electronics, an assumption that breaks down when the signal is subjected to heavy filtering that can result in the time-response of the filter dominating that of the RC response. While quite effective for events with a relatively small number of well-separated sublevels (typically just one or two), the use of nonlinear fitting over many parameters (3 independent parameters for each sublevel) means that the algorithm performs increasingly poorly as events get more complex and often suffers from difficult-to-debug numerical errors that result in rejection

of valid events. Moreover, because an estimate of the number of sublevels needed for fitting is a required input to the algorithm, the same challenges with respect to heavily distorted sublevels exist. This approach was extended using more general functional forms [22].

Some other notable methods include using a stochastic model to determine sublevel changes. Markov chains are used in several studies [73,95,96] for the recognition of sublevels in multi-level current blockage events. Bandara, YM Nuwan DY, et al. [5] use Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [26] to get an initial guess of the number of sublevels, detect abrupt changes, and then iteratively check against their adjacent levels to see if they are sufficiently apart compared to a user-defined threshold, otherwise merged.

The family of piecewise functions [23, 47, 59, 60, 60, 66, 82, 89] is vast and has several applications [7, 31, 34, 44, 51, 67, 92]. The specific kind of piecewise function, which is the piecewise constant function (Section 2.3.1) is a general mathematical function that has applications outside the field of nanopores as well, for example, in the analysis of abnormal network traffic [14, 74], neuronal activity pattern recognition [43, 80], active fault detection [18], and stability analysis of neural networks [93], to name just a few examples. There has also been a lot of research in piecewise constant function fitting, in general, [6, 9, 15, 19, 33, 40, 41, 46] but they have several problems of their own, like pre-defining the number of pieces beforehand, to general to handle specific use cases, not being able to handle narrow peaks and many more. Nanopore data is a bit different in itself and has several challenges besides these as well (discussed in Section 2.3.2) which requires a specific algorithm tailored to it for better fits. Hence, Nano Trees (Section 4) was developed to tackle most of these problems, as a piecewise constant function fitting algorithm for specifically fitting nanopore data, but general enough to be used outside this domain as well.

3.3 Summary

In this chapter, we discussed several techniques for fitting piecewise constant functions. Some of them are in the domain of nanopores, where they help model the varying nanopore signals into constantly defined sublevels. Others were more general and in other domains outside nanopores. One key technique we explored is the CUSUM+ algorithm and its several implementations in various research for extracting piecewise constant functions from nanopore data, and their shortcomings. In Chapter 4 we introduce the Nano Trees algorithm, a novel approach designed to address some of the shortcomings of existing techniques. We will delve into the specifics of the Nano Trees algorithm, its implementation details, and automation techniques to improve its hyperparameter tuning, setting the stage for a comprehensive understanding of its contributions to the field of nanopore analysis.

Chapter 4

Nano Trees

In this chapter, we will introduce the Nano Trees algorithm around which this entire thesis is based. Nano Trees is a sequential process, so we begin with a brief introduction about this algorithm, followed by precise details of these processes, also called passes. Section 4.1 provides a brief description of the main machine learning algorithms used for developing the Nano Trees algorithm. Section 4.2 contains details of height functions used throughout the algorithm. After that, we delve into the technical implementation of this algorithm. We will introduce a framework in which all these passes and subpasses are implemented in Section 4.3, the Fitting Blocks Project, a flexible architecture to code Nano Trees in a modular and easily editable fashion. Nano Trees depend on several hyperparameters, so to automate the retrieval of some of those and to fasten the overall process some automation techniques will be discussed in Section 4.4.

4.1 Machine Learning

The Nano Trees algorithm relies to some extent on certain machine learning algorithms and terminologies. Although these algorithms and terminologies are well-defined in the machine learning literature, this section will provide a brief overview of the specific aspects that we are using in this work. We will begin by discussing the AdaBoost algorithm [24, 29, 58] and decision trees algorithm [10, 38, 58]. Finally, we will explain the meaning of overfitting in the context of Nano Trees and how it is being used in the Nano Trees algorithm in Section 4.2.

4.1.1 Adaptive Boosting

AdaBoost [24, 29, 58] is an ensemble learning technique. It is an algorithm that learns by training several base (or weak) learners and improves itself by learning from their mistakes. Base learners or weak learners are models that perform slightly better than random chance, which can be decision trees, linear models, or any model that is only slightly better than random guessing. AdaBoost uses them as building blocks to create a strong ensemble predictive model through iterative training and re-weighting.

Over the iterations, using m weak learner it tries to reduce the sum of mean square error ϵ_k for the k^{th} weak learner. This error is used to calculate the updated weight α_k for each weak learner in the ensemble according to their individual performance depicted by ϵ_k. The process is repeated several times till the combined weighted error is reduced and the prediction $\hat{y}(x)$ is done using Equation (4.1) where $h_k(x)$ is the individual prediction of each weak learner in the ensemble.

$$\hat{y}(x) = \sum_{k=1}^{m} \alpha_k \cdot h_k(x) \tag{4.1}$$

4.1.2 Decision Trees

The decision trees algorithm [10, 38, 58] is a supervised machine learning algorithm that can be used both for classification and regression. In our work, we only focus on decision tree regressors that work by grouping similar data into sections that are represented by a single value, which is then iteratively split into smaller nodes (or sublevels in this context) until the mean-squared error arising from this local approximation is minimized. Initially, we have a single root node, consisting of the entire event. In each step, the algorithm selects a unique position in all sublevels at the lowest level of the tree, to split them where it achieves the best split as measured by minimizing the least-squared error resulting after the split. This process is repeated until the stopping criteria are met and each node's final value is calculated using the average of values encapsulated by each leaf node. These nodes are comparable to a piecewise constant approximation of the data, such that data points that are close to each other in terms of their values and time position can be well approximated by the same current value, representing a sublevel over that region.

4.1.3 Overfitting

The Nano Trees algorithm discussed in Section 4.2 uses the term "overfitting" to describe the outputs of its first two passes. This section justifies the use of this term along with the reason why it was used and how it is advantageous in achieving the results we are trying to produce through Nano Trees.

Traditionally in machine learning, overfitting refers to a situation where a machine learning model, instead of learning the true underlying patterns in the data, starts capturing noise, anomalies, or irrelevant details that do not generalize over the entire data distribution [32, 52, 63]. We can empirically observe overfitting when the model performs really well on the training data but equally poorly on the testing data based on performance metrics. It usually occurs when the training is done for too long, the model is overly complex to the point that the general patterns are not being learned, or there is insufficient data for training. Some ways to reduce overfitting are to use regularization, cross-validation, model pruning, early stopping, or getting more data that has a similar pattern that we want the model to recognize.

In Pass 1 (Section 4.2.2) and Pass 2 (Section 4.2.3) of the Nano Trees algorithm (Section 4.2) we train a model using a single time series or event at a time. Machine learning models like decision trees or AdaBoost perform well when they have multiple time series data to train on. For example, hypothetically if we were to use multiple events for the same molecule similarly translocating the nanopore, the model might be able to extract a generalized pattern in the events and possibly return sublevels without the need for any further passes. In this scenario, the model would learn the patterns in the data and ideally not memorize the data it is being trained on. In contrast to this, in our experiments, we provide the algorithm with just a single event which corresponds to just 1 biomolecule translocating the nanopore just once. It is not possible to get more data that looks similar to it because, in a nanopore experiment, we are not sure what types of events we will end up getting. Even when the user has a broad sense of what the resultant shape might look like, it is not possible to create a generalizable algorithm based on this partial information that can fit any type of event. Hence, the model overfits to learn just that single event along with several noises present in that event, which is desirable for what we are trying

to achieve in the first two passes of the algorithm.

Methods like cross-validation cannot be used in this situation as we have only one event (time series), making it challenging to create any splits, let alone meaningful 'k' splits on it. We cannot perform early stopping as we do not know when to stop, because there is no numerical metric available, that could inform the training procedure when to stop training on this single event data. Simplifying the model is also not a good way to approach this problem, because even if everything works, and a simpler model misses a real sublevel, considering it to be noise, then the fit would anyhow be useless.

To solve this problem, we decided to use tree-based regression techniques not to make the model learn the pattern from that single event but to generalize that single event into a fairly larger number of splits (or sublevels), that are guaranteed to be way more than the physical number of sublevels present in the event in all cases. Using the hyperparameters in Pass 1 and Pass 2, we prevent the algorithm from getting extremely overfitted to a point where each data point in the event corresponds to an individual leaf node in the tree. Rather we are able to stop the training way before that point, but let the training go long enough way after the point where it produces exactly the correct number of sublevels expected at the end of the entire Nano Trees procedure, ensuring that we retain all possible real sublevels along with some amount of event specific noise in the trained model. This is the reason why we use the term overfitting in Section 4.2 to describe Pass 1 (Section 4.2.2) and Pass 2 (Section 4.2.3) of the Nano Trees algorithm.

Given that this overfitting is the beginning of Nano Trees, there needs to be some way to reduce it and keep only the physical sublevels. This task is performed by all the further passes in the Nano Trees pipeline after Pass 2, as discussed in Section 4.2. It can be considered a post-pruning procedure, methodologically removing branches from the regression tree (not directly in the model) that correspond to non-physical sublevels in the

final fit.

Therefore, in this section, we justified the use of the term "overfitting" in the context of the Nano Trees algorithm (see Section 4.2). We explained the necessity of overfitting and how we achieved it. Finally, we described how we will reduce this overfitting using the later passes in the Nano Trees.

4.2 Nano Trees Algorithm

Nano Trees is a sequential process in which the data is hierarchically grouped into sublevels, beginning with an overfit of the data and iteratively merging sublevels and improving the fit through increasingly fine-grained passes over the underlying data. This process continues until it is deemed physically accurate according to a set of context-specific, user-specified hyperparameters. Figure 4.1 provides a block diagram overview of the process from signal generation to event fitting using Nano Trees.

The Nano Trees pipeline (cf. Algorithm 4.1) consists of steps that iteratively smooth and improve the overall fit by refining the estimate of the times when a significant step change occurred in the data. At each step, the event e becomes progressively smoother as physically insignificant sublevels are removed according to various criteria, specified by the user through adjustable hyperparameters. Each pass listed here represents a modular component that operates independently of the others, which can be rearranged or reapplied as needed for a particular fitting task. The selection and order of passes listed here were found to be effective for fitting the data discussed in this work but are not necessarily prescribed for all nanopore data and can easily be updated as needed.

Algorithm 4.1 Nano Trees Pipeline

Input:
E ▷ Array of events e
H ▷ Hyperparameters H.pn for pass 'n' and H.p for post processing.
Procedure:
$fits \leftarrow []$
for e **in** E **do**
Let y be the current values in e
$H.pn$ be the hyperparameters for pass n
$\hat{Y}_1 \leftarrow \text{PASS1}(y, H.p1)$ ▷ Adaboost noise stripping. See Section 4.2.2
$\hat{Y}_2 \leftarrow \text{PASS2}(\hat{Y}_1, H.p2)$ ▷ Decision tree noise stripping. See Section 4.2.3
$\hat{Y}_3 \leftarrow \text{PASS3}(\hat{Y}_2, H.p3)$ ▷ Merge small current steps. See Section 4.2.4
$\hat{Y}_4 \leftarrow \text{PASS4}(\hat{Y}_3, H.p4)$ ▷ Correct short duration sublevels. See Section 4.2.5
$\hat{Y}_5 \leftarrow \text{PASS5}(\hat{Y}_4, H.p5)$ ▷ Merge small current steps. See Section 4.2.6
$\hat{Y}_6 \leftarrow \text{PASS6}(\hat{Y}_5, H.p6)$ ▷ Clear baseline. See Section 4.2.7
$\hat{Y}_7 \leftarrow \text{PASS7}(\hat{Y}_6)$ ▷ Backtrack. See Section 4.2.8
Let p be the hyperparameters for post-processing
$\hat{Y} \leftarrow \text{POSTPROCESSING}(\hat{Y}_7, H.p)$ ▷ See Section 4.2.9
$fits \leftarrow \hat{Y}$
end for
Output: $fits$

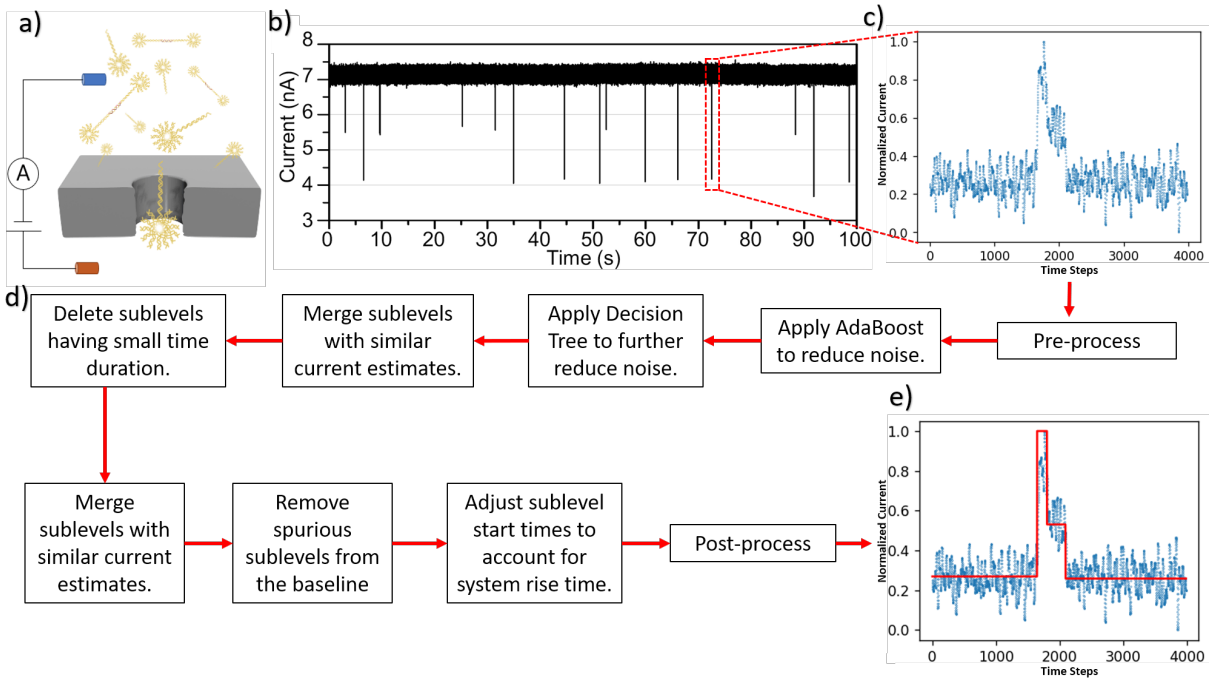

Figure 4.1: Workflow of a biomolecule translocation, signal generation, and nanopore fitting. a) Representation of a nanopore and biomolecules passing through it under the influence of an applied voltage. b) Current trace produced by sample molecules translocating a nanopore. c) An event out of several in the recorded current trace. d) Flow chart representing various passes in the Nano Trees Algorithm. e) The sampled event fitted using Nano Trees and depicting various sublevels in the event.

4.2.1 Sublevel Current Estimation Function

The Sublevel Current Estimation Function or Height Function, as used interchangeably throughout the code and manuscript, is a function used throughout the Nano Trees pipeline, used to determine the current blockage or the constant values for the sublevels or the sublevel height for a given sublevel region. At each step of Nano Trees, the sublevel structure is updated, and the current value assigned to each sublevel must be recalculated. This is done in a system-aware manner by considering how our measurement distorts the

signal.

Algorithm 4.2 l50_max_height

Input:
$shortSublevelDefinition$ ▷ Hyperparameter
$sublevel$ ▷ Array of length 'n'
$previousHeight$ ▷ Height of previous sublevel
Procedure:
$sublevelHeight \leftarrow 0$ ▷ Sublevel Height
if $n < shortSublevelDefinition$ **and** $previousHeight$ **exists then**
  $direction = \sum_{i=0}^{n-1}(sublevel[i] - previousHeight)$
  if $direction < 0$ **then**
    $sublevelHeight \leftarrow \min(sublevel)$
  else
    $sublevelHeight \leftarrow \max(sublevel)$
  end if
else
  $sublevelHeight \leftarrow \lfloor \frac{2}{n} \rfloor \sum_{i=\lfloor n/2 \rfloor}^{n-1} sublevel[i]$
end if
Output: $sublevelHeight$

The sublevel height calculation can be done in various ways. Algorithm 4.2 defines l50_max_height function which is used to calculate the sublevel heights, by default, throughout the Nano Trees pipeline unless otherwise stated. In this function, because the duration of a sublevel affects the extent to which it gets distorted, we use a hyperparameter $shortSublevelDefinition$ to split sublevels into two categories based on whether they last long enough to reach a steady state or not. For sublevels that have fewer data points than $shortSublevelDefinition$, the extreme values are used as the current estimate. This means that the current value that is furthest from the previous sublevel in absolute value is used as the estimate of the sublevel current, effectively using local noise to offset systematic underestimation arising from finite bandwidth limitations. For all the other sublevels, the average of the last 50% of the data points in the sublevel is taken as the

sublevel current.

4.2.2 Pass 1: Adaptive Boosting

The Nano Trees pipeline begins by stripping away as much noise as possible from the signal before starting to determine where the sublevels exist. To do so, AdaBoost [24, 29] is used and decision trees [10, 38] are employed as our weak learners (as discussed in Section 4.1.1 and 4.1.2). During prediction, the AdaBoost Algorithm categorizes data points that are close together in terms of values and time into a distinct group. This entire group gets assigned a single representative current value, thereby reducing any noise over this region. Employing an ensemble technique in this context mitigates the occurrence of exaggerated false subdivisions that would otherwise have been made by individual iterations of decision trees.

It should be noted that we are using the AdaBoost regression model not to predict unknown data, but to build a less noisy piecewise constant function. The process is shown in Figure 4.2, starting with the raw data in Figure 4.2a. For an event e of size n with current values represented by an array Y as shown in Equation 4.2 we apply normalization and optional filtering to get Y_1 as shown in Equation 4.3 where Y_{min} and Y_{max} are the minimum and maximum values in Y. Y_1 is the min-max normalized version of Y scaled between 0 and 1, sampled at a constant time interval. When applying AdaBoost the input is an array of indices X from 0 to the $n-1$ as shown in Equation 4.4. The target value here is Y_1 based on which AdaBoost tunes its model parameters.

$$Y = [y_0, y_1, ..., y_{n-1}] \tag{4.2}$$

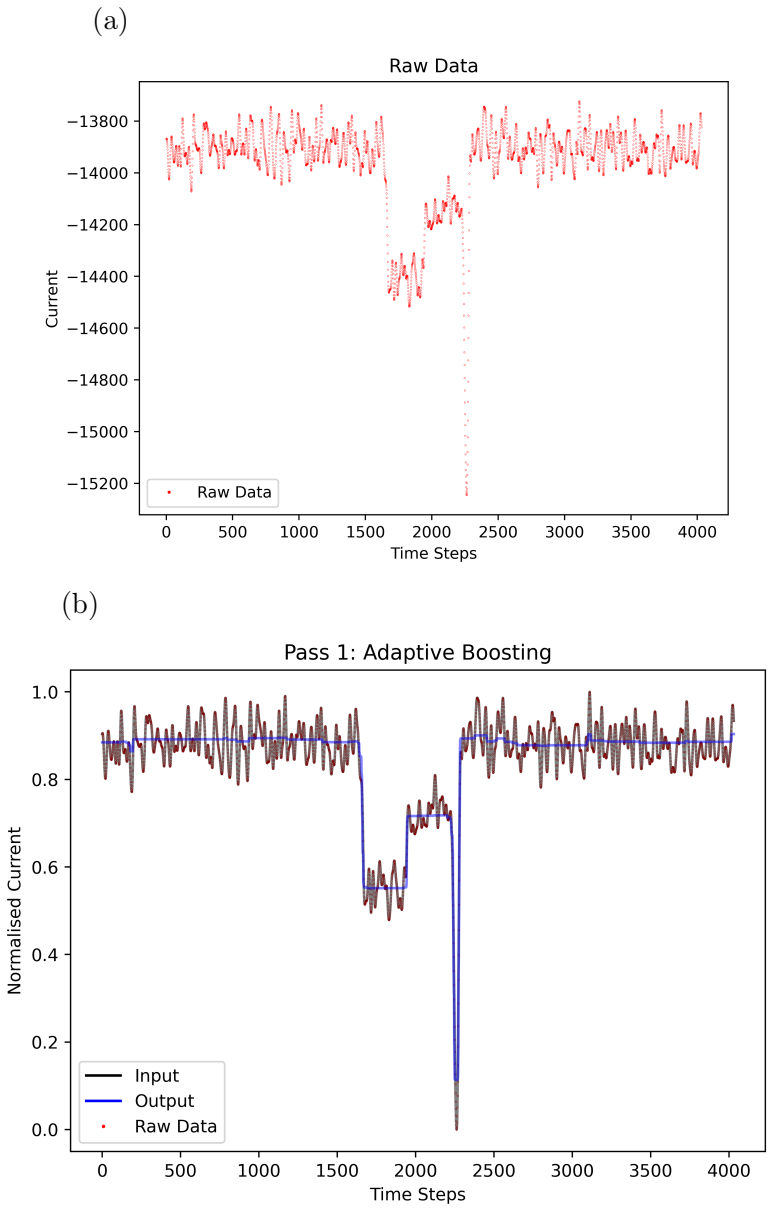

Figure 4.2: a) Example current trace caused by a partly folded 2kbp dsDNA carrier with a 12-arm DNA star cargo bound on one end generated from passing 12-arm star nanostructures attached to a 2kbp dsDNA carrier through a ∼13 nm pore (3.6 M LiCl, 75 mV) with a measurement bandwidth of 1 MHz (sampled at 4.17 MHz) and digitally low-pass filtered for fitting at 250 kHz [69]. b) The input and output from the first pass, consisting of the raw data normalized to (0,1) and an overfitted piecewise constant approximation using the AdaBoost algorithm, respectively.

$$Y_1 = \frac{Y - Y_{min}}{Y_{max} - Y_{min}} \tag{4.3}$$

$$X = [0, 1, \ldots, n-1] \tag{4.4}$$

After the algorithm learns the mapping, prediction is made using the same input X to get the output $\hat{Y_1}$ as shown in Equation 4.5, which is an array of current values that approximates the input as a piecewise constant to an extent. The goal of this pass is not to perfectly fit the event, but rather to smooth out the data without sacrificing much detail, erring on the side of overfitting, as shown in Figure 4.2b.

$$\hat{Y_1} = [\hat{y}_0^1, \hat{y}_1^1, ..., \hat{y}_{n-1}^1] \tag{4.5}$$

4.2.3 Pass 2: Decision Trees

The output from AdaBoost is a partially denoised signal containing many nonphysical sublevels, giving an overfitted but simplified estimate of the sublevel structure of the event compared to the raw data. Using the output array $\hat{Y}_1$ from AdaBoost as the target value Y_2 as shown in Equation 4.6 , we apply a single decision tree regression [10, 38] to further reduce the noise in the signal to more closely represent the piecewise constant function with the number of pieces in it being as close to the real count of sublevels as possible. Similar to the previous pass, here too the input to the decision trees algorithm is an array of indices X as shown in Equation 4.4. Using the same indices X as shown in Equation 4.4 as the input and Y_2 as the expected output, the decision trees model is trained. Then, using the same input X on this trained model we get the output $\hat{Y_2}$ as shown in Equation 4.7. $\hat{Y_2}$ is

close to our final fit visually but usually still contains some nonphysical sublevels which are identified and corrected in subsequent passes. Figure 4.3 shows an example of this pass.

$$\hat{Y}_1 = Y_2 = [y_0^2, y_1^2, ..., y_{n-1}^2] \tag{4.6}$$

$$\hat{Y}_2 = [\hat{y}_0^2, \hat{y}_1^2, \ldots, \hat{y}_2^{n-1}] \tag{4.7}$$

4.2.4 Pass 3: Merge Small Current Steps

The merging of small current steps or merging of similar current heights is the Pass 3 in the Nano Trees algorithm that converts the partly denoised data into sublevels deemed to be physical, based on the user-provided hyperparameters as shown in Figure 4.4. This pass performs 3 major tasks, namely, boosting, merging, and refreshing estimates, which we explain next.

4.2.4.1 Subpass 1: Boosting

In some cases, the effect of systemic distortion leads to an over- or under-estimate of the current within the sublevel, which is detectable by considering the sign of the residuals of the local fit. To avoid issues in which levels are erroneously merged due to inaccurate estimates, we first apply a correction step. In cases where the local fit has residuals that are asymmetrically distributed around zero, we apply a simple correction using two hyperparameters - $oneSidedPercentParity$ and $minDataPointsToBeBoosted$. A sublevel needs correction if the absolute difference between the number of positive residuals and

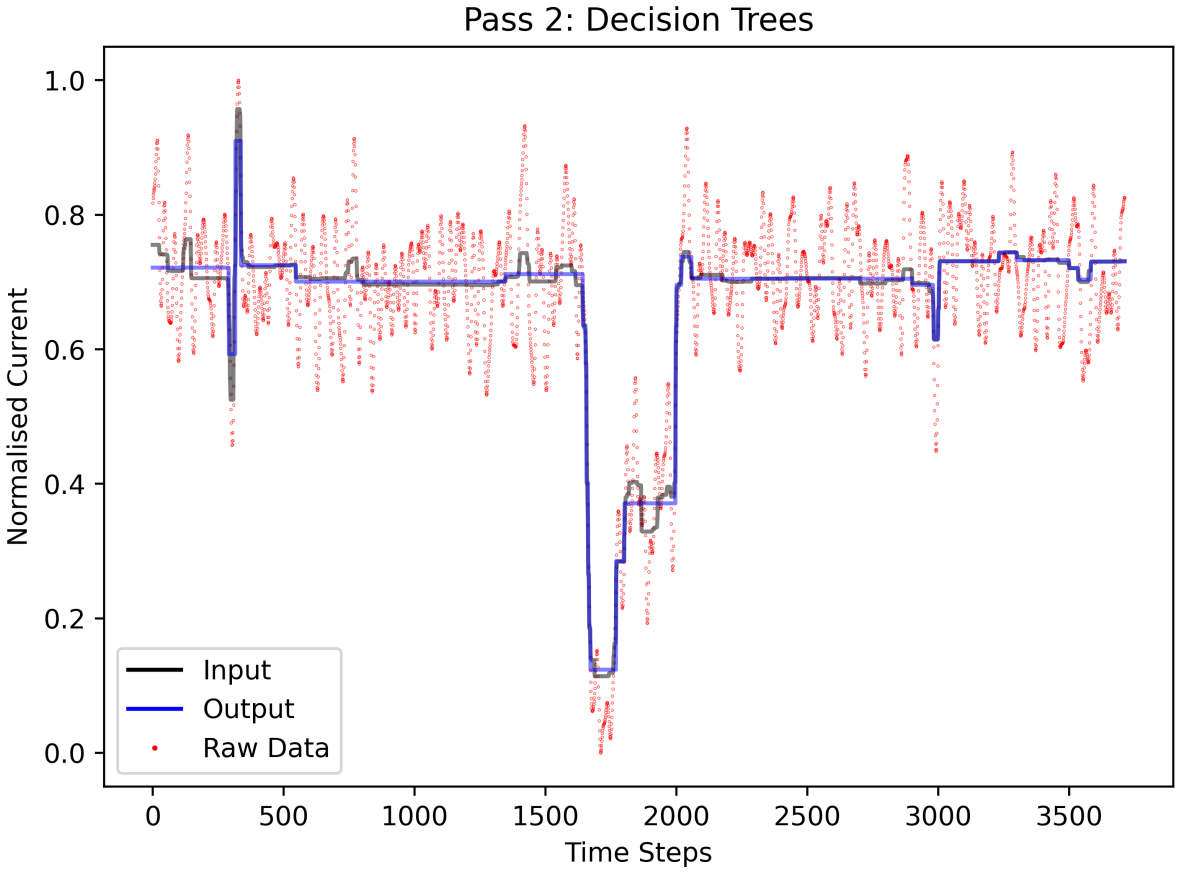

Figure 4.3: Input and output of Pass 2 – Decision Trees. The input to pass 2 is the output from pass 1. The output is an overfitted but improved fit to the raw data, shown in red for visual reference.

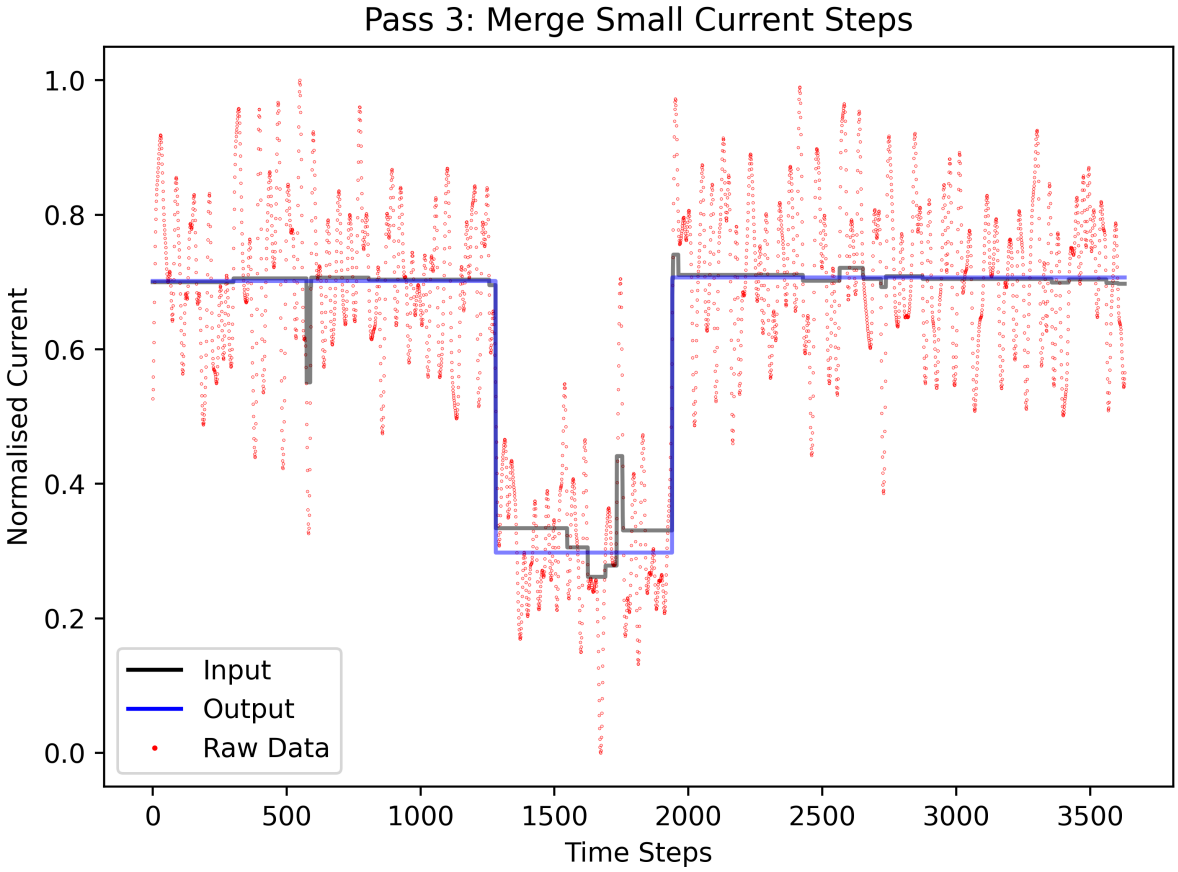

Figure 4.4: Input and output of Pass 3 – Merge Small Current Steps. The input is the output from pass 2, while the output is a smoothed version of the fit with sublevels that have unphysically small steps between them merged.

negative residuals is greater than $oneSidedPercentParity$. If a sublevel needs correction and it has more data points than $minDataPointsToBeBoosted$ then the sublevel current is estimated as either the mean of the last 50% of data in the sublevel, the data point that deviates most from the previous sublevel, or the mean of the whole sublevel, depending on which one results in a minimal asymmetry in the sign of the resulting errors.

4.2.4.2 Subpass 2: Merging

Up to this point, the algorithm still tends to produce overfits. These are smoothed over by combining neighbouring sublevels into a single sublevel when the difference in current between them is too small to represent a physical change, as defined by the user. The process of merging small current steps requires several parameters, of which $numberOfStdAboveAndBelow$ does the major heavy lifting. We use this parameter to create a threshold, such that if any sublevel has a current value within that threshold of its previous sublevel, then it is merged with it.

Algorithm 4.3 defines the exact functionality of this merging procedure. Here σ is the standard deviation of baseline, calculated beforehand, and is used for ease of defining thresholds. *sublevels* is an array structure that holds the positions and heights of all sublevels from the previous pass. We iterate through this to delete the sublevel that does not cross the threshold defined by *mhd* which represents the threshold for the minimum height difference between two consecutive sublevels to be considered a separate sublevel. If any sublevel fails that threshold, then it is deleted and merged with neighbouring sublevels using the $MERGE$ procedure in Algorithm 4.3, and the iteration over *sublevels* restarts until one complete iteration is done during which no sublevel has been deleted. When done, the *sublevels* now contain the merged sublevels and their heights.

Algorithm 4.3 Merge Similar Heights (Pass 3)

Input:
$\sigma \leftarrow$ Baseline standard deviation
$numberOfStdAboveAndBelow$ ▷ Hyperparamter
$sublevels$ ▷ This will store the array of sublevels
Procedure:
function MERGE($sublevel1, sublevel2$)
    Apply $l50\_max\_height$ function to the combined region of sublevel1 and sublevel2.
    return combined sublevel with updated height and position.
end function
$mhd \leftarrow numberOfStdAboveAndBelow \times \sigma$
$n \leftarrow$ Number of sublevels
while true **do**
    for $i \leftarrow 1$ to $n - 1$ **do**
        $check \leftarrow |sublevels[i].height - sublevels[i + 1].height|$
        if $check < mhd$ **then**
            $sublevels[i] \leftarrow$ MERGE($sublevels[i], sublevels[i + 1]$)
            del sublevels[i+1]
            break ▷ Exit the current for loop and restart from the beginning.
        end if
    end for
    if $i \geq n - 1$ **then**
        break ▷ If in an iteration no sublevels are merged then merging is complete.
    end if
end while
Output: $sublevels$

4.2.4.3 Subpass 3: Refreshing Estimates

In some cases, large deviations from the local sublevel current estimate within that sublevel can lead to errors in sublevel classification downstream. We use the *exceptionalHeightBaseMaxDiffForHeightRefresh* hyperparameter to update any sublevel if any point within that sublevel deviates from the local fitted current estimate by more than this value, using the default sublevel current estimation function.

4.2.5 Pass 4: Categorize and Correct Sublevels with Short Durations

It is often challenging to identify whether a short sublevel represents a physical change. A short sublevel here refers to a sublevel that has a low number of data points in it. Simply removing them is not a solution as physical sublevels can often have very short durations relative to the system rise time. In this pass, we identify short sublevels and categorize them as follows:

1. Normal sublevels: Sublevels that last long enough to reach a steady state (Figure 4.5a), and do not require correction in this pass. This is determined by the "minDataPointsToBeSubLevel" hyperparameter.

2. Peaked sublevels: Sublevels which do not last long enough to reach a steady state between current steps in opposite directions (Figure 4.5b).

3. Sloped sublevels: Sublevels which do not last long enough to reach a steady state between two current steps in the same direction (Figure 4.5c).

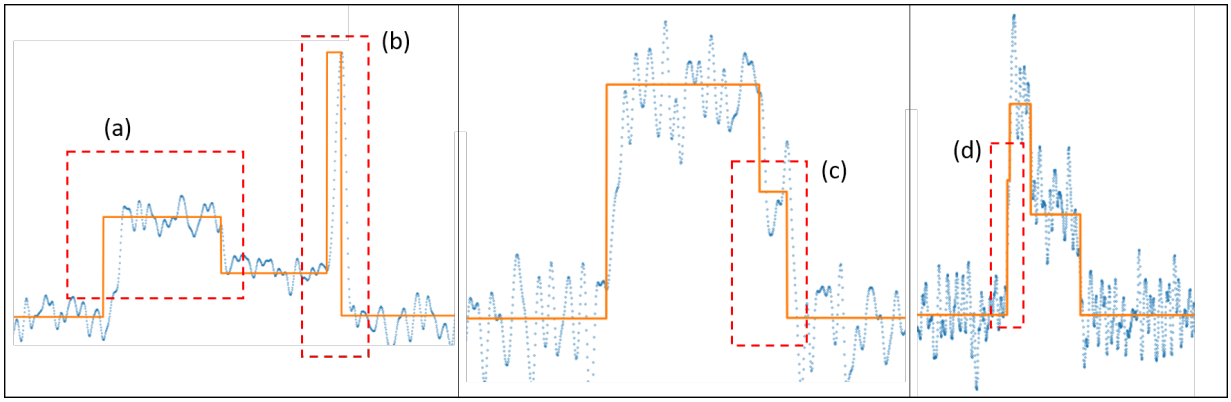

Figure 4.5: An example of all four types of sublevels in any nanopore data. a) Normal sublevel b) Peaked sublevel c) Slope sublevel and d) Bad sublevel.

4. Bad sublevel: Any sublevel not classified into one of the previous categories (Figure 4.5d).

This categorization is done using Algorithm 4.4. The algorithm returns true for Normal, Peaked and Sloped sublevels, but returns False for bad sublevels. The algorithm works using several hyperparameters, but the $minDataPointsToBeSubLevel$ hyperparameter is the only required hyperparameter, others are optional. $minDataPointsToBeSubLevel$ defines the minimum number of data points a sublevel must have so as to not be deleted, and these sublevels are called Normal sublevels.

Not all sublevels that are not normal sublevels have to be deleted because there are other criteria that make a sublevel physically viable besides having a decent amount of data points in it. To account for that, Nano Trees has a special set of hyperparameters, called exceptional hyperparameters. These are optional but can be set to allow for peaked and sloped sublevels. The exceptional parameters that allow for further testing for peaked sublevels begin with $exceptionalPeak\_$ and include:

1. $exceptionalPeak\_MinHeightStdAboveAndBelow$ - The absolute difference between

Algorithm 4.4 Sublevel category for Pass 4.

Input:
$sublevels$ ▷ This will store the array of sublevels
$i \leftarrow$ Index of sublevel to categorize.
$\sigma \leftarrow$ Baseline standard deviation
$minDataPointsToBeSubLevel$ ▷ Hyperparameters
Exceptional slope and peak hyperparameters.
Procedure:
$n \leftarrow$ Number of datapoints in sublevels[i]
if $n \geq minDataPointsToBeSubLevel$ **then**
    return true ▷ Normal sublevel (Keep)
end if
$prevHeight \leftarrow$ Height of previous sublevel
$nextHeight \leftarrow$ Height of next sublevel
$hd1 \leftarrow sublevels[i].height - prevHeight$
$hd2 \leftarrow sublevels[i].height - nextHeight$
$exceptionalHeight \leftarrow exceptionalPeak\_MinHeightStdAboveAndBelow \times \sigma$
$e \leftarrow \sigma \times exceptionalPeak\_BaseDifferenceStdAtleast$
$c1 \leftarrow |hd1| > exceptionalHeight$
$c2 \leftarrow |hd2| > exceptionalHeight$
$c3 \leftarrow hd1 \times hd2 \geq 0$
$c4 \leftarrow sublevels[i].width > exceptionalPeak\_WidthLowerBound$
if $c1$ **and** $c2$ **and** $c3$ **and** $c4$ **then**
    $nextHeightNonExceptional \leftarrow$ Next sublevel after the current sublevel that has more data points in it than $minDataPointsToBeSubLevel$
    $checkC5 \leftarrow |nextHeight\_nonExceptional - prevHeight|$
    if $checkC5 > exceptionalBaseDifferenceStdAtleastHeight$ **then**
        return true ▷ Peaky sublevel (keep)
    end if
end if
$c1 \leftarrow \min(|hd1|, |hd2|) > (exceptionalSlope\_MinHeightStdOfMinDiff \times \sigma)$
$c2 \leftarrow sublevels[i].width > exceptionalSlope\_WidthLowerBound$
if $c1$ **and** $c2$ **and not** $c3$ **then**
    return true ▷ Slope sublevel
end if
return False ▷ Bad sublevel
Output: Boolean defining whether or not to keep this sublevel.

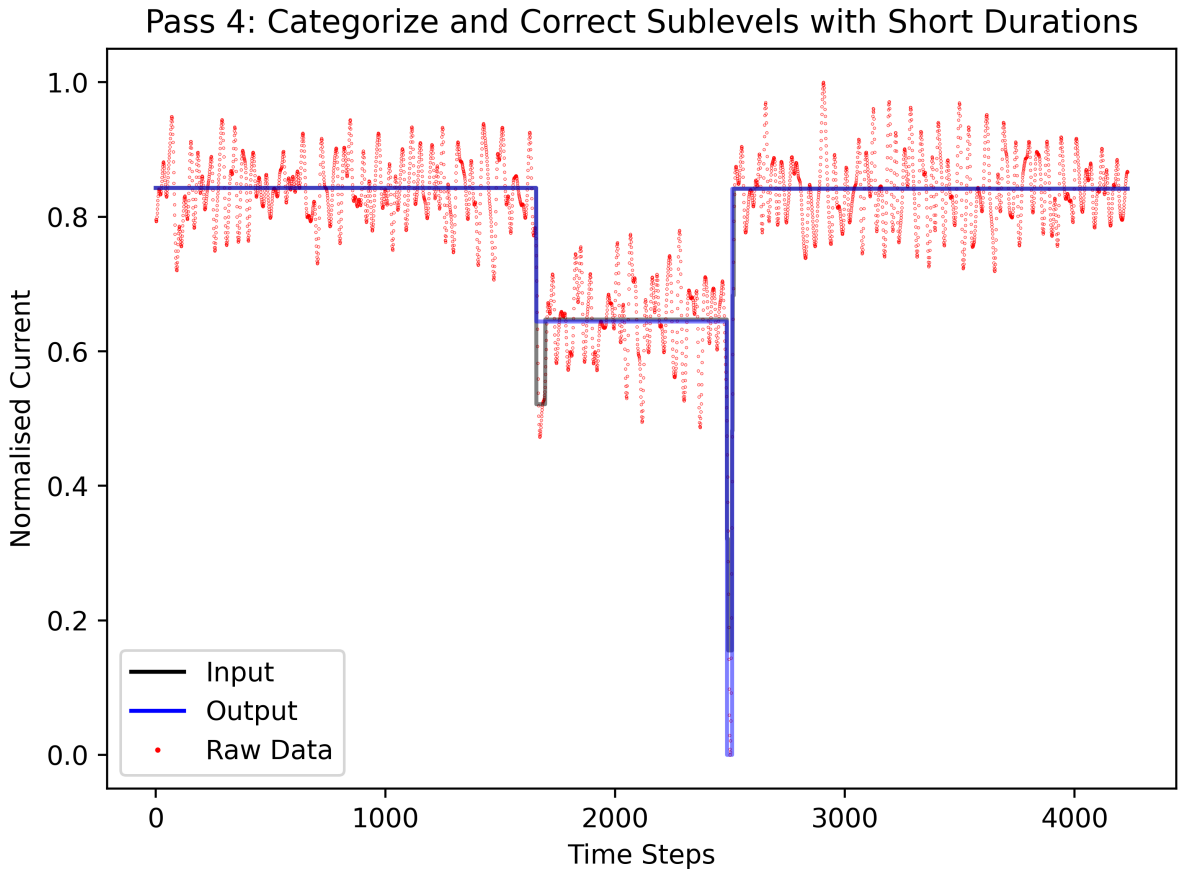

Figure 4.6: Input and output of Pass 4 – Categorize and Correct Sublevels with Short Durations. The input is the output from pass 3, while the output is an improved fit which corrects underestimates of current steps for sublevels that approach the time resolution of the system.

the height of a sublevel and both the previous and next sublevel height should be greater than this parameter (multiplied by baseline standard deviation).

2. $exceptionalPeak\_WidthLowerBound$ - The exceptional peaky sublevel should have more data points in it than this parameter.

3. $exceptionalPeak\_BaseDifferenceStdAtleast$ - The absolute difference between the height of the previous sublevel and the height of the first non-exceptional sublevel after the current sublevel should be greater than this parameter (multiplied by baseline standard deviation).

Exceptional parameters that allow for further testing for peaked sublevels begin with $exceptionalSlope\_$ and include:

1. $exceptionalSlope\_MinHeightStdOfMinDiff$ - The exceptional slope sublevel should have more data points in it than this parameter.

2. $exceptionalSlope\_WidthLowerBound$ - The minimum of, absolute height difference between the current and previous sublevel, and between the current and next sublevel should be greater than this parameter (multiplied by baseline standard deviation).

Sublevels that are neither normal sublevels nor pass the exceptional check for peaked or sloped sublevels are deemed nonphysical and deleted iteratively. The hyperparameter $numberOfStdAboveAndBelow$ is used to decide exactly where such sublevels are split according to Algorithm 4.5. If the sublevel to be deleted is at the beginning or end of the event, it gets merged with the sublevel next to it. For all other sublevels, the hyperparameter $numberOfStdAboveAndBelow$ is used to devise an up-and-down threshold based on the sublevel's standard deviation (σ_i), and the first point from the left outside this

threshold is used as the split point. All data points to the left of this are added to the left sublevel, and all data points to the right of it are added to the right sublevel. Whenever a sublevel is split, the heights are updated using Algorithm 4.2. Figure 4.6 presents an example of these steps in this pass.

4.2.6 Pass 5: Merge Small Current Steps (Repeat)

The previous pass removes most false positive sublevels. However, in some cases, sublevels that survived Pass 4 could attain a current estimate close to each other. This requires a second call to Algorithm 4.3. It is observed that after Pass 5, there is no need to Pass 4 because Pass 5 does not create any new sublevel that has a small width, thus a repetition would not affect the fit. Figure 4.7 presents an example of this pass.

4.2.7 Pass 6: Clear Baseline

Because this pipeline treats the baseline before and after the event as its own sublevel, it is possible that the fits to this point have assigned more than one sublevel to the baseline current. These sublevels are not physical and can simply be removed. Using the approximate start and endpoint of the event, any sublevels that fall outside of these bounds are merged into a single baseline sublevel. These points can be extracted using a simple step change detection from both sides of an event from the baseline with a threshold of 3 times the baseline standard deviation. Figure 4.8 presents an example of this pass.

Algorithm 4.5 Split sublevels with small widths (Pass 4)

Input:
$sublevels$ ▷ This will store the array of sublevels
$i \leftarrow$ Index of sublevel to check
$numberOfStdAboveAndBelow$ ▷ Hyperparameter
Procedure:
function MERGE($sublevel1, sublevel2$)
    Apply Algorithm 4.2 to the combined region of sublevel1 and sublevel2.
    return combined sublevel with updated height and position.
end function
while true **do**
    $n \leftarrow$ Number of sublevels
    for $i \leftarrow 1$ to $n - 1$ **do**
        if ALGORITHM 4.4($sublevels[i]$) $= false$ **then**
            if $i = 0$ **then**
                $sublevels[1] \leftarrow$ MERGE($sublevel[0], sublevel[1]$)
                delete $sublevel[0]$ and **restart** the for loop
            else if i=n-1 **then**
                $sublevels[n-2] \leftarrow$ MERGE($sublevel[n-2], sublevel[n-1]$)
                delete $sublevel[n-1]$ and **restart** the for loop
            else
                $\sigma_i \leftarrow$ standard deviation of sublevel i
                $threshold \leftarrow \sigma_i \times numberOfStdAboveAndBelow$
                $upThreshold \leftarrow sublevel[i].height + threshold$
                $downThreshold \leftarrow sublevel[i].height - threshold$
                $j \leftarrow$ First data point in the sublevel greater that $upThreshold$ or lower than $downthreshold$
                $sublevels[i-1] \leftarrow$ MERGE($sublevel[i-1], sublevel[i][0:j]$)
                $sublevels[i+1] \leftarrow$ MERGE($sublevel[i+1], sublevel[i][j:end]$)
                delete $sublevel[i]$ and **restart** the for loop
            end if
        end if
    end for
    If no sublevels are merged then merging is complete, exit the while loop.
end while
Output: $sublevels$ ▷ With merged and updated sublevels

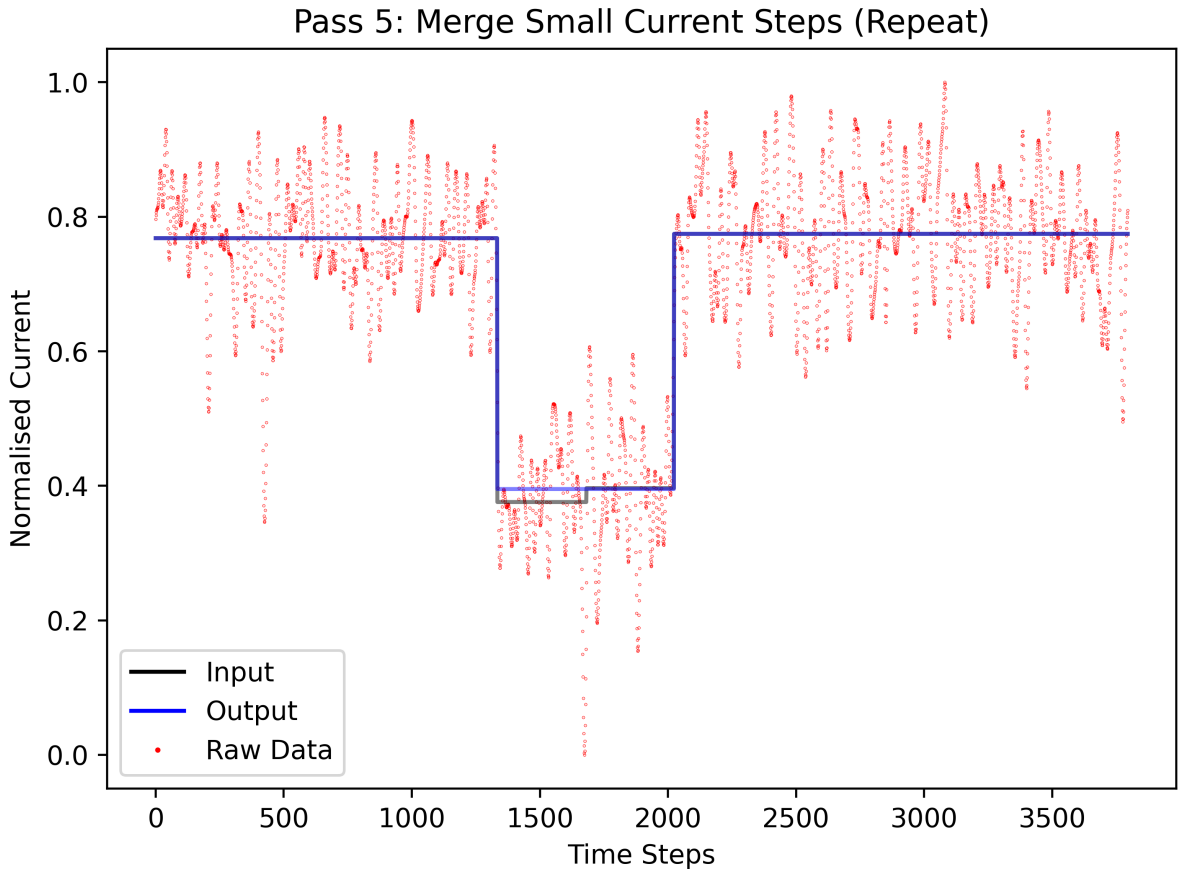

Figure 4.7: Input and output of Pass 5 – Merge Small Current Steps. The input is the output from pass 4, while the output is a smoothed version of the fit with sublevels that have small current change between them merged.

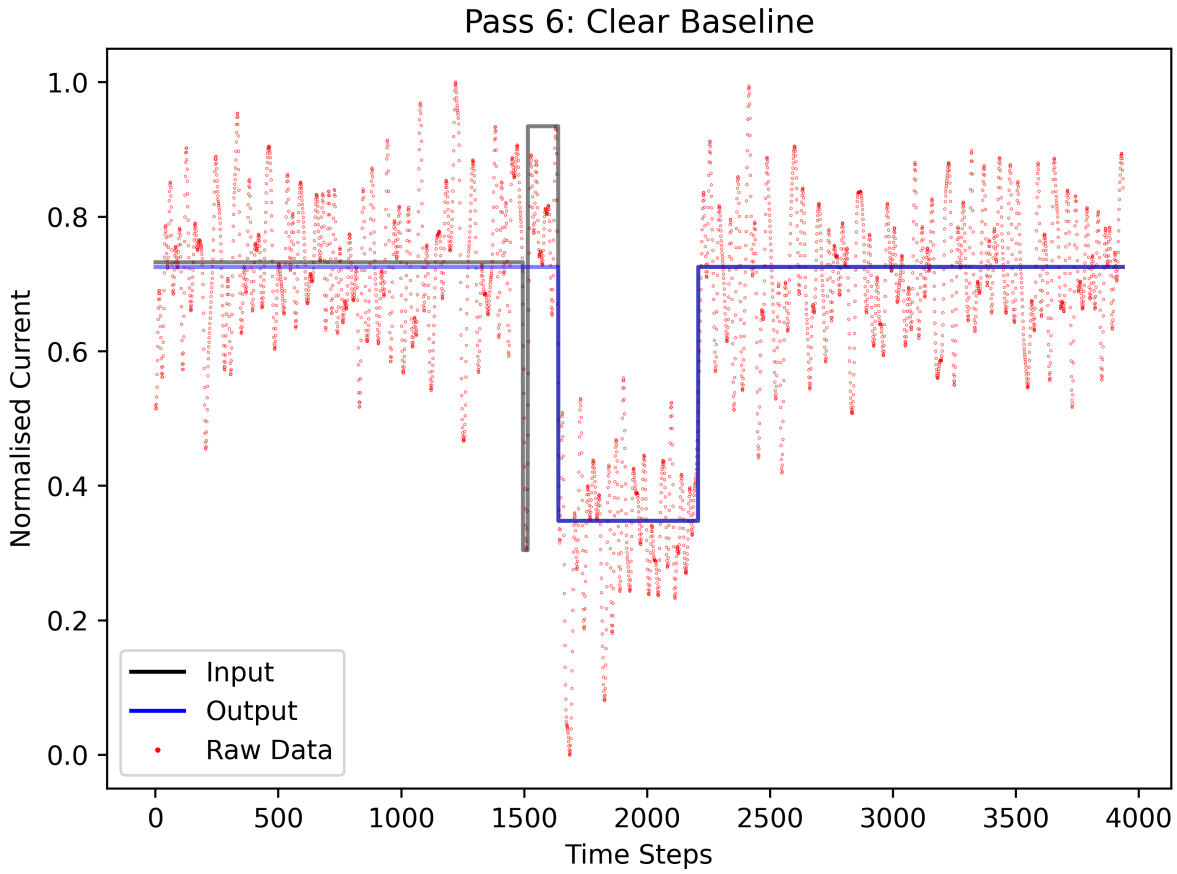

Figure 4.8: Input and output of Pass 6 – Clear Baseline The input is the output from Pass 5, while the output has suppressed any sublevels that were detected before the beginning or after the end of the event.

4.2.8 Pass 7: Backtrack

Having identified all physical sublevels in the event, we must correct their start and end times to the proper data point to account for the system rise time as previously discussed. We update the starting point of each sublevel by iteratively calculating the sign of the difference between the current point and the previous point in the backward direction, until it is the same as the sign of the difference between the current estimate of the current sublevel and the current estimate of the next sublevel, backtracking until we have reached the first point to depart from the previous sublevel. For a bandwidth-limited system, this corresponds to the actual time at which the event occurred, and the sublevel begins. Once all sublevels have been backtracked, the current estimates are updated using the default sublevel current estimation function and the new sublevel time bounds, as shown in Figure 4.9.

4.2.9 Post-Processing

For the results presented in this work, the fitting process is completed with slight restorations detailed below. The current estimate of the Slope sublevel is slightly adjusted based on the mean of the last 50% of data, replacing the present current estimate if it reduces the ratio of data points above and below it as shown in Figure 4.10. The event is then denormalized using the original maximum and minimum values. Finally, event-based information, including the number of sublevels, sublevel changepoints, and sublevel current estimates are extracted and stored for further testing and analysis. Figure 4.11 represents a sample fit of this entire algorithm.

More passes can be added or repeated to further improve the fit, but the seven passes described above that are combined to create Nano Trees strike a balance between ap-

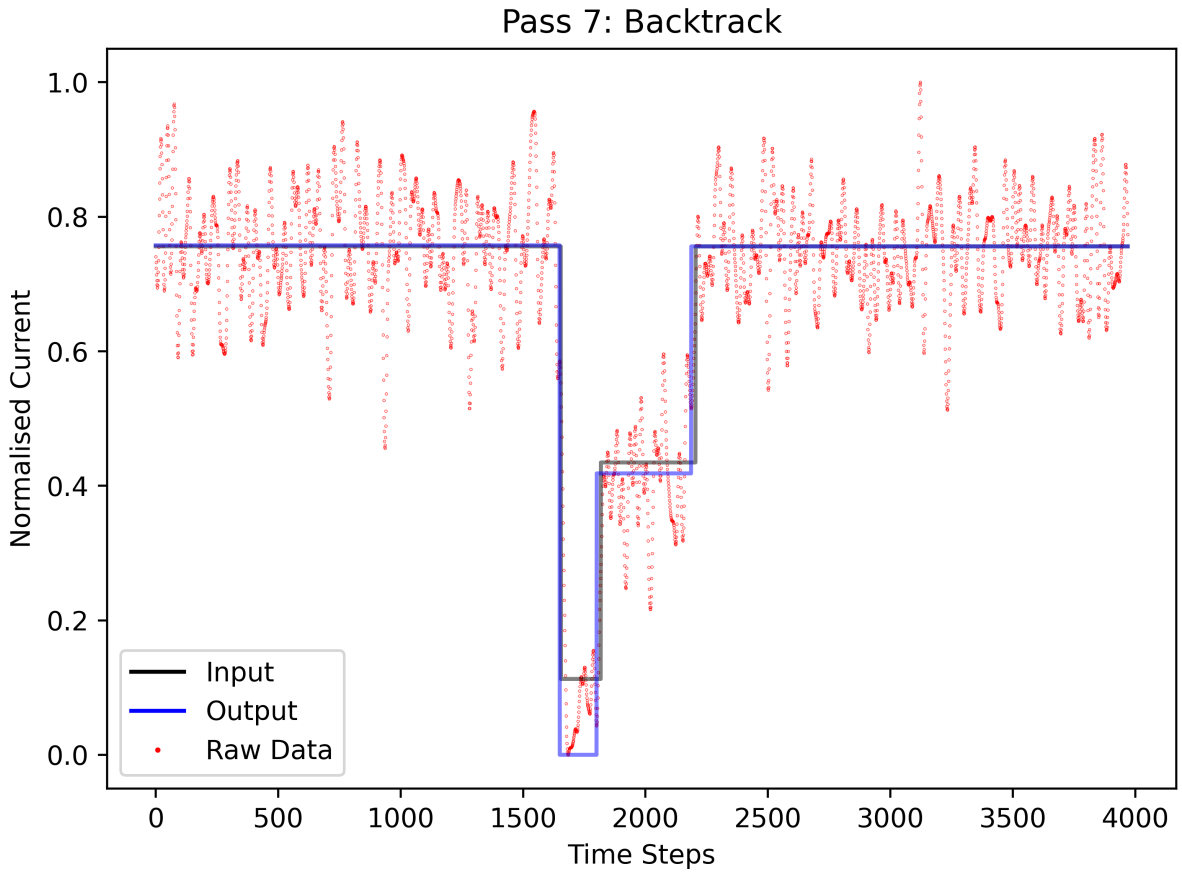

Figure 4.9: Input and output of Pass 7 – Backtrack. The input consists of the output from pass 6, while the output corrects minor errors in the time indices of the changes between sublevels that arose from previous passes.

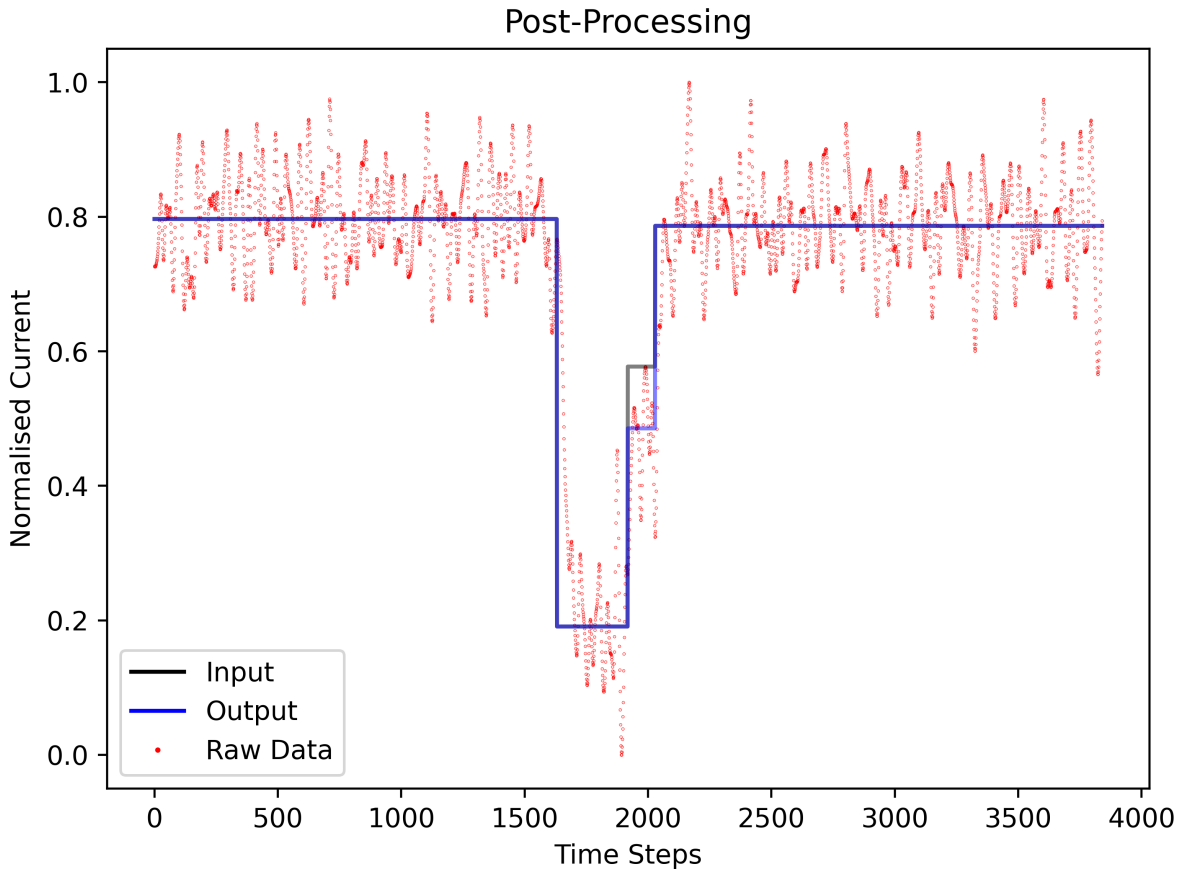

Figure 4.10: Input and output of Post-Processing. Having corrected the start and end times, this pass applies one final check to ensure accurate baseline level fitting.

proximation (fitting a smaller number of larger sublevels) and detail (fitting many smaller sublevels), ensuring that we can accurately characterize the current fluctuations occurring within the nanopore as molecules translocate through.

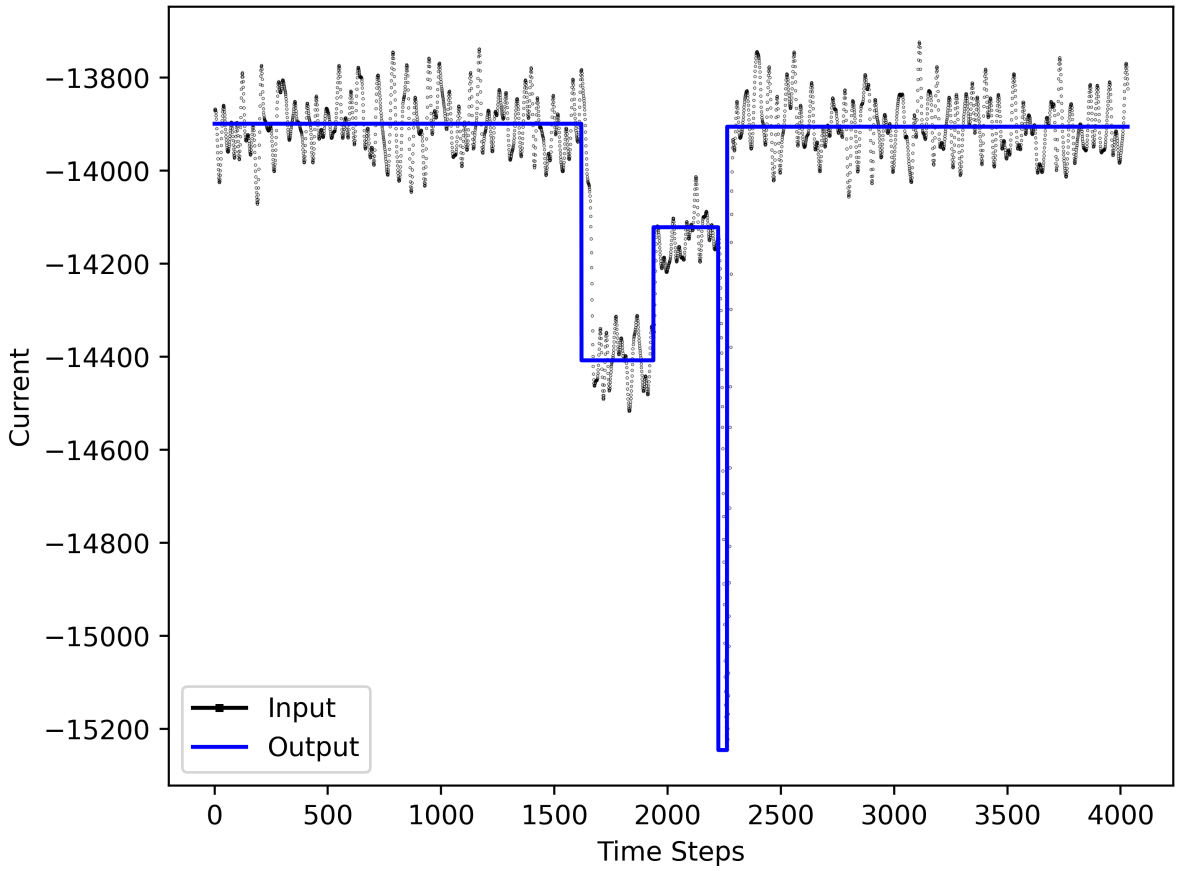

Figure 4.11: A sample event fit by Nano Trees algorithm.

4.3 Implementation - Fitting Blocks Project

The Nano Trees algorithm code is implemented in a block manner, such that blocks can be added, removed, or replaced based on various implementations, and is called the Fitting Blocks Project. Before developing Nano Trees, there were several directions to explore for developing an algorithm like this. Hence, the Fitting Blocks Project was built to make sure that it could support the implementation of any idea, in a fast and efficient manner while preventing any redundancies. A lot of code that goes into implementing an idea, usually stays the same, like reading a dataset, filtering it, extracting metadata, etc. The Fitting Blocks Project allowed the creation of reusable blocks for these tasks. This means that they could be shared between implementations with minimal duplication, easily modified, and added or removed speeding up the transition from concept to code to testing.

Besides handling data, the actual algorithm implementation is also modular. Each step of the algorithm is implemented as a pass, like the passes in the Nano Trees algorithm. The passes work sequentially, but the entire pipeline of passes can be parallelized. Since the processing of one event does not depend on the processing of another, multiprocessing can be used to improve efficiency.

The code is written entirely in Python. This allows it to be Operating System (OS) agnostic as Python can be executed on almost any OS. Packages like NumPy and Pandas have been used extensively for data and numerical processing. Matplotlib has been used for plotting and intermediate-step debugging. Other packages like tqdm, pickle, etc. are also used, and a list of the exact packages and their versions is present in the "requirements.txt" file in the GitHub repository of the Fitting Blocks Project which currently only contains one implemented algorithm, that is Nano Trees.

4.4 Automation

Nano Trees is a long algorithm that requires several hyperparameters to do the fitting. These hyperparameters can be tricky to tune but are the reason for the flexibility and generalizability that this algorithm offers. Hence, it is important to tune these hyperparameters well to get good results while fitting any data. This section discusses two automation steps that make the task of tuning the hyperparameters easier. These steps are not developed to obtain the best values for this task. Instead, they provide a standardized way to get the starting values for several hyperparameters over which the user can choose to fine-tune further if needed.

4.4.1 Overfitting Automation

Pass 1 and Pass 2 in Nano Trees try to overfit the data (as discussed in Section 4.1.3) to prevent any sublevels from being missed in the further passes while also reducing as much noise as possible in the fit. This automation aims to streamline the results of these passes by iteratively exploring various hyperparameter values for the decision tree regressor to automate this process of getting an overfit on the events. This automation is called "Overfitting Automation" not because we are trying to overfit the data and present it as the final output of Nano Trees, but because it is trying to "automate" the "overfitting" part of the Nano Trees pipeline, which is Pass 1 (Section 4.2.2) and Pass 2 (Section 4.2.3). In the scikit-learn [58] implementation of decision tree regression, there are several hyperparameters with numerous possible values. This makes the overall search space extensive and unusable for practical applications. To reduce this search space, we went through several manual iterations of various combinations of hyperparameters to get meaningful results that are practically possible in a reasonable amount of time.

Using the Friedman MSE [30] as the criterion for node splitting, we iterate through the max_leaf_nodes parameter in the range of $[2, 100]$, keeping the default values for the remaining parameters. Next, we plot the result of this iteration as shown in Figure 4.12a and observe that the curve has a bent point often referred to as a knee/elbow point. We extract this point using the "kneed" algorithm developed by Satopaa, Ville, et al. [71]. More details about this process, the parameters used, and the exact algorithm are discussed in Appendix A.2. In Figure 4.12a the Y-axis represents MSE for various max_leaf_nodes, the points to the left of the extracted elbow point can be considered as an underfit because the MSE are highly varied in this region and is consistently decreasing for the most part. The points after the elbow have a significantly lower rate of change for the MSE which means that even when the the number of leaves in the tree is increased the overall quality of the fit is not much impacted. It is also important to note that the elbow point cannot be used directly as we are looking to overfit the data here. So, to make sure that this value overfits the entire dataset, using just one run of this automation, while neither missing any sublevels nor overfitting too much, we multiply the elbow point by a factor of α. In our research $\alpha = 2$ seemed to produce good results for most events.

Friedman MSE, a variation of traditional MSE designed to reduce the variance of the error estimate, was chosen because it provides better generalization performance, by penalizing large deviations and thus helping in achieving more robust splits. MSE can alternatively be used as well, but Friedman MSE provided better and faster results in the limited testing performed on several datasets. The max_leaf_nodes or the "Maximum Leaf Nodes" parameter controls the maximum number of leaf nodes the decision tree can have, thereby controlling the complexity and consequently, the approximate number of sublevels in the fit (after that pass) produced by the model. The range of $[2, 100]$ is an arbitrary range, where 2 is the minimum acceptable value for this parameter, but can be

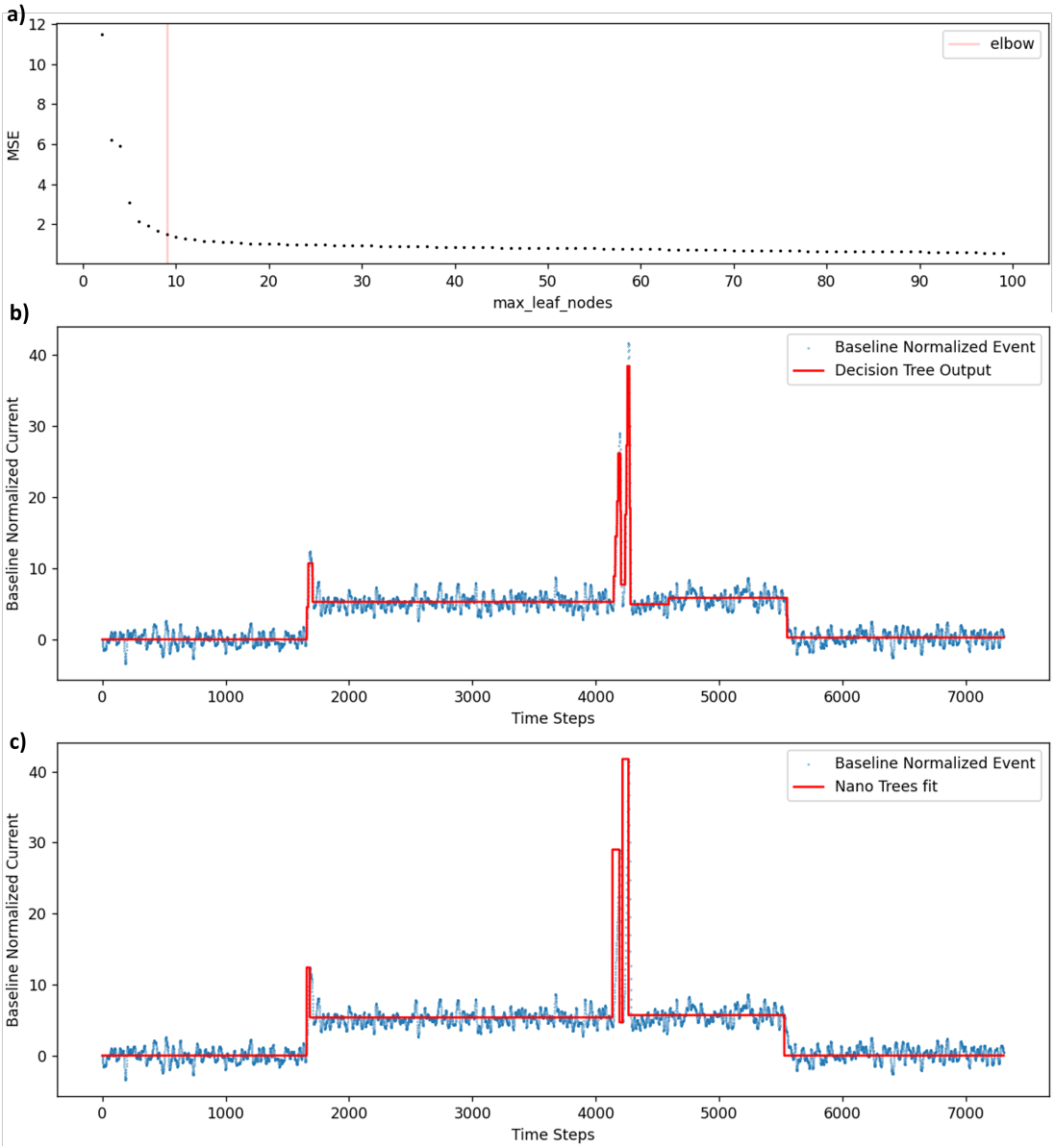

Figure 4.12: Overfitting automation summary. a) MSE values for increasing max_leaf_nodes parameter and the calculated elbow at $max\_leaf\_nodes = 9$. b) Decision Tree output of the overfitted fit on an event from Section 5.3 with $max\_leaf\_nodes = 9 * 2 = 18$. c) Final fit produced using Nano Trees utilizing the automation overfit.

switched to 3, to correspond to the minimum number of sublevels in an event (Baseline, single sublevel, and baseline). The events used here are baseline normalized, which means the entire event is normalized using the mean and standard deviation of the event baseline. The Y-axis in Figure 4.12a is MSE because it is easier to interpret. There can be other variations using other parameters, as discussed in Appendix A.1.

Finally, using this value of 2 times elbow point, we train the model (Decision Tree in this case, but others like AdaBoost can also be used) to get automated overfit on the event data as shown in Figure 4.12b and processed further in the Nano Trees pipeline as shown in 4.12c, skipping Pass 1 and Pass 2 entirely. In Figure 4.12 the elbow point was determined to be 9, using the method discussed in Appendix A.2. Using 18 as the value for max_leaf_nodes and Friedman MSE criterion, the overfitted fit on the event was produced as shown in 4.12b and using Nano Trees, the smoothed out final fit was produced by Nano Trees. Figure A.2 shows other results of this procedure. The process of overfitting is highly forgiving and by using $\alpha = 2$ we ensure that the result falls in the acceptable range in most cases, but it might still fail sometimes. This automation is more of a heuristic than an algorithm, that can provide good starting values for the hyperparameters in most cases and should be considered as an optional add-on to Nano Trees.

4.4.2 Sublevel Loss Function

Optimizing any process necessitates having a specific target to optimize towards. One of the major issues with fitting the piecewise constant function on nanopore data was not having a ground truth. This automation is divided into two parts, the first part will describe a method to extract a value that helps determine the automated starting value

of several hyperparameters in the middle passes of the Nano Trees algorithm, and also for the functioning of the second part, the actual loss function. The second part describes this loss function for individual sublevel fitting which provides the direction toward the ground truth and leaves us to find the method to get to it and when to stop.

4.4.2.1 Minimum Delta

Δ_{min} represents the minimum vertical distance that a sublevel must have from its previous sublevel for it to be not merged with it. It is not a fixed quantity and corresponds to several hyperparameters in the Nano Trees pipeline. It is not possible to extract this value from the event itself without fitting the event, because the fitting process requires this value to achieve an accurate fit. Without incorporating this value correctly, the fitting algorithm can fail to function properly, potentially introducing inaccuracies. Currently, users approximate this value through guesswork, highlighting the critical need for automation to enhance accuracy and reliability in the fitting process.

Event baseline is the region where the actual biomolecule used in the experiment is not translocating the nanopore. Hence, it is safe to assume that no sublevels exist in the baseline. We use the baseline to extract the size of continuously increasing or decreasing regions and build a distribution using Algorithm 4.6. The value of Δ_{min} can be derived using this distribution, as the mean of this distribution. Some other alternatives include the maximum of this distribution, median, or a value β standard deviations (σ) above the mean(μ) as shown in Equation 4.8. Δ_{min} can be used as the default value for $minDataPointsToBeSubLevel$ and $minDataPointsToBeBoosted$, directly or with a constant scaling factor. Section 4.4.2.2 will further describe another use of Δ_{min}.

Algorithm 4.6 Determining continuous regions.

Input:
$array$ ▷ Input Array
Procedure:
$previousDirection \leftarrow sign(array[2] - array[1])$
Let n be the size of $array$
$continiousRegions \leftarrow []$
$count \leftarrow 0$
for $i \leftarrow 1$ **to** $n - 1$ **do**
    $currentDirection \leftarrow sign(array[i] - array[i - 1])$
    if $currentDirection = previousDirection$ **then**
        $count \leftarrow count + 1$
    else
        $continiousRegions \leftarrow continiousRegions + [count]$
        $count \leftarrow 1$
        $previousDirection \leftarrow currentDirection$
    end if
end for
Output: $continiousRegions$

$$\Delta_{min} = \mu + \beta \times \sigma \tag{4.8}$$

4.4.2.2 2D Loss Function

The loss function we are discussing in this section is a single sublevel loss function. This means it is a better (in most cases) but slower, drop-in replacement of the height function (discussed in Section 4.2.1). Despite its advantages, it is considerably slower, so in scenarios where timely results are essential, this may not be a suitable substitute. Instead, it can be effectively utilized in situations where it is applied selectively rather than extensively, thereby mitigating its impact on overall processing time, such as for hyperparameter tuning, analyzing the quality of fits, final height correction after fitting, and other scenarios where precision is critical and ample time is available.

Given a sublevel s and a height estimate H for that sublevel, this loss function returns a value describing the goodness of that estimate. The sublevel is part of a 0-1 normalized event. A lower loss value signifies that the estimate is good and vice versa. This is an iterative function, that works by trying to distinguish normal sublevels (as discussed from 4.2.5) and short sublevels apart. For the short sublevels, which are the sublevels that last for a very short duration compared to other sublevels in the event, the loss is not calculated directly. For these sublevels, the algorithm automatically determines the optimal height, which can be used as is or can be converted into a pseudo-loss value to maintain consistency with the other sublevels if needed.

For short sublevels we use Algorithm 4.6, to extract Δ_{min} from baseline as discussed in Section 4.4.2.1. Next, we again use Algorithm 4.6 to determine the continuous region in the sublevel s, but this time on the sublevel being processed. Using these continuous

regions in the sublevels, we iteratively check for the smallest region, and if it is smaller than Δ_{min}, then it gets merged into its neighbours as discussed in Algorithm 4.7. Here, regions are being merged that can only have two patterns, "increasing" -> "decreasing" -> "increasing" or "decreasing" -> "increasing" -> "decreasing". In both cases, if the middle region is deleted then the neighboring regions have the same direction to them. Hence, whenever a region is deleted, it along with both of its neighboring regions are merged into one region. The process is repeated until the smallest region in the sublevel being processed is bigger than Δ_{min}. Once done, we count the number of remaining regions. If no regions are remaining, then something went wrong, probably in the calculation of Δ_{min}, determination of sublevels, or something else, manual debugging is required. Otherwise, there can be 3 distinct possibilities -

1. If only 1 region remains, then the extreme point of that region is used as the optimal sublevel height. This is because if all other regions in this sublevel were short enough to be iteratively removed by Algorithm 4.7 then the one remaining region encapsulates the entire sublevel with everything else being considered as noise.

2. If 2 regions remain, then the last point of the first region or the extreme value in the sublevel is used as the optimal sublevel height. This is because in this case, the sublevel has two major regions, one increasing and the other decreasing or vice versa. This is a case when some beginning portion from the next sublevel remains in the current sublevel but has an opposite direction. Therefore the main part of the current sublevel gets encapsulated by one region and the remaining by the second region.

3. If more than 2 regions remain, then this sublevel is not a short sublevel, and Algorithm 4.8 will be used to determine the actual loss for this sublevel and the height provided.

Here the primary takeaway point is the determination whether a sublevel is a short sublevel or not without the need of user interference. The height calculation can be done in numerous ways, of which some are listed above.

Algorithm 4.7 Iterative region reduction.

Input:
Δ_{min}
s
Procedure:
Let n be the size of array s
while true **do**
  Let i be the smallest region in s, and s[i] be m
  if $n \leq 1$ **OR** $s[i] \geq \Delta_{min}$ **then**
    break
  end if
  if i = 0 **then**
    Merge region 0 and region 1 in s
  else if i = n **then**
    Merge region n and region $n-1$ in s
  else
    Merge region i, $i-1$ and $i+1$ regions in s
  end if
end while
Output: Merged regions s

As the expected result from a loss function is to return a loss value, so this height estimate can be converted into a loss value L using Equation 4.9. Here, H is the provided height for this sublevel, and $\overline{H}$ is the estimated height calculated by the algorithm. If $\overline{H}$ is equal to H then the loss value will be the lowest, which is 0. Otherwise, it will have a positive value as high as 1 because the event is 0-1 normalized.

$$L = |H - \overline{H}| \tag{4.9}$$

For sublevel s which is not a sublevel, we use the provided height estimate H to determine a loss value. The name 2D loss function represents that we consider dimensions while calculating this loss value. So we begin by converting the 1D data points in the event (of size n) into 2D by defining the horizontal distance between each data point in the event. Due to normalization, the maximum vertical distance between any 2 points in the event is 1 as shown in Figure 4.13. So to have comparable horizontal values between the points, we scale the entire event to have $1/n$ distance between consecutive points and the distance between the first and the last point horizontally becomes 1. Now each point in the event has an 'x' and a 'y' coordinate, thereby making the data 2D. As the event may require different scaling horizontally we can have a scaling factor S_f multiplied to all horizontal distances. Therefore for any given point i in the event, the horizontal distance is given by Equation 4.10 as shown in Figure 4.13.

$$s[i] - s[i+1] = S_f/n \tag{4.10}$$

Now that we have filtered out short sublevels, converted the event into 2D and rescaled it, we can begin defining the 2D loss function for the remaining normal sublevels described in Algorithm 4.8. To begin with, we create two vertical and a horizontal line over the sublevel s of size n in consideration, as shown in Figure 4.13 to represent a pseudo fit. The vertical lines start from the first and the last point in s up till H which is the height provided for the sublevel. The horizontal line spans between the first and the last point in s with a y-coordinate of H. Now for each point in s, we calculate distances d_l, d_r and d_v from the left, right and horizontal lines respectively, as shown in Figure 4.13. d_l and d_r are only calculated for a point i if the y-coordinate if i falls within y-coordinates of the left and the right vertical lines respectively and their absolute values are less than a

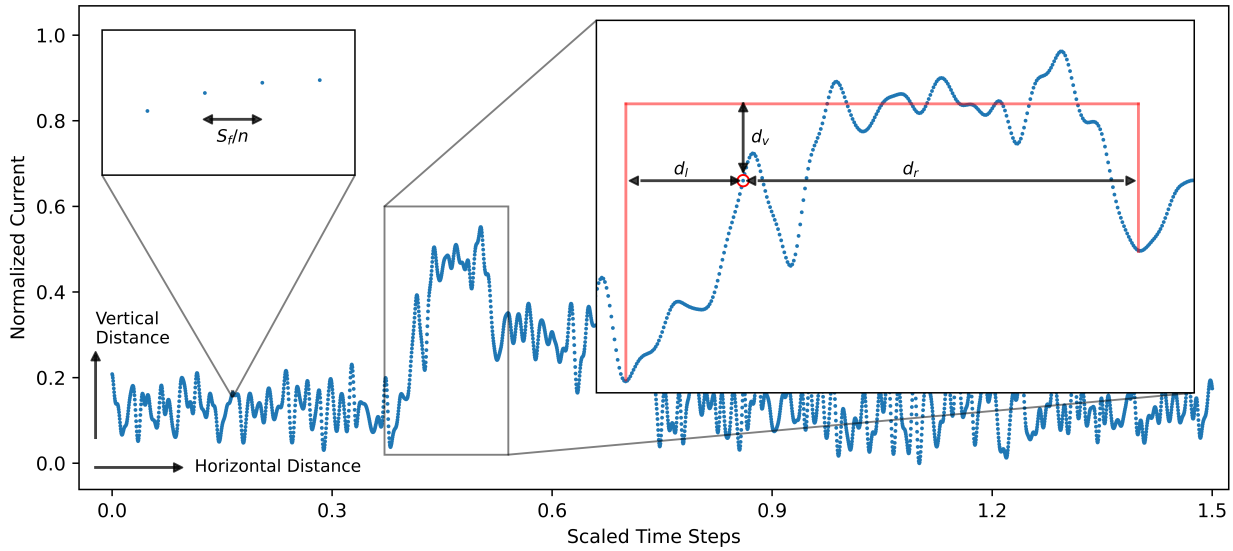

Figure 4.13: Event description for automation at $S_f = 1.5$.

preset threshold of Th_l and Th_r respectively. d_v is calculated for all points but an optional threshold Th_v can be used as well. If both d_l and d_r exist for a i, the smallest of the two is considered while calculating the loss, otherwise the one that does exist. If none of them exist for i then d_v is used. An optional condition can be added if Th_v is in use, such that if neither of d_l, d_r or d_v exists then a constant value or computational penalty of some sort is used for i signifying that it is far from all the 3 lines, making the loss value high. After deciding what value to use for i, we add that to a running loss value counter, add up these values for each i in s and return the average of this as the final loss for s, for a given height H.

This 2D loss function can have several applications as previously discussed. One of those applications is in the determination of sublevel height. By iterating through several height values h in a sublevel, a loss plot can be generated. The height that provides the lowest loss value can be used as the best height estimate for that sublevel. Figure 4.14 demonstrates this procedure of height determination. Figure 4.14a is the loss functions

Algorithm 4.8 2D loss function.

Input:
s
H
Th_l
Th_r
Th_v
Procedure:
Let n be the size of array s
loss $= 0$
Define the left vertical line using Th_l from $s[0].y$ to H at $s[0].x$
Define the right vertical line using Th_r from $s[n-1].y$ to H at $s[n-1].x$
Define the horizontal line using Th_v from $s[0].x$ to $s[n-1].x$ across H
for $i \leftarrow 0$ **to** $n-1$ **do**
    if $s[i].y$ is in between $s[0].y$ and H **and** $s[i].x < s[0].x + Th_l$ **then**
        $d_l \leftarrow |s[i].x - s[0].x|$
    end if
    if $s[i].y$ is in between $s[n-1].y$ and H **and** $s[i].x > s[n-1].x - Th_r$ **then**
        $d_r \leftarrow |s[i].x - s[n-1].x|$
    end if
    $d_v \leftarrow |s[i].y - H|$
    if both d_l and d_r exist **then**
        $loss \leftarrow loss + min(d_l, d_r)$
    else if d_l exists **then**
        $loss \leftarrow loss + d_l$
    else if d_r exists **then**
        $loss \leftarrow loss + d_r$
    else if d_v exists **then**
        $loss \leftarrow loss + d_v$
    else
        $loss \leftarrow loss + 1$ ▷ Other penalties like d_v, $1/n$ etc can be used as well.
    end if
end for
$loss \leftarrow loss/n$
Output: $loss$

for 4.14b. 4.14d is a short sublevel, and 4.14c represents a pseudo loss function for this sublevel using Equation 4.9 to maintain consistency for use cases that require a loss value.

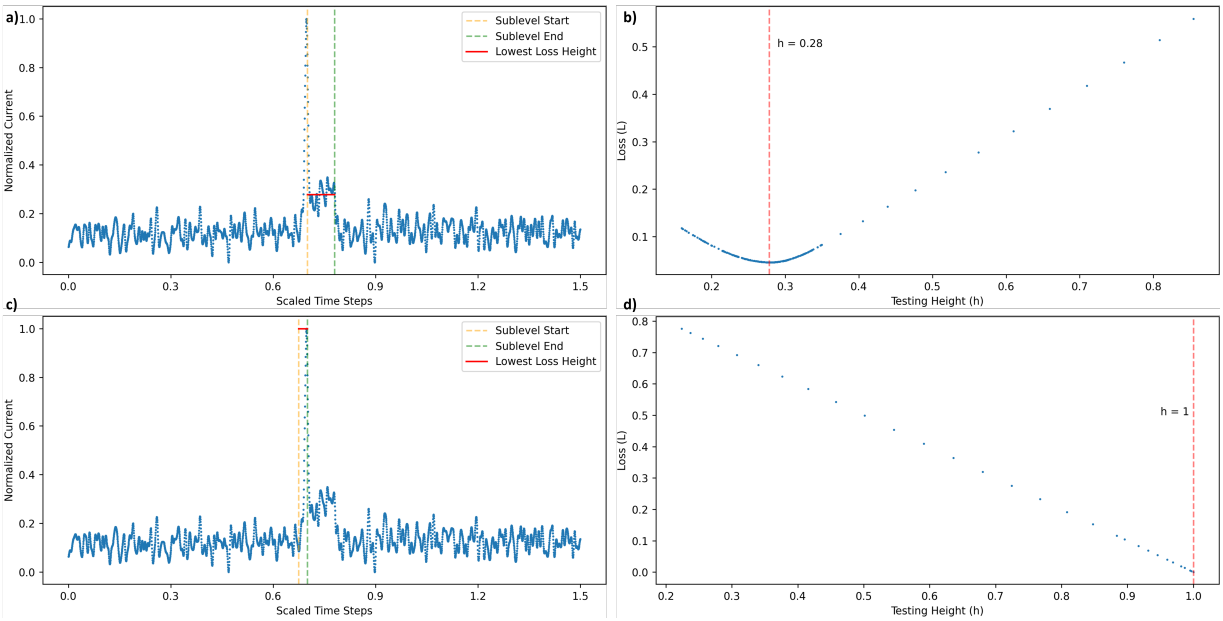

Figure 4.14: 2D Loss function application on sublevels from Section 5.2. a) Normal sublevel b) Loss function for various heights on normal sublevel with lowest loss at $h = 0.28$ c) Short sublevel d) Pseudo-loss function on short sublevel with lowest loss at the determined height using the extreme value in the sublevel at $h = 1$.

4.5 Summary

In this chapter, we introduced Nano Trees, a novel algorithm for fitting piecewise constant functions of nanopore (or any) data. We discussed how it works and the iterative approach it uses to fit the data using several passes. We went on to discuss the implementation details of Nano Trees using the Fitting Blocks Project and the generalizability of that approach. Finally, we concluded by exploring automation procedures for hyperparameter

tuning. We demonstrated how these automation techniques can enhance the efficiency and effectiveness of the approach by systematically adjusting certain hyperparameters to find the optimal configuration. In the next chapter, we will delve into the datasets used for our experiments and present the results obtained from applying Nano Trees to those synthetic and real datasets. We will present an overview of Nano Trees' ability to fit fast transients and narrow sublevels, which we previously discussed as something that other methodologies struggle to achieve effectively.

Chapter 5

Datasets and Results

This chapter discusses several datasets comprising both real and synthetic nanopore data on which the performance of Nano Trees is tested. These specific datasets have been chosen to cover a broad range of event shapes and use cases, highlighting the generality of the approach. Each section in this chapter is focused exclusively on a single dataset. For each dataset, the corresponding section begins with a detailed description, highlighting its source and relevance. We then detail the experimental setup used for that dataset, including the hyperparameters used for fitting and the metrics which will be used to judge the performance of Nano Trees on that dataset. Finally, we present the performance of Nano Trees n=on that dataset and provide a discussion to elaborate the results.

In this chapter, the primary comparison is made with CUSUM+ with appropriate and optimized user settings [11, 28]. The comparison is done only against CUSUM+ because this algorithm has been widely used in numerous research works in this field. The underlying principles of this algorithm have also been used in some form in other algorithms for nanopore sublevel fitting as discussed in Section 3. This is also the algorithm that is being

used in the lab for all experimentation, so we have a good understanding of its pros and cons, and the results of CUSUM+ are available on several datasets generated in the lab. We also tried to test out some other algorithms for the comparison, but for several of them either the implementation code was unavailable or unusable. However, it is important to emphasize that the comparison presented here is not solely intended to demonstrate the superiority of the Nano Trees algorithm over any other method. Instead, it highlights the inherent strengths of Nano Trees, which include their exceptional fitting accuracy, flexibility to fit any kind of sublevels, generalizability over various datasets and domains, and remarkable ability to fit fast transients or narrow sublevels. These qualities make the Nano Trees algorithm a robust and versatile tool in its own right, showcasing its potential for a wide range of applications beyond simply outperforming alternative methods.

5.1 Synthetic Data

We began the testing under controlled conditions of a synthetic dataset for which we know the true underlying signal shape, and we compared Nano Trees to the CUSUM+ algorithm which has been used for most data analysis over the past several years. This data was generated using the description in Chapter 1 and Chapter 2 for the anatomy of a nanopore signal, with event sublevels chosen to represent the most common difficult-to-fit motifs found in nanopore signals in the literature. Any nanopore event can be constructed as a linear combination of these basic subevents. These motifs are shown in the top row of Figure 5.1 and correspond to the following classes:

1. Class 1: Event consisting of a single transient symmetric peaked sublevel
2. Class 2: Event with transient symmetric peaked sublevel (asymmetric peak) at the

beginning

3. Class 3: Event with transient symmetric peaked sublevel (symmetric peak) in the middle
4. Class 4: Event with asymmetric transient peaked sublevel at the end
5. Class 5: Event with a transient sloped sublevel at the beginning
6. Class 6: Event with a transient sloped sublevel at the end
7. Class 7: Event containing a double peaked sublevel pair with transient separation

This dataset contains 500 events per true duration per class for 4500 events per class and 31500 events total. The events are generated with a sampling rate of 4.167 MHz filtered at 1MHz using a 4-pole Bessel filter after adding white noise with 50 pArms.

The experimental setup is straightforward, the entire dataset was fitted using CUSUM+ and Nano Trees. The hyperparameters discussed in Appendix B were tuned from their default values mentioned in Appendix B.2 using the debugging steps mentioned in Appendix B.4. After tuning, the final set of hyperparameters used for this experiment are mentioned in Table 5.1.

To judge the results, we compared 3 distinct metrics of fit quality - shape accuracy, transient sublevel duration error, and transient sublevel blockage error; with consistent hyperparameters for all signal classes as mentioned in Table 5.1. The first metric is shape accuracy, which simply considers whether the sublevel structure of the fit is correct. To pass, a fit must have the correct number of sublevels, in the correct order with respect to depth, and have an error in the fitted blockage depth of the transient level not exceeding three standard deviations of the baseline noise. In Figure 5.1, results that are closer to

Table 5.1: Nano Trees hyperparameters used to fit the synthetic dataset.

Pass	Hyperparameter	Value
1	approxSubLevelEstimate	10
1	adaBoostRegressorNEstimators	800
2	approxSubLevelEstimate	10
3	numberOfStdAboveAndBelow	2.5
3	oneSidedPercentParity	0.2
3	minDataPointsToBeBoosted	2% event length
3	exceptionalHeightBaseMaxDiffForHeightRefresh	0.35
4	minDataPointsToBeSubLevel	8
4	numberOfStdAboveAndBelow	2
4	exceptionalPeak_MinHeightStdAboveAndBelow	3
4	exceptionalPeak_WidthLowerBound	8
4	exceptionalPeak_BaseDifferenceStdAtleast	0
4	exceptionalSlope_MinHeightStdOfMinDiff	∞
4	exceptionalSlope_WidthLowerBound	∞
5	numberOfStdAboveAndBelow	3.2
6	baselineStdThreshold	1.5
-	directionalThreshold	0.5
-	shortSublevelDefinition	16

100% for shape accuracy indicate a high-quality fit. The second metric is duration error, which is the error in the estimate of the duration of the transient part of the signal for events deemed to have the correct shape, as a multiple of the system rise time. The third metric is blockage error, which is the error in the estimate of the blockage level for the transient part of the signal for events deemed to have the correct shape, as a multiple of the standard deviation of the baseline noise. For rows 3 and 4 (duration error and blockage error), being close to 0 is desirable, as it indicates minimal error in the duration and blockage depth estimates. Thus, good results are characterized by high values for shape accuracy (close to 100%) and low values for duration and blockage errors (close to 0), demonstrating the effectiveness of the fitting process for a given class and true duration.

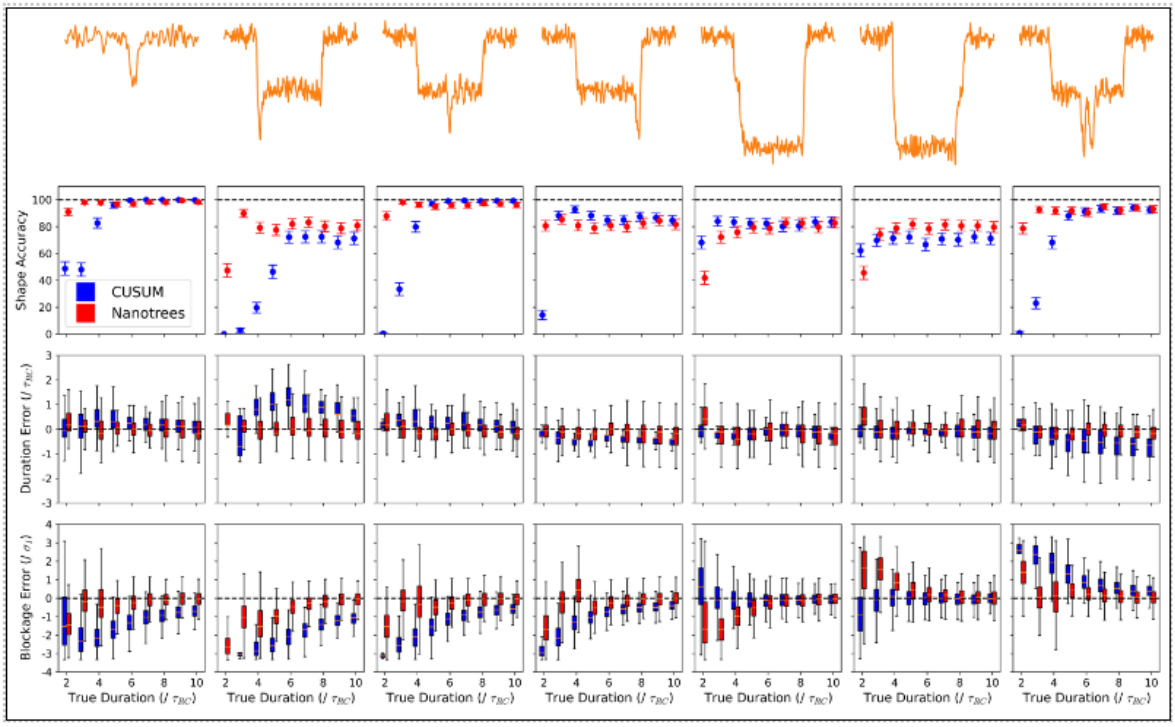

Figure 5.1: Comparing Nano Trees to CUSUM+ on a synthetic dataset. From left to right, row 1 presents a representative example of the 7 classes used in this dataset. Inside each column, from left to right, the plots present the comparison for each class with an increasing true duration for each class from 2-10.

Analyzing the results from Figure 5.1, it is immediately clear that Nano Trees outperforms CUSUM+ significantly when transients are shorter than 4 times the system rise time. Even when transients reach a steady state, Nano Trees still slightly outperforms or approximately matches the performance of CUSUM+ in most cases. Like shape accuracy, Nano Trees outperforms CUSUM+ for short transients without sacrificing performance on longer ones. It is worth noting that both algorithms underestimate true blockage depth to increasing degrees as the transient duration approaches zero, which is to be expected given systemic distortion, though this effect is suppressed in Nano Trees compared to CUSUM+.

Of note, the sloping sublevels (classes 5 and 6) display a clear directional bias, with Nano Trees underestimating or overestimating the blockage depth depending on whether the event is sloping upwards or downwards to a greater degree than CUSUM+. This is likely because the transients are themselves asymmetric and can be improved using more sophisticated algorithms to estimate the depth of sloping sublevels. This is a topic of ongoing study and improvement in the analysis framework as discussed in Section 4.4.

For this dataset, metrics that utilize global deviations are not used because they disproportionately penalize outliers, which may not accurately represent the overall data quality. These metrics often assume that errors are symmetrically distributed, but in this case, the error distribution is asymmetric. Smaller deviations arising from the correct or missing fit on the data especially on the special features of each class would have been missed in that approach. Therefore, an alternative approach focusing on the correctness, width and height of these special features in each class have been used to judge fit quality.

These signals provide clear insight into the limitations and types of errors that arise when using Nano Trees for fitting and highlight the improvements available over incumbent analysis methods in the regime of fast transients. The insights from this synthetic data are critical to inform an evaluation of the quality of fits to real datasets and to understand the strengths and limitations of the approach.

5.2 DNA molecule attached to a linear DNA carrier

Nano Trees is built to be used in practical situations, hence the next dataset we apply Nano Trees on is a real dataset developed by Roelen et al. [69]. The dataset comprises events generated by the translocation of a biomolecule containing two parts - a long dsDNA of variable length referred to as "carrier", and a relatively short yet bulky DNA nanostructure

referred to as the "cargo", that has been reversibly attached to one end of the carrier, through a nanopore. The fitting was limited to non-rejected events as classified by the original authors of this dataset [69]. This reduced the number of events to be considered from 75,801 events to just 18,028 events. A sample of the majority of those events is shown in Figure 5.2 (reproduced with permission [69]).

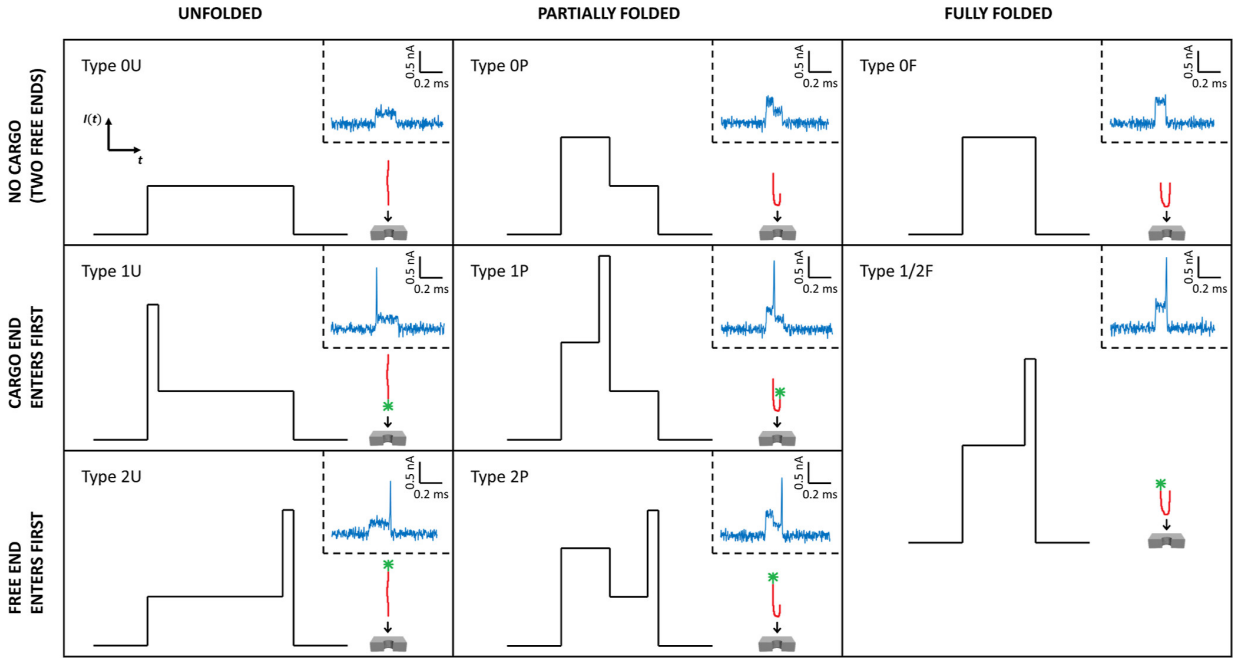

Figure 5.2: A representation of what most of the events in the dataset look like, broadly classified into 8 distinct classes [69]. The red line represents the dsDNA carrier and the green star represents the cargo attached to it.

Reprinted with permission from Roelen, Zachary, Kyle Briggs, and Vincent Tabard-Cossa. "Analysis of nanopore data: classification strategies for an unbiased curation of single-molecule events from DNA nanostructures." ACS sensors 8.7 (2023): 2809-2823 [69]. Copyright 2023 American Chemical Society.

The fitting was performed on these events using the hyperparameters mentioned in Table 5.2. As this is a real dataset, the ground truth of what the piecewise constant sublevels would be, for any event, does not exist. Hence, we randomly sampled a subset of these events from the reduced dataset to manually inspect the fit quality. The selection

was not entirely random, we utilized the CUSUM+ fits and Nano Trees fits to create three broad categories - events that had a different number of sublevels in both fits, events that had the same number of sublevels but had a mismatch in sublevel order, relative height etc., and the remaining events. From these three categories, events were almost equally sampled to a total of 3000 events. An impartial 3rd party was used to manually assess the quality of fitting on these events, and the corresponding results of this assessment can be found in Table 5.3. The labelling process involved two responses per event: one "Yes" or "No" for the Nano Trees fit and another "Yes" or "No" for the CUSUM+ fit, indicating whether each fit was deemed acceptable for that specific event. The final tallied result provided a fitting accuracy for each algorithm, with a higher percentage of "Yes" responses indicating better performance for that algorithm on the dataset.

As shown in Table 5.3 row 1, Nano Trees outperformed CUSUM+ in the number of events the blind and impartial 3rd party labelled as correct fits. Row 2 in Table 5.3 are the events for which the reviewer labelled either the CUSUM+ fit or the Nano Trees fit as bad, which means "No" for both fits. These events can be considered as hard events, as at least one of the algorithms fails to produce a good fit for that event. So, out of those 1350 events from the original 3000, Nano Trees was still able to outperform CUSUM+ by a significant margin. The results indicate that Nano Trees not only exhibit superior fitting ability compared to CUSUM+ but also showcase a strong general capability to fit piecewise constant functions on real nanopore data, suggesting broad effectiveness in this task.

Figure 5.3 presents another point of view to observe and compare the results of this experiment. This figure is a scatter plot for all the sublevels in the entire dataset for the events that had an equal number of sublevels in the Nano Trees fit and CUSUM+ fit, excluding baseline sublevels. It can be observed that the vertical line in Figure 5.3

Table 5.2: Nano Trees hyperparameters used to fit the cargo-carrier dataset [69].

Pass	Hyperparameter	Value
1	approxSubLevelEstimate	4
1	adaBoostRegressorNEstimators	500
2	approxSubLevelEstimate	5
3	numberOfStdAboveAndBelow	2.1
3	oneSidedPercentParity	0.2
3	minDataPointsToBeBoosted	2% event length
3	exceptionalHeightBaseMaxDiffForHeightRefresh	1
4	minDataPointsToBeSubLevel	2% event length
4	numberOfStdAboveAndBelow	2
4	exceptionalPeak_MinHeightStdAboveAndBelow	3
4	exceptionalPeak_WidthLowerBound	0.2% event length
4	exceptionalPeak_BaseDifferenceStdAtleast	0.5
4	exceptionalSlope_MinHeightStdOfMinDiff	2.2
4	exceptionalSlope_WidthLowerBound	∞
5	numberOfStdAboveAndBelow	1
6	baselineStdThreshold	1.5
-	directionalThreshold	0.5
-	shortSublevelDefinition	4% event length

Table 5.3: Fitting results on cargo-carrier dataset [69].

	Count	Nano Trees accuracy	CUSUM+ accuracy
Categorically Sampled Dataset	3000	83.33%	70.47%
Difficult Events	1350	62.96%	34.37%

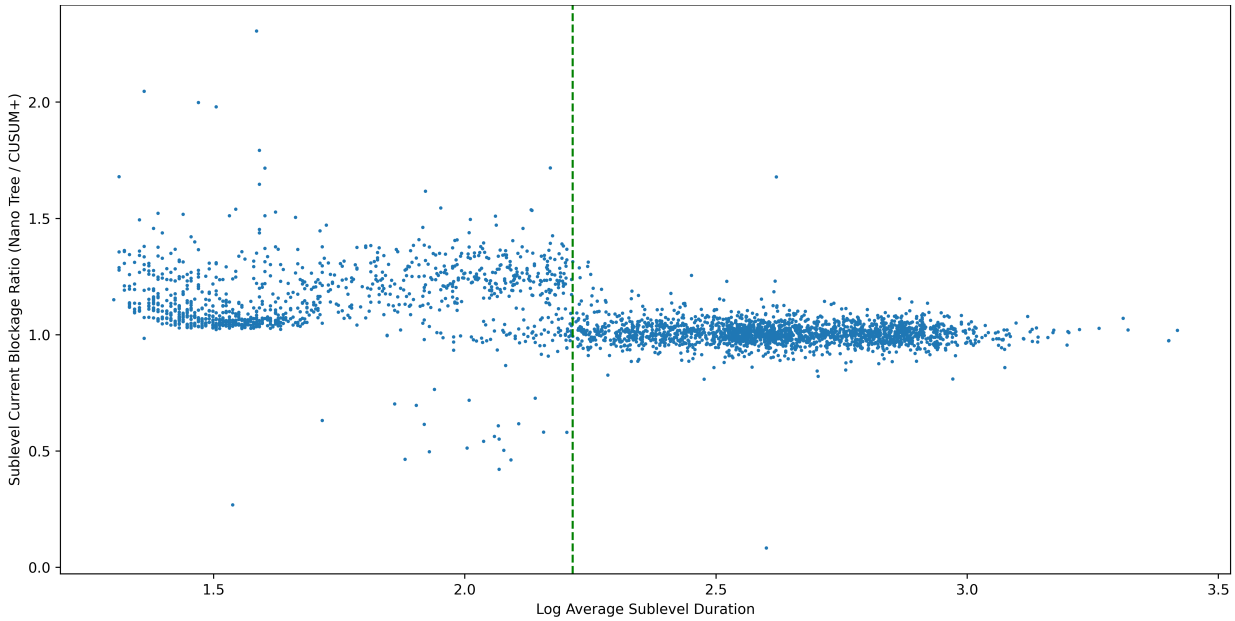

Figure 5.3: Scatter plot of all sublevels from the events that have an equal number of sublevels in CUSUM+ and Nano Trees fits (excluding baseline sublevels). The X-axis represents the log of average between the sublevel duration estimate of Nano Trees fit and CUSUM+ fit. The Y-axis represents the ratio of sublevel current blockage for these sublevels by Nano Trees and CUSUM+. The vertical line corresponds to 4 times the rise time of the experiment (calculated numerically as 41).

corresponds to 4 times the rise time of the experiment which almost clearly segregates the plot into 2 distinct regions. The points to the right of the 4-times rise-time line can be considered as normal sublevels (as described in Section 4.2.5). In this region, the ratio of heights is almost close to 1, which means that both Nano Trees and CUSUM+ almost agree on the height estimate of these sublevels.

The region of interest here is to the left of the 4-times rise-time line. The points here correspond to sublevels that are faster than the 4-times rise time of the system. It can be observed that the majority of these points do not lie around 1. As in the ratio on the Y-axis, the Nano Tree height estimate is in the numerator, it can be noted that for sublevels in this region, Nano Trees consistently produces a higher height estimate. Since Nano Trees selects the most extreme available data point for sublevels in this region, which is closest to the ground truth, Nano Trees fits can be considered more accurate, especially for sublevels in this region. As this is the region where CUSUM+ and other methodologies struggle to fit sublevels accurately [11, 28], we can conclude that Nano Trees is successfully able to improve the fitting for these fast transients indicating its effectiveness in capturing the rapid changes in signal dynamics more accurately.

5.3 DNA Nanostructure Barcodes

To demonstrate the utility of this approach in the context of a real nanopore experiment, we apply Nano Trees to analyze a dataset for which our existing CUSUM+ framework fails consistently. This data arises from the translocation of a complex molecule that encodes information in the form of a double-stranded DNA backbone with binding sites to which side chains can bind through DNA hybridization. These side chains carry a DNA nanostructure—a star with either 4 or 12 dsDNA arms [39], which produce clearly

Table 5.4: Nano Trees hyperparameters used to fit DNA Nanostructure Barcodes datasets.

Pass	Hyperparameter	Value
1	approxSubLevelEstimate	4
1	AdaBoostRegressorNEstimators	500
2	approxSubLevelEstimate	4
3	numberOfStdAboveAndBelow	2
3	oneSidedPercentParity	0.2
3	minDataPointsToBeBoosted	20
3	exceptionalHeightBaseMaxDiffForHeightRefresh	0.35
4	minDataPointsToBeSubLevel	60
4	numberOfStdAboveAndBelow	2
4	exceptionalPeak_MinHeightStdAboveAndBelow	3
4	exceptionalPeak_WidthLowerBound	10
4	exceptionalPeak_BaseDifferenceStdAtleast	0
4	exceptionalSlope_MinHeightStdOfMinDiff	2.2
4	exceptionalSlope_WidthLowerBound	∞
5	numberOfStdAboveAndBelow	4.5
6	baselineStdThreshold	1.5
-	directionalThreshold	0.5
-	shortSublevelDefinition	80

distinguished blockage levels when they pass through a nanopore and represent a digital 0 or 1, respectively. Events for which the 12-arm star passes the pore first are classified as "10 events" while events for which the 4-arm star passes the pore first are classified as "01 events". Because of the need to achieve high storage density, these chains are quite small and result, in the ideal case, in symmetric peaked sublevels of differentiated depth and of short duration relative to the rise time of our measurement electronics. These transient levels are too fast for CUSUM+ to consistently recognize, let alone fit accurately, as shown in Figure 5.4.

Judging the fit quality is not a straightforward approach for data that is unlabeled. The true fits for this data are not available because that is what we are trying to do using these

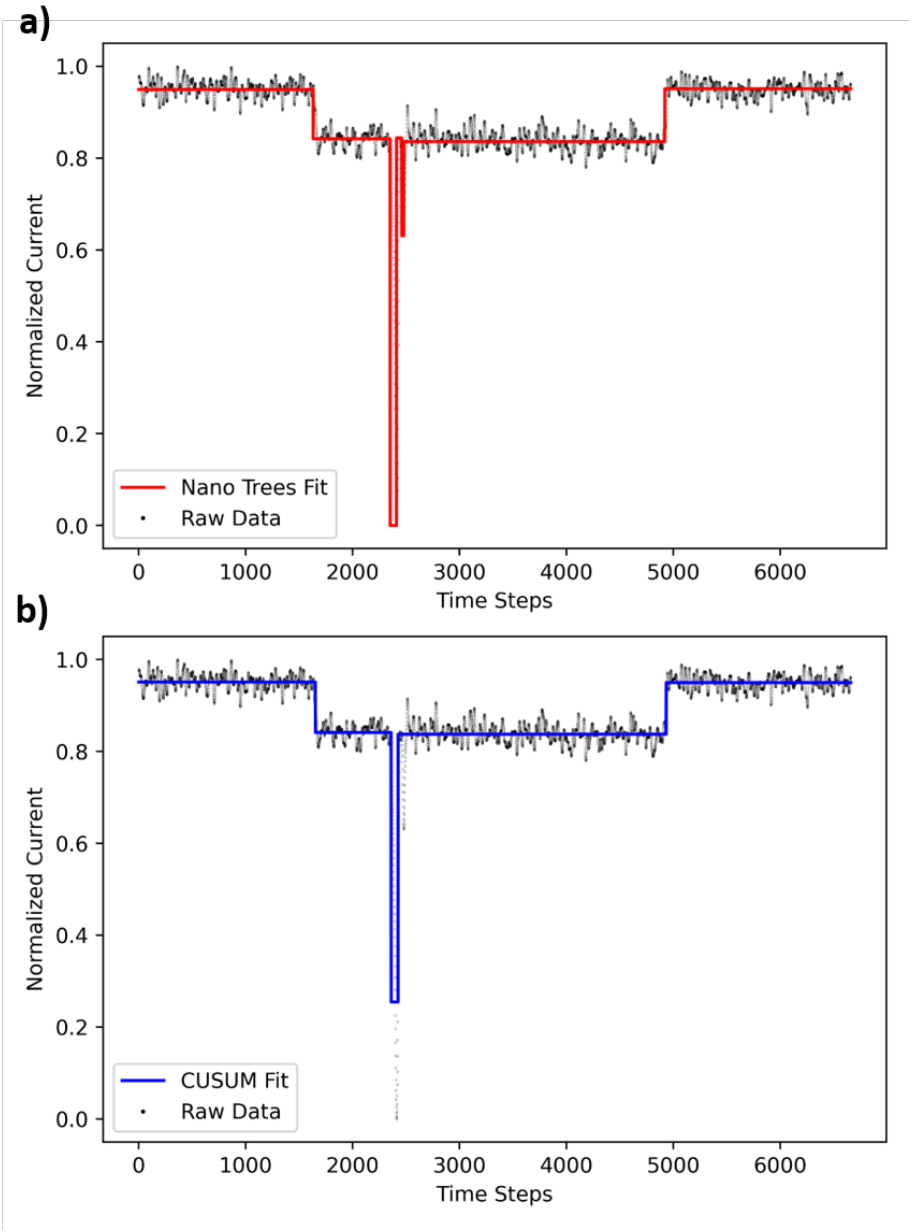

Figure 5.4: a) Nano Trees fit on DNA Nanostructures Barcodes event b) CUSUM+ fit on DNA Nanostructure Barcodes event.

Table 5.5: Accuracy results on DNA Nanostructures Barcodes dataset.

	Nano Trees fit accuracy	CUSUM+ fit accuracy
10 Dataset	94.55%	59.18%
01 Dataset	89.65%	48.96%
Combined	92.12%	54.10%

two fitting algorithms. Therefore, to judge the performance of Nano Trees and CUSUM+ on this dataset, we separately created fits for both algorithms. The experiment to generate these events is still in development phase, so most of the events generated do not contain the desirable "0" and "1" bits. Thus, a basic selection filter was applied to weed out bad events, and on the remaining events, the fits were generated. The 01 dataset contains 145 events and the 10 dataset contains 147 events, adding up to a total of 292 events in this dataset.

Both datasets are fitted using the same set of hyperparameters mentioned in Table 5.4 tuned according to the steps mentioned in Appendix B. For inspecting the fits, an impartial judge manually went through the entire dataset to label for each event whether Nano Trees fit is good, CUSUM+ fit is good, both fits are good or none of the fits are good. The results of this experiment are displayed in Table 5.5 in terms of percentage. The last row in Table 5.5 contains the average of the two subsets for both algorithms.

As evident from Table 5.5 Nano Trees outperform CUSUM+ on this dataset. For both 10 and 01 subset datasets Nano Trees has a better fit accuracy than CUSUM+. Factoring in a degree of error for manual labelling, still, the fit accuracy of Nano Trees is significantly higher than CUSUM+, demonstrating the superior performance that Nano Trees has to offer in fitting piecewise constant functions on nanopore data.

With all the great results provided by Nano Trees, it is also important to highlight some

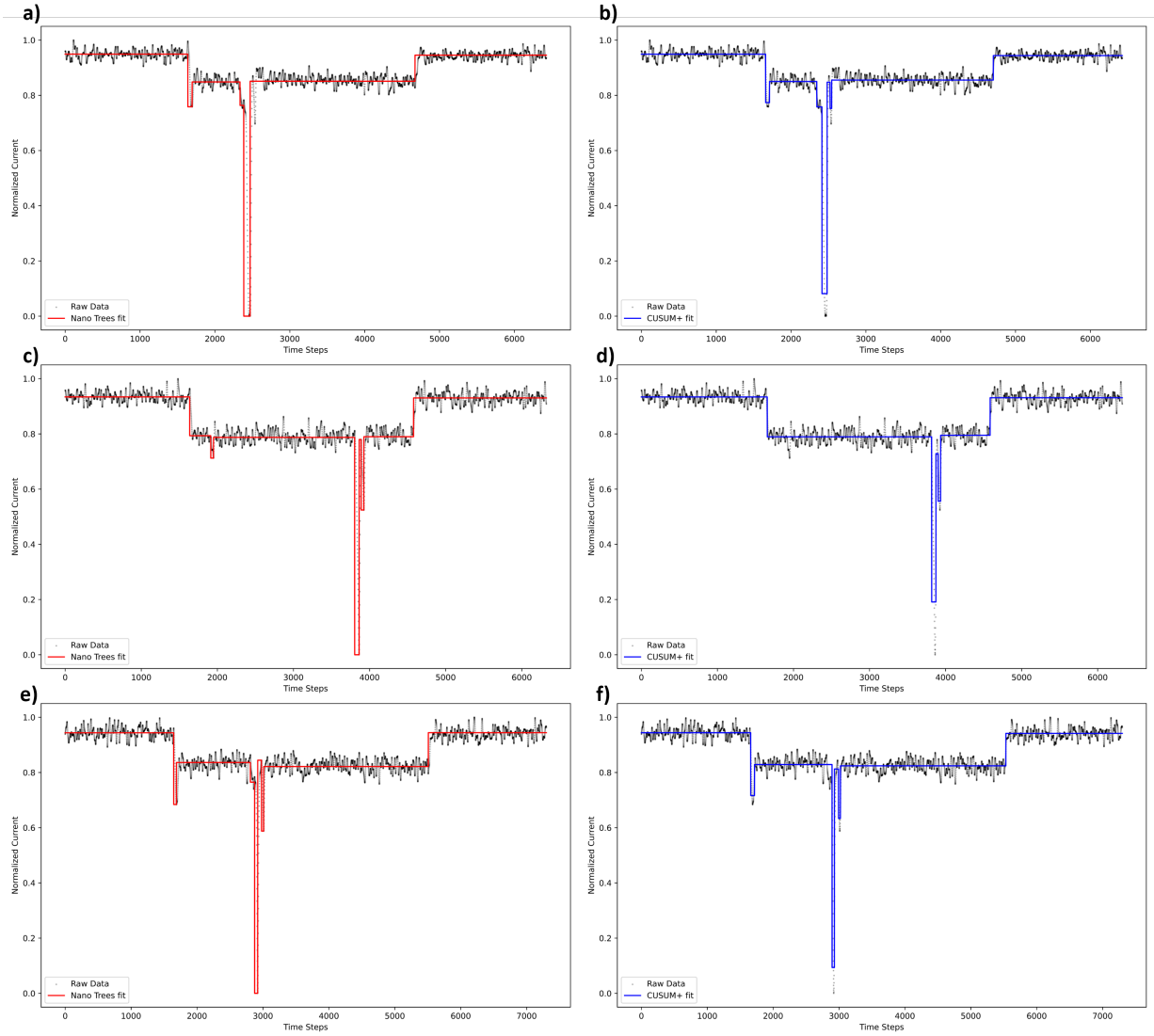

Figure 5.5: Examples of events for which CUSUM+ fit was labelled correct and Nano Trees fit was labelled wrong.

of the shortcomings of this algorithm. In the combined dataset, there were only 11 events (8 events from the 01 dataset and 3 events from the 10 dataset) for which the CUSUM+ fit was labelled as correct, and the Nano Trees fit was labelled as wrong. Figure 5.5 includes a few examples of these events. Figure 5.5a shows a Nano Trees fit with a missing peak, which gets fitted correctly in the CUSUM+ fit as shown in Figure 5.5b. Whereas Figure 5.5c shows a Nano Trees fit with an extra sublevel fitted in the beginning. This extra sublevel should not have been present in the final fit. CUSUM+ correctly skips this sublevel as shown in Figure 5.5d. Upon analysis, it was found that the initial passes of overfitting made these sublevels in Figure 5.5a and Figure 5.5c slightly outside and slightly inside the threshold. These fits can be improved by adjusting the hyperparameters but it is possible that the chain effect of this change could negatively affect the overall fit. There is also no way for the user to specify during Nano Trees fit, that we only expect 2 bits, a "1" and a "0" in the final fit. This causes bad fits like these to occur and are hard to fix.

Figure 5.5e is another example of Nano Trees going wrong with the fits. The slope sublevel next to the large peak should not have been fitted, as shown in Figure 5.5f. This issue can be resolved by adjusting the hyperparameters that control the exceptional slope sublevel. Alternatively, the slope sublevels can be entirely turned off in the Nano Trees fit, but the downsides of doing this outweigh the benefits as discussed before. It is important to note that, while these examples showcase specific shortcomings, Nano Trees generally outperforms CUSUM+ on a broader scale. It is evident from Table 5.5 and the fact that there are 122 events (67 events in the 01 dataset and 55 events in the 10 dataset) in which the CUSUM+ fit was labelled as wrong while Nano Trees were labelled as correct. These examples demonstrate that while Nano Trees algorithm generally performs well, there are instances where it may either miss critical sublevels or introduce unnecessary ones, leading to inaccurate fits. However, these cases are exceptions rather than the norm, and they

highlight opportunities for refining the algorithm to enhance its accuracy and reliability.

5.4 Summary

In this chapter, we examined three nanopore datasets, encompassing both real and synthetic data. The first dataset was a synthetic one, created to analyze the Nano Trees algorithm's performance in a controlled environment with known ground truth. The second dataset used in this study, by Roelen et al. [69] and the third dataset, related to DNA Nanostructure Barcodes, were real datasets produced by actual translocation of specific molecules through a nanopore. They were used to demonstrate the Nano Trees algorithm's performance in real-world practical situations.

The results showed that Nano Trees consistently delivered accurate and reliable performance across all three datasets and highlighted a few shortcomings of the algorithm. For the synthetic dataset, Nano Trees successfully identified the expected patterns with high shape accuracy, low duration error and low blockage error in almost all cases. For the real datasets, Nano Trees was able to handle the complexities and variabilities inherent in real nanopore data. This performance underscores Nano Trees' robustness and versatility in handling both synthetic and real nanopore data. The algorithm's ability to deliver consistent results across diverse datasets highlights its potential for widespread application in nanopore sequencing and related fields.

The next chapter will provide a comprehensive conclusion of the thesis, summarizing the key contributions, discussing areas of improvement, and outlining potential directions for future research.

Chapter 6

Conclusion

In this thesis, we focused on signal processing of nanopore signals. The signal generation through a nanopore was briefly discussed and also through the point of view of someone with no background in this field. We then went on to discuss what piecewise constant function fitting is, why is it required, and what challenges we face in the process. The primary goal of this research was to do this fitting in an easy, robust, and accurate manner. Several researches that try to perform this fitting on nanopore and other fields were discussed along with the CUSUM+ algorithm which we use as a comparison metric for our results. We developed Nano Trees as a piecewise constant function fitting algorithm specifically to fit nanopore data, which is general enough to adapt to any kind of signal even from other domains. Nano Trees is a strong yet complicated algorithm that depends on several hyperparameters. We discussed its implementation and several automations to make the process of selecting the optimal hyperparameters better.

Using Nano Trees to fit events from several physically generated and synthetic datasets, we discuss the results to evaluate the performance of Nano Trees. We observe statistical

measures to analyze their effectiveness in accurately modelling the data and compare these results to existing methods. Although Nano Trees receives better scores on all the datasets used in this study (discussed in Section 5) they are not cross-comparable. But it can be stated about Nano Trees based on observation, that they perform better when there are fewer different types of events to fit. While the algorithm is highly adaptable to different types of events, selecting hyperparameters is easier when there are fewer types of events to fit within a single dataset using the same set of hyperparameters. That being said, with well-chosen hyperparameters, the fitting accuracy remains high regardless of the number of different event types. However, it is always preferable to have fewer event types, as this simplifies the hyperparameter selection process and further enhances fitting accuracy.

This chapter concludes this thesis by discussing the primary contributions of this research and exploring potential improvements and future applications.

6.1 Contributions

In this thesis, we have made several significant contributions, which are outlined as follows:

- **Nano Trees algorithm:** We created the Nano Trees algorithm for piecewise constant function fitting of nanopore data. The algorithm is robust to fast transients and can fit them quite well. It is generalizable enough to adapt to any dataset by correctly tuning the hyperparameters, making it a versatile tool for various applications. It is a novel algorithm that uses machine learning to get an overfit on the data and then reduces those sublevels step by step to get the final fit. The robustness of Nano Trees in handling fast transients addresses a significant challenge in nanopore data analysis, where traditional methods often fall short. Overall, Nano Trees stands

out as a significant advancement in the field of nanopore data analysis. Its innovative approach, combined with its robust handling of fast transients and generalizability, makes it a powerful tool for sublevel extraction in this field.

- **Fitting Blocks project:** The code of Nano Trees has been implemented in a block fashion where each block is separately customizable, replaceable, and reusable. This architecture separates different functionalities used to produce a fit into separate blocks and can hence be used to implement other algorithms as well. We built this structure in this way to allow for easy and efficient changes to accommodate the various iterations the original algorithm went through to evolve to its current state and be called Nano Trees. Future changes to Nano Trees or the development of a different algorithm can be done using the Fitting Blocks project by reusing most of the code blocks, minimizing redundancies and maximizing efficiency.

- **Nano Trees automation:** The Nano Trees algorithm in its current state relies on several hyperparameters which is also one of the reasons that makes it flexible enough to adapt to various datasets. We created several automation tools to reduce the workload of a user while tuning these hyperparameters. These automations not only make the process of hyperparameter tuning faster but also better. They can be either used as is or as a starting point for further fine-tuning.

- **Extensive experimental comparison of Nano Trees and CUSUM+ on several datasets:** We experiment on several real and synthetic nanopore datasets to compare the piecewise constant function fitting ability of Nano Trees. We do so by comparing it with CUSUM+ using numerical statistics and manual human validation. Our results demonstrate that Nano Trees not only outperforms CUSUM+ but also exhibits superior performance by capturing the underlying structure of the

data with remarkable accuracy, reflected in its numerical superiority across various datasets. This numerical advantage is evident in the accuracy percentage and higher fidelity to the true signal when compared to CUSUM+. These quantitative measures highlight the robustness of Nano Trees in handling the complexities inherent in nanopore datasets. Moreover, Nano Trees shows a significant proficiency in fitting short transients, which remains a challenging aspect in the nanopore analysis field.

As a result of these contributions, we achieved the following:

- Research paper publication on Nano Trees - Deekshant Wadhwa, Philipp Mensing, James Harden, Paula Branco, Vincent Tabard-Cossa, and Kyle Briggs. Nano trees: Nanopore signal processing and sublevel fitting using decision trees. Submitted, 2024.
- Patent on the Nano Trees algorithm- Deekshant Wadhwa, Kyle Briggs, Vincent Tabard-Cossa, and Paula Branco. Pre-processing nanopore signals. Patent Application, 2024.

6.2 Areas of Improvement and Future Work

Nano Trees is a combination of specific functions in a specific order that can be optimized individually or combinedly. The first improvement would be to implement Nano Trees along with the automation discussed in Section 4.4. These automation were developed after obtaining the fits on all datasets, hence they were not used while obtaining current results. Nano Trees in its current state does not have these automation in use, so a new implementation of Nano Trees can be developed using these automations to improve the overall performance and the time taken for fine-tuning the hyperparameters (or as a starting point).

Another significant improvement would be to use parallel processing in Nano Trees. The fitting of an event does not rely on any other event or fit in the dataset. While there are shared stages in pre-processing and post-processing, the core task of fitting individual events, which is the most time-consuming part, can be parallelized based on the number of processors available. This will reduce the computation time and significantly boost the performance of the algorithm.

As far as reimplementation goes, the current version of Nano Trees has been implemented in Python, a high-level language that, while user-friendly and versatile, is not as fast as low-level programming languages such as C, C++, or Rust. Thus, a reimplementation in one of these faster languages could provide a substantial performance boost.

The exceptional parameters, as outlined in Section 4.2.5, used in Nano Trees contribute significantly to the algorithm's versatility. However, upon deeper analysis, these parameters only allow for three types of sublevels, resulting in three distinct clusters in the blockage depth versus dwell time plot. This can be improved to allow for multiple regions for the sublevels to exist. Although this would require a larger number of unknown hyperparameters, this challenge could potentially be addressed by integrating another machine-learning model capable of autonomously extracting clusters corresponding to physically probable sublevels.

The ability of Nano Trees to fit fast transients opens several opportunities for future research, using Nano Trees to fit these short sublevels accurately. Several applications like protein sequencing, biomolecule fingerprinting, event classification etc., which involve the use of these short sublevels to be accurately fitted can be greatly improved by this approach. Allowing for novel applications in single-molecule studies, custom biosensors, high-bandwidth signal processing and much more, from fundamental research to practical implementations Nano Trees has the potential to revolutionize not only the field of bio-

science but also contribute to broader fields beyond it. The findings presented here lay the groundwork for further research and pave the way for new discoveries.

References

[1] Nima Arjmandi, Willem Van Roy, Liesbet Lagae, and Gustaaf Borghs. Improved algorithms for nanopore signal processing. *arXiv preprint arXiv:1207.2319*, 2012.

[2] Kevin Arvai. kneed. https://github.com/arvkevi/kneed, August 2020. Version 0.7.0, Accessed: 2024-07-07.

[3] Arvind Balijepalli, Jessica Ettedgui, Andrew T. Cornio, Joseph W. F. Robertson, Kin P. Cheung, John J. Kasianowicz, and Canute Vaz. Quantifying short-lived events in multistate ionic current measurements. *ACS Nano*, 8:1547–1553, 2 2014.

[4] Arvind Balijepalli, Jessica Ettedgui, Andrew T. Cornio, Joseph W. F. Robertson, Kin P. Cheung, John J. Kasianowicz, and Canute Vaz. Correction to quantifying short-lived events in multistate ionic channel measurements. *ACS Nano*, 9:12583–12583, 12 2015.

[5] YM Nuwan DY Bandara, Jugal Saharia, Buddini I Karawdeniya, Patrick Kluth, and Min Jun Kim. Nanopore data analysis: baseline construction and abrupt change-based multilevel fitting. *Analytical Chemistry*, 93(34):11710–11718, 2021.

[6] Piero Barone and Riccardo March. Reconstruction of a piecewise constant function from noisy fourier coefficients by padé method. *SIAM Journal on Applied Mathematics*, 60(4):1137–1156, 2000.

[7] Farhad Bayat, Tor Arne Johansen, and Ali Akbar Jalali. Flexible piecewise function evaluation methods based on truncated binary search trees and lattice representation in explicit mpc. *IEEE Transactions on Control Systems Technology*, 20(3):632–640, 2011.

[8] Nicholas A. W. Bell and Ulrich F. Keyser. Digitally encoded dna nanostructures for multiplexed, single-molecule protein sensing with nanopores. *Nature Nanotechnology*, 11:645–651, 7 2016.

[9] Leif Bergerhoff, Joachim Weickert, and Yehuda Dar. Algorithms for piecewise constant signal approximations. In *2019 27th European Signal Processing Conference (EUSIPCO)*, pages 1–5. IEEE, 2019.

[10] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification And Regression Trees*. Routledge, 10 2017.

[11] Kyle Briggs. Solid-state nanopores: fabrication, application, and analysis. *Ph. D. dissertation*, 2018.

[12] Kyle Briggs, Harold Kwok, and Vincent Tabard-Cossa. Automated fabrication of 2-nm solid-state nanopores for nucleic acid analysis. *Small*, 10(10):2077–2086, 2014.

[13] Tom Z Butler, Mikhail Pavlenok, Ian M Derrington, Michael Niederweis, and Jens H Gundlach. Single-molecule dna detection with an engineered mspa protein nanopore. *Proceedings of the National Academy of Sciences*, 105(52):20647–20652, 2008.

[14] C. Callegari, S. Giordano, M. Pagano, and T. Pepe. Wave-cusum: Improving cusum performance in network anomaly detection by means of wavelet analysis. *Computers & Security*, 31:727–735, 7 2012.

[15] Huifen Chen and Bruce Schmeiser. I-smooth: iteratively smoothing piecewise-constant poisson-process rate functions. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 469–480. IEEE, 2011.

[16] Kaikai Chen, Jinbo Zhu, Filip Bošković, and Ulrich F. Keyser. Nanopore-based dna hard drives for rewritable and secure data storage. *Nano Letters*, 20:3754–3760, 5 2020.

[17] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[18] Dongkyoung Choe, Stephen L Campbell, and Ramine Nikoukhah. Optimal piecewise-constant signal design for active fault detection. *International Journal of Control*, 82(1):130–146, 2009.

[19] Vincent Cohen-Addad and Varun Kanade. Online optimization of smoothed piecewise constant functions. In *Artificial Intelligence and Statistics*, pages 412–420. PMLR, 2017.

[20] Naren Das, Bhaswati Chakraborty, and Chirasree RoyChaudhuri. A review on nanopores based protein sensing in complex analyte. *Talanta*, 243:123368, 6 2022.

[21] Naren Das, N Mandal, Praveen Kumar Sekhar, and Chirasree RoyChaudhuri. Signal processing for single biomolecule identification using nanopores: a review. *IEEE Sensors Journal*, 21(11):12808–12820, 2020.

[22] David Deamer, Mark Akeson, and Daniel Branton. Three decades of nanopore sequencing. *Nature Biotechnology*, 34:518–524, 5 2016.

[23] Michael Dreher and Ansgar Jüngel. Compact families of piecewise constant functions in lp (0, t; b). *Nonlinear Analysis: Theory, Methods & Applications*, 75(6):3072–3077, 2012.

[24] Harris Drucker. Improving regressors using boosting techniques. In *Icml*, volume 97, page e115. Citeseer, 1997.

[25] Rayane El Sibai, Yousra Chabchoub, Raja Chiky, Jacques Demerjian, and Kablan Barbar. An in-depth analysis of cusum algorithm for the detection of mean and variability deviation in time series. In *International Symposium on Web and Wireless Geographical Information Systems*, pages 25–40. Springer, 2018.

[26] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[27] Eugene F Fama. The behavior of stock-market prices. *The journal of Business*, 38(1):34–105, 1965.

[28] Jacob H. Forstater, Kyle Briggs, Joseph W.F. Robertson, Jessica Ettedgui, Olivier Marie-Rose, Canute Vaz, John J. Kasianowicz, Vincent Tabard-Cossa, and Arvind Balijepalli. Mosaic: A modular single-molecule analysis interface for decoding multistate nanopore data. *Analytical Chemistry*, 88:11900–11907, 12 2016.

[29] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 8 1997.

[30] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[31] Björn Geißler, Alexander Martin, Antonio Morsi, and Lars Schewe. Using piecewise linear functions for solving minlps. In *Mixed integer nonlinear programming*, pages 287–314. Springer, 2011.

[32] Ian Goodfellow. Deep learning, 2016.

[33] Melody S Goodman, Yi Li, and Ram C Tiwari. Detecting multiple change points in piecewise constant hazard functions. *Journal of applied statistics*, 38(11):2523–2532, 2011.

[34] Ardeshir Goshtasby. Piecewise cubic mapping functions for image registration. *Pattern Recognition*, 20(5):525–533, 1987.

[35] Pierre Granjon. The cusum algorithm - a small review, 6 2013.

[36] Olivia A Grigg, VT Farewell, and DJ Spiegelhalter. Use of risk-adjusted cusum and rsprtcharts for monitoring in medical contexts. *Statistical methods in medical research*, 12(2):147–170, 2003.

[37] Farzin Haque, Jinghong Li, Hai-Chen Wu, Xing-Jie Liang, and Peixuan Guo. Solid-state and biological nanopore for real-time sensing of single chemical and sequencing of dna. *Nano today*, 8(1):56–74, 2013.

[38] Trevor Hastie, Jerome Friedman, and Robert Tibshirani. *The Elements of Statistical Learning*. Springer New York, 2001.

[39] Liqun He, Daniel R Tessier, Kyle Briggs, Matthaios Tsangaris, Martin Charron, Erin M McConnell, Dmytro Lomovtsev, and Vincent Tabard-Cossa. Digital immunoassay for biomarker concentration quantification using solid-state nanopores. *Nature Communications*, 12(1):5348, 2021.

[40] Marcus Hutter. Bayesian regression of piecewise constant functions. *arXiv preprint math/0606315*, 2006.

[41] Marcus Hutter. Exact bayesian regression of piecewise constant functions. *Bayesian Analysis*, 2(4):635 – 664, 2007.

[42] Simon King, Kyle Briggs, Robert Slinger, and Vincent Tabard-Cossa. Screening for group a streptococcal disease via solid-state nanopore detection of pcr amplicons. *ACS Sensors*, 7:207–214, 1 2022.

[43] Lena Koepcke, Go Ashida, and Jutta Kretzberg. Single and multiple change point detection in spike trains: Comparison of different cusum methods. *Frontiers in Systems Neuroscience*, 10, 6 2016.

[44] Nidhi Kohli and Jeffrey R Harring. Modeling growth in latent variables using a piecewise function. *Multivariate Behavioral Research*, 48(3):370–397, 2013.

[45] Harold Kwok, Kyle Briggs, and Vincent Tabard-Cossa. Nanopore fabrication by controlled dielectric breakdown. *PloS one*, 9(3):e92880, 2014.

[46] Loic Landrieu and Guillaume Obozinski. Cut pursuit: Fast algorithms to learn piecewise constant functions on general weighted graphs. *SIAM Journal on Imaging Sciences*, 10(4):1724–1766, 2017.

[47] Yaron Lipman and David Levin. Approximating piecewise-smooth functions. *IMA journal of numerical analysis*, 30(4):1159–1183, 2010.

[48] Sixiang Liu, Junlong Tian, and Wang Zhang. Fabrication and application of nanoporous anodic aluminum oxide: a review. *Nanotechnology*, 32(22):222001, 2021.

[49] Nikos K Logothetis, Jon Pauls, Mark Augath, Torsten Trinath, and Axel Oeltermann. Neurophysiological investigation of the basis of the fmri signal. *nature*, 412(6843):150–157, 2001.

[50] Randolph Lopez, Yuan-Jyue Chen, Siena Dumas Ang, Sergey Yekhanin, Konstantin Makarychev, Miklos Z Racz, Georg Seelig, Karin Strauss, and Luis Ceze. Dna assembly for nanopore data storage readout. *Nature communications*, 10(1):2933, 2019.

[51] R Misener and CA Floudas. Piecewise-linear approximations of multidimensional functions. *Journal of optimization theory and applications*, 145:120–147, 2010.

[52] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[53] Jeff Nivala, Douglas B Marks, and Mark Akeson. Unfoldase-mediated protein translocation through an α-hemolysin nanopore. *Nature biotechnology*, 31(3):247–250, 2013.

[54] Seiji Ogawa, Tso-Ming Lee, Asha S Nayak, and Paul Glynn. Oxygenation-sensitive contrast in magnetic resonance image of rodent brain at high magnetic fields. *Magnetic resonance in medicine*, 14(1):68–78, 1990.

[55] Toshihisa Osaki, Hiroaki Suzuki, Bruno Le Pioufle, and Shoji Takeuchi. Multichannel simultaneous measurements of single-molecule translocation in α-hemolysin nanopore array. *Analytical chemistry*, 81(24):9866–9870, 2009.

[56] E. S. Page. Continuous inspection schemes. *Biometrika*, 41:100, 6 1954.

[57] Daniel Pedone, Matthias Firnkes, and Ulrich Rant. Data analysis of translocation events in nanopore experiments. *Analytical chemistry*, 81(23):9689–9694, 2009.

[58] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[59] Philipp Petersen and Felix Voigtlaender. Optimal approximation of piecewise smooth functions using deep relu neural networks. *Neural Networks*, 108:296–330, 2018.

[60] Michael Plass and Maureen Stone. Curve-fitting with piecewise parametric cubics. In *Proceedings of the 10th annual conference on Computer Graphics and Interactive Techniques*, pages 229–239, 1983.

[61] Calin Plesa and Cees Dekker. Data analysis methods for solid-state nanopores. *Nanotechnology*, 26(8):084003, 2015.

[62] James M Poterba, Andrew A Samwick, Andrei Shleifer, and Robert J Shiller. Stock ownership patterns, stock market fluctuations, and consumption. *Brookings papers on economic activity*, 1995(2):295–372, 1995.

[63] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[64] C. Raillon, P. Granjon, M. Graf, L. J. Steinbock, and A. Radenovic. Fast and automatic processing of multi-level events in nanopore translocation experiments. *Nanoscale*, 4:4916, 2012.

[65] Camille Raillon, Pierre Granjon, Michael Graf, LJ Steinbock, and Aleksandra Radenovic. Fast and automatic processing of multi-level events in nanopore translocation experiments. *Nanoscale*, 4(16):4916–4924, 2012.

[66] Steffen Rebennack and Vitaliy Krasko. Piecewise linear function fitting via mixed-integer linear programming. *INFORMS Journal on Computing*, 32(2):507–530, 2020.

[67] Rita P Ribeiro. Utility-based regression. *Ph. D. dissertation*, 2011.

[68] Felipe Rivas, Osama K Zahid, Heidi L Reesink, Bridgette T Peal, Alan J Nixon, Paul L DeAngelis, Aleksander Skardal, Elaheh Rahbar, and Adam R Hall. Label-free analysis of physiological hyaluronan size distribution with a solid-state nanopore sensor. *Nature communications*, 9(1):1037, 2018.

[69] Zachary Roelen, Kyle Briggs, and Vincent Tabard-Cossa. Analysis of nanopore data: classification strategies for an unbiased curation of single-molecule events from dna nanostructures. *ACS sensors*, 8(7):2809–2823, 2023.

[70] Sarah E. Sandler, Robert I. Horne, Sara Rocchetti, Robert Novak, Nai-Shu Hsu, Marta Castellana Cruz, Z. Faidon Brotzakis, Rebecca C. Gregory, Sean Chia, Gonçalo J. L. Bernardes, Ulrich F. Keyser, and Michele Vendruscolo. Multiplexed digital characterization of misfolded protein oligomers via solid-state nanopores. *Journal of the American Chemical Society*, 145:25776–25788, 11 2023.

[71] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a" kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st international conference on distributed computing systems workshops*, pages 166–171. IEEE, 2011.

[72] Grégory F Schneider, Stefan W Kowalczyk, Victor E Calado, Grégory Pandraud, Henny W Zandbergen, Lieven MK Vandersypen, and Cees Dekker. Dna translocation through graphene nanopores. *Nano letters*, 10(8):3163–3167, 2010.

[73] Jacob Schreiber and Kevin Karplus. Analysis of nanopore data using hidden markov models. *Bioinformatics*, 31(12):1897–1903, 2015.

[74] Milton Severo and Joao Gama. Change detection with kalman filter and cusum. In *International Conference on Discovery Science*, pages 243–254. Springer, 2006.

[75] Wei Si and Aleksei Aksimentiev. Nanopore sensing of protein folding. *ACS nano*, 11(7):7091–7100, 2017.

[76] Ralph MM Smeets, Ulrich F Keyser, Nynke H Dekker, and Cees Dekker. Noise in solid-state nanopores. *Proceedings of the National Academy of Sciences*, 105(2):417–421, 2008.

[77] Norman R Swanson and Halbert White. Forecasting economic time series using flexible versus fixed specification and linear versus nonlinear econometric models. *International journal of Forecasting*, 13(4):439–461, 1997.

[78] Vincent Tabard-Cossa, Dhruti Trivedi, Matthew Wiggin, Nahid N Jetha, and Andre Marziali. Noise analysis and reduction in solid-state nanopores. *Nanotechnology*, 18:305505, 8 2007.

[79] Vincent Tabard-Cossa, Dhruti Trivedi, Matthew Wiggin, Nahid N Jetha, and Andre Marziali. Noise analysis and reduction in solid-state nanopores. *Nanotechnology*, 18(30):305505, 2007.

[80] Andreas Thiel, Martin Greschner, Christian W. Eurich, Josef Ammermüller, and Jutta Kretzberg. Contribution of individual retinal ganglion cell responses to velocity and acceleration encoding. *Journal of Neurophysiology*, 98:2285–2296, 10 2007.

[81] Veerle Van Meervelt, Misha Soskine, Shubham Singh, Gea K Schuurman-Wolters, Hein J Wijma, Bert Poolman, and Giovanni Maglia. Real-time conformational changes and controlled orientation of native proteins inside a protein nanoreactor. *Journal of the American Chemical Society*, 139(51):18640–18646, 2017.

[82] Martin Von Mohrenschildt. A normal form for function rings of piecewise functions. *Journal of Symbolic Computation*, 26(5):607–619, 1998.

[83] Deekshant Wadhwa, Kyle Briggs, Vincent Tabard-Cossa, and Paula Branco. Preprocessing nanopore signals. Patent Application, 2024. Patent application filed with application number XXXXXXXXX.

[84] Deekshant Wadhwa, Philipp Mensing, James Harden, Paula Branco, Vincent Tabard-Cossa, and Kyle Briggs. Nano trees: Nanopore signal processing and sublevel fitting using decision trees. Submitted, 2024.

[85] Pradeep Waduge, Rui Hu, Prasad Bandarkar, Hirohito Yamazaki, Benjamin Cressiot, Qing Zhao, Paul C Whitford, and Meni Wanunu. Nanopore-based measurements of protein size, fluctuations, and conformational changes. *ACS nano*, 11(6):5706–5716, 2017.

[86] Meni Wanunu. Nanopores: A journey towards dna sequencing. *Physics of life reviews*, 9(2):125–158, 2012.

[87] Matthew Waugh, Kyle Briggs, Dylan Gunn, Mathieu Gibeault, Simon King, Quinn Ingram, Aura Melissa Jimenez, Samuel Berryman, Dmytro Lomovtsev, Lukasz An-

drzejewski, et al. Solid-state nanopore fabrication by automated controlled breakdown. *Nature Protocols*, 15(1):122–143, 2020.

[88] Ruoshan Wei, Daniel Pedone, Andreas Zürner, Markus Döblinger, and Ulrich Rant. Fabrication of metallized nanopores in silicon nitride membranes for single-molecule sensing. *small*, 6(13):1406–1414, 2010.

[89] Eric W. Weisstein. Piecewise constant function. https://mathworld.wolfram.com/PiecewiseConstantFunction.html. Accessed: 2024-06-15.

[90] Chenyu Wen, Dario Dematties, and Shi-Li Zhang. A guide to signal processing algorithms for nanopore sensors. *ACS sensors*, 6(10):3536–3555, 2021.

[91] James Wilson, Leila Sloman, Zhiren He, and Aleksei Aksimentiev. Graphene nanopores for protein sequencing. *Biophysical Journal*, 110(3):326a, 2016.

[92] Longhua Wu, Jiawen Zhou, Tong Zhou, Zhu Li, Jinping Jiang, Dong Zhu, Jinyu Hou, Zhaoyang Wang, Yongming Luo, and Peter Christie. Estimating cadmium availability to the hyperaccumulator sedum plumbizincicola in a wide range of soil types using a piecewise function. *Science of the total environment*, 637:1342–1350, 2018.

[93] Tianhu Yu, Dengqing Cao, Shengqiang Liu, and Huatao Chen. Stability analysis of neural networks with periodic coefficients and piecewise constant arguments. *Journal of the Franklin Institute*, 353(2):409–425, 2016.

[94] Erik C Yusko, Brandon R Bruhn, Olivia M Eggenberger, Jared Houghtaling, Ryan C Rollings, Nathan C Walsh, Santoshi Nandivada, Mariya Pindrus, Adam R Hall, David Sept, et al. Real-time shape approximation and fingerprinting of single proteins using a nanopore. *Nature nanotechnology*, 12(4):360–367, 2017.

[95] Jian-Hua Zhang, Xiu-Ling Liu, Zheng-Li Hu, Yi-Lun Ying, and Yi-Tao Long. Intelligent identification of multi-level nanopore signatures for accurate detection of cancer biomarkers. *Chemical communications*, 53(73):10176–10179, 2017.

[96] Jianhua Zhang, Xiuling Liu, Yi-Lun Ying, Zhen Gu, Fu-Na Meng, and Yi-Tao Long. High-bandwidth nanopore data analysis by using a modified hidden markov model. *Nanoscale*, 9(10):3458–3465, 2017.

[97] Jinbo Zhu, Niklas Ermann, Kaikai Chen, and Ulrich F Keyser. Image encoding using multi-level dna barcodes with nanopore readout. *Small*, 17(28):2100711, 2021.

APPENDICES

Appendix A

Automation

A.1 Extended Overfitting Automation

The overfitting automation discussed in Section 4.4.1 is just one of the several variations we tested for this purpose. This section discusses several other changes to this automation that work similarly, and produce similar results but use other algorithms. Beginning with, the parameter choice in the Decision Tree regression. Instead of max_leaf_nodes, max_depth can also be used for determining the change point and consequently, the overfit on the data as shown in Figure A.1. max_leaf_nodes was used because it more more interpretable and almost comparable to event sublevels. But max_depth does a similar job of controlling the complexity of the decision tree but proportional to the sublevels approximately exponentially. Both these parameters can be used together as well, increasing the search space, and fine-tuning the results, but making the process way slower for not much gain in the results.

Besides the use of these 2 parameters, other parameters like $min\_samples\_split$,

min_samples_leaf, *min_impurity_decrease* etc. can also be used, not for iteration but to reduce the overfitting to the minimum while still keeping all sublevels intact while iterating. But this can only be done if the dataset on which Nano Trees is applied is well known and the user has a good estimate of what these parameters are, how to correlate them to the properties of the data, and to get a good enough estimate such as to not ruin the overall process. As doing this would make the results far better and make the use of several downstream passes redundant, this would also reduce the generalizability of the process. It would make the fit entirely dependent on the best guess of the user and defeat the purpose of developing a robust, adaptable and generalizable algorithm like Nano Trees. It is essential to strike a balance between leveraging user expertise and maintaining a systematic methodology that can generalize well across various scenarios and datasets.

In Section 4.4.1, we used MSE on the Y-axis to determine the changepoint. This can be replaced with other scoring/error functions like R^2 Score, Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), average tree impurity, etc. The resultant graph in those cases might not be the same as Figure 4.12b, but they would be similar in the sense that would have a knee or an elbow point close enough in the acceptable range. The plots can be convex or concave, and increasing or decreasing, but the procedure of extracting the change point remains the same, which is discussed in Appendix A.2.

Finally, the model used to produce the overfitted output can be switched from the Decision Trees to other similar machine learning models as well, like AdaBoost with Decision Trees, Extreme Gradient Boosting (XGBoost) [17] etc. Decision Trees were the simplest solution to this problem, hence more complicated methods were not tested, but they can just as easily be used as an alternative. The reason, all these alternatives are possible is because the process of overfitting is quite forgiving. As long as the output of this automation is in the correct ballpark, bigger than it should be, the further passes adjust the

overall fit anyhow. The change point gets us close to the ballpark and the α parameter makes sure that the final estimation does not fall short. The set of choices we made in Section 4.4.1 seem to be the most stable throughout the limited visual testing done on its outputs. However, it is likely that a different set might be better for some other types of signals, or a different alpha is required for more complicated signals. This is the reason we consider this an automation heuristic, used as a good starting point for hyperparameter tuning, and it should not be applied automatically in practical situations without manual validations.

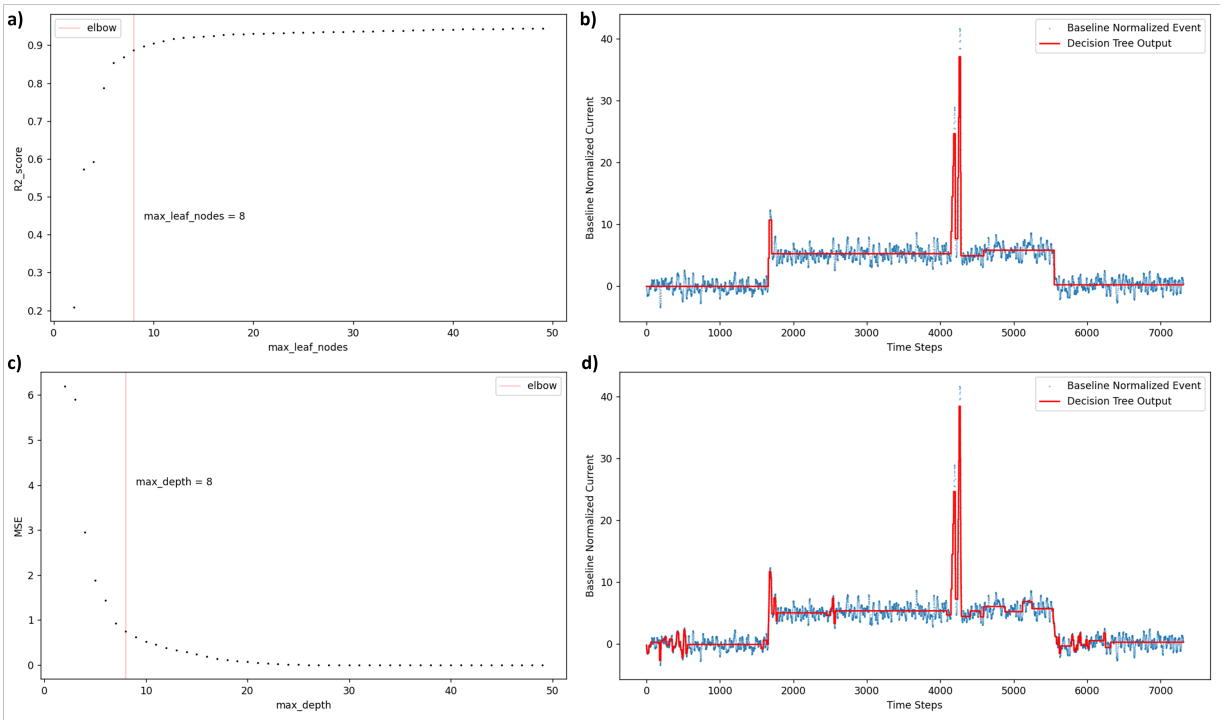

Figure A.1: Other variations for Overfitting Automation. a) Using max_leaf_nodes with R^2 Score. b) Overfitted fit for Figure A.1a and $\alpha = 2$. c) Using max_depth with MSE. d) Overfitted fit for Figure A.1c and $\alpha = 1$

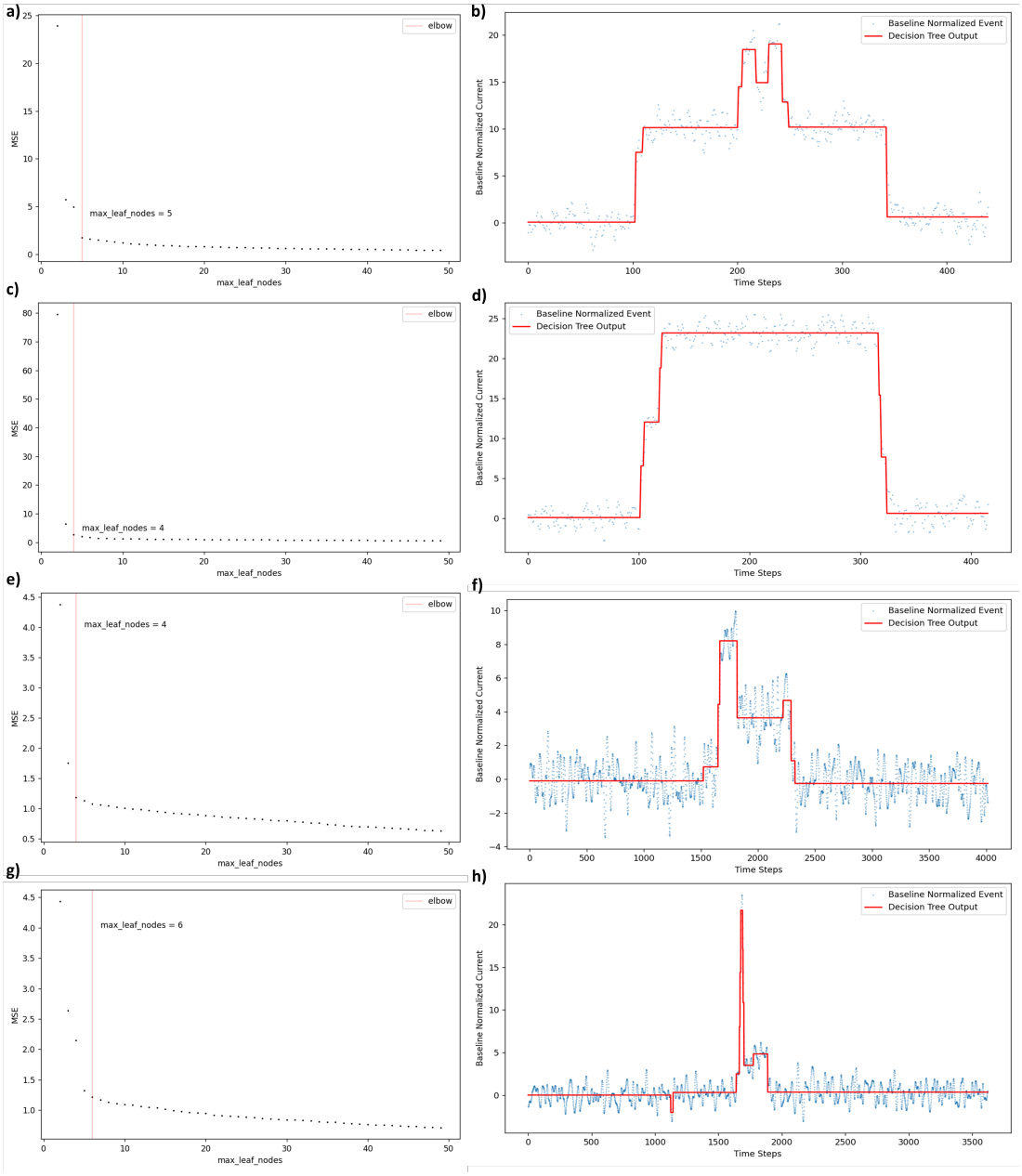

Figure A.2: Overfitting Automation applied on other datasets from Section 5.

A.2 Change Point Detection

The elbow point or the knee point has been referred to as the change point in this thesis. Determining that change point is separate research on its own, Using a manual threshold in a higher-order derivative of the change point graph as shown in 4.12a could provide us with the result we are looking for, and given the high tolerance of the Overfitting Automation procedure (Discussed in Section 4.4.1) it would be in the right ballpark as well. But for this research, and for the sake of ease, we have opted for an alternative approach. We rely on the research by Satopaa, Ville, et al. [71], where the authors develop the "kneed" algorithm [2] for finding the knee point of a function.

The "kneed" python package [2] can be used on both convex and concave curves, irrespective of whether they are increasing or decreasing. The algorithm also has a *online* parameter that tries to correct the change point iteratively, in our change point plot several times there is a small deviation in the beginning, setting this parameter to $True$ helps in finding the correct value. For Figure 4.12a, we used the following parameters to find the elbow point using the "kneed" package.

```
from kneed import KneeLocator
elbowPoint = KneeLocator(
    x = maxLeafNodes,
    y = meanSquaredErrorValues,
    curve='convex',
    direction='decreasing',
    online=True
)
newMaxLeafNodes = 2*elbowPoint
```

Appendix B

Hyperparameters

B.1 Hyperparameter description

Nano Trees use several hyperparameters to generate fits. The following section provides a summary of each hyperparameter used in all passes and a description of what it does.

Pass 1: AdaBoost

- approxSubLevelEstimate – It is the maximum depth of Decision trees. This parameter controls how deep the Decision tree is allowed to grow. The deeper the tree is, the more accurate its predictions will be and the more overfitted the result will be. A shallower tree will have fewer levels and be a rougher approximation to the data. At this stage we want the predictions to only be accurate enough to extract a general structure in the signal without losing so much information that the overall shape is lost. If the sublevels seem to be missing due to this pass, increase this value.

- adaBoostRegressorNEstimators – This is the number of Decision trees used during boosting. It is similar to approxSubLevelEstimate and controls how accurate the predictions will be in this pass. If the sublevels seem to be missing due to this pass, increase this value.

Pass 2: Decision Trees

- approxSubLevelEstimate – Same as in Pass 1 but keep this value as high as possible to reduce noise in the system without losing the overall shape. Even if a few heights are incorrect after this pass, further passes will try to recover them but the shape should be maintained as is while tuning this parameter.

Pass 3: Merge Small Current Steps

- numberOfStdAboveAndBelow – This parameter (multiplied by baseline standard deviation internally) controls the height difference between any two consecutive sublevels. If the height of a sublevel is within the range of previous height $\pm$ this threshold then these two sublevels are merged into one.

- minDataPointsToBeBoosted - Before merging takes place, some sublevels are to be boosted (Height correction using height function) so that they are not wrongly merged into/with another sublevel. If the number of data points in a sublevel is greater than this parameter, only then it is considered for boosting. This is done such that extremely narrow sublevels do not get boosted and eventually in some cases, wrongly establish a false positive sublevel.

- oneSidedPercentParity – Not all sublevels considered for boosting are actually boosted. The percentage of data points above and below a sublevel is calculated, and if their absolute difference is greater than this threshold, only then is that sublevel boosted.

- exceptionalHeightBaseMaxDiffForHeightRefresh – After boosting and merging, in some extreme cases, certain sublevels may still exhibit inaccurate height values. To address this for these sublevels, we calculate the absolute difference between the highest data point and sublevel height, and between the lowest data point and the sublevel height. If the maximum of these two values exceeds this parameter, then the height of these sublevels is updated using the height function.

Pass 4: Categorize and Correct Sublevels with Short Durations

- minDataPointsToBeSubLevel – Any sublevel that is shorter than this (number of data points) parameter (and is not exceptional) is deleted, and its data points are split into the sublevels to its left and right.

- numberOfStdAboveAndBelow – This parameter (multiplied by baseline standard deviation) is used to distribute data points of a deleted sublevel. Any data point within this threshold range of the previous sublevel height becomes a part of the previous sublevel, and all data points after (and including) the first data point outside this range become part of the sublevel to the right of the deleted sublevel.

- exceptionalPeak_MinHeightStdAboveAndBelow – This parameter is used while determining whether a sublevel is an exceptional peaky sublevel. The absolute difference between the height of a sublevel and both the previous and next sublevel height should be greater than this parameter (multiplied by baseline standard deviation).

- exceptionalPeak_WidthLowerBound – This parameter is used while determining whether a sublevel is an exceptional peaky sublevel. The exceptional peaky sublevel should have more data points in it than this parameter.

- exceptionalPeak_BaseDifferenceStdAtleast – This parameter is used while determining whether a sublevel is an exceptional peaky sublevel. The absolute difference between the height of the previous sublevel and the height of the first non-exceptional sublevel after the current sublevel should be greater than this parameter (multiplied by baseline standard deviation).

- exceptionalSlope_WidthLowerBound – This parameter is used while determining whether a sublevel is an exceptional slope sublevel. The exceptional slope sublevel should have more data points in it than this parameter.

- exceptionalSlope_MinHeightStdOfMinDiff – This parameter is used while determining whether a sublevel is an exceptional slope sublevel. The minimum of absolute height difference between the current and previous sublevel, and between the current and next sublevel should be greater than this parameter (multiplied by baseline standard deviation).

Pass 5: Merge Small Current Steps (Repeat)

(Same as Pass 3)

Pass 6: Clear Baseline

- baselineStdThreshold – All sublevels in the range of baseline height ± this parameter are deleted and considered as baseline before finding the largest event in this pass.

Pass 7: Backtrack

(Backtracking does not require any parameters because it is direction-based. A tolerance parameter can be added if extremely precise sublevel positions are required or if the noise

present in the event is significantly high which was not removed in previous passes. The tolerance parameter will control how many opposite-direction data points to ignore before reaching the absolute point of inflection used as the sublevel endpoint.)

Post-Processing

- directionalThreshold – If the ratio of data points above and below the height of a sublevel is greater than this parameter then other height functions are used to minimize that ratio.

Sublevel Current Estimation Function

- shortSublevelDefinition – Sublevels that have more data points than this parameter have their heights calculated by the mean of the last 50% of the data points in them. The remaining sublevels have too few data points, inasmuch that their height is determined to be equal to the most extreme point in that sublevel.

B.2 Defaults and Changes

Knowing the details of all hyperparameters is not enough to to use it to get a fit. The following section provides a default value for all those hyperparameters and the consequence of changing them. These descriptions are for independently changing (increasing or decreasing) each hyperparameter. Users are advised to follow these for minor changes but refer to Appendix B.4 to get details on specific debugging scenarios to adjust the hyperparameters starting from these default values. Starting from the default values provided, these parameters can be varied depending on the desired outcome. For example,

if a user requires that every sublevel be detected no matter how small, the user could increase the hyperparameters of the first 2 passes to significantly higher values, set the "minDataPointsToBeSublevel" to a smaller value (just outside the noise region), and the corresponding "numberOfStdAboveAndBelow" to a lower value as well. The exceptional parameters can be ignored because we plan to keep most of the suspected sublevels in the fit as is. The rest of the parameters in all the passes can be kept as default. This will overfit the data, allowing the fit to retain most of the sublevels and experiment-based filtering can be later applied based on users' experimental requirements.

Pass 1: AdaBoost

approxSubLevelEstimate

- Default Value: 5
- Increasing it will increase the maximum depth of Decision trees allowed throughout the AdaBoost ensemble. This will allow for more accurate fits but will also open the room for overfitting, and ultimately forcing this pass to have no overall effect on the fit.
- Decreasing it will increase the maximum depth of Decision trees in the AdaBoost ensemble. Doing this will allow for more relaxed fits thereby reducing the noise but still maintaining the correct shape of the event. Decreasing it too low could cause the fit to lose its overall shape and to an extent from where it might not even be recovered by further passes.

adaBoostRegressorNEstimators

- Default Value: 500

- Increasing it will increase the number of Decision trees used during boosting, providing a more accurate fit but also increasing the time taken to generate results. It has a similar effect on the fits as the "approxSubLevelEstimate" hyperparameter. Users are advised to maintain a good balance between the two hyperparameters for a good and efficient fit.
- Decreasing it will decrease the number of Decision trees used during boosting. This will allow the algorithm to terminate quicker at the cost of fit quality. It has a similar effect on the fits as "approxSubLevelEstimate" and should hence be tuned accordingly.

Pass 2: Decision Trees

approxSubLevelEstimate

- Default Value: 5
- Increasing and/or decreasing it has the same effect as the hyperparameter with the same name in pass 1.

Pass 3: Merge Small Current Steps

numberOfStdAboveAndBelow

- Default Value: 2.2
- Increasing it will increase the threshold within which 2 sublevels are merged. The threshold is also proportional to the baseline standard deviation, so if the baseline standard deviation is too high, this parameter should not be increased by much.

- Decreasing it decreases the merging threshold.

minDataPointsToBeBoosted

- Default Value: 20
- Increasing it will increase the number of data points a sublevel needs to have for it to be boosted before further processing can take place in this pass.
- Decreasing it lowers the number of data points a sublevel needs to have for it to be boosted.

oneSidedPercentParity

- Default Value: 0.2
- Increasing it increases the parity threshold required for sublevel boosting. As parity determines the ratio of data points higher and lower than the current sublevel estimate, a higher threshold would mean a lesser number of sublevels will be boosted overall.
- Decreasing it decreases the parity threshold required for sublevel boosting. This means more sublevels would have a parity higher than the threshold, hence more boosting would be done.

exceptionalHeightBaseMaxDiffForHeightRefresh

- Default Value: 0.35

- Increasing it raises the threshold that a sublevel needs to cross for its current estimate to be refreshed.
- Decreasing it lowers the threshold that a sublevel needs to cross for its current estimate to be refreshed.

Pass 4: Categorize and Correct Sublevels with Short Durations

minDataPointsToBeSubLevel

- Default Value: 2% event length
- Increasing it makes the sublevels shorter than this hyperparameter get added to the exceptional sublevels queue where they are judged using the criterion of an exceptional slope or an exceptional peak, and deleted if it satisfies none of them.
- Decreasing it allows shorter sublevels to exist as is without any additional exceptional sublevel checks.

numberOfStdAboveAndBelow

- Default Value: 2
- Increasing it extends the threshold below which the proportion of a removed sublevel is shifted to the left sublevel. Simultaneously, it also diminishes the proportion of the deleted sublevel transferred to the right sublevel. This adjustment depends on how much the deleted sublevel falls below the baseline standard deviation multiplied by the hyperparameter threshold of the sublevel to the left of the removed sublevel.

- Decreasing it lowers the threshold for shifting the deleted sublevel proportion to the left, while concurrently increasing its transfer to the right.

exceptionalPeak_MinHeightStdAboveAndBelow

- Default Value: 3
- Increasing it increases the absolute sublevel current estimate delta between the sublevel and the previous sublevel, and the sublevel and the next sublevel required to get removed from being an exceptional peaked sublevel.
- Decreasing it decreases that absolute delta and hence allows more, exceptional peaked sublevels to be present in the final fit that do not vary much in terms of sublevel current estimate from its previous and next sublevel.

exceptionalPeak_WidthLowerBound

- Default Value: 0.2% event length
- Increasing it raises the threshold for the minimum number of data points an exceptional peaked sublevel requires to not get deleted. A sublevel falls to the category of being exceptional if it has less than the "minDataPointsToBeSubLevel" number of datapoints, hence, if this hyperparameter "exceptionalPeak_WidthLowerBound " is set equal to greater than "minDataPointsToBeSubLevel" then all exceptional peaked sublevels are discarded.
- Decreasing it lowers the requirement of the minimum number of data points an exceptional peaked sublevel requires not to get deleted. It cannot be negative, and if

it is set to 0, then this check of the minimum number of data points for exceptional peaked sublevels is discarded and other tests are used to check whether to keep or discard this sublevel.

exceptionalPeak_BaseDifferenceStdAtleast

- Default Value: 0
- Increasing it increases the sublevel current estimate delta between the next and the previous non-exceptional sublevels.
- Decreasing it decreases the sublevel current estimate delta between the next and the previous non-exceptional sublevels. Setting it to 0 disables this check to determine whether to keep an exceptional peaked sublevel but other non-disabled tests are used.

exceptionalSlope_WidthLowerBound

- Default Value: 10% event length
- Increasing it raises the threshold for the minimum number of data points an exceptional slope sublevel requires to not get deleted. A sublevel falls into the category of being exceptional if it has less than the "minDataPointsToBeSubLevel" number of datapoints, hence, if this hyperparameter "exceptionalSlope_WidthLowerBound " is set equal to greater than "minDataPointsToBeSubLevel" then all exceptional peaked sublevels are discarded.
- Decreasing it lowers the requirement of the minimum number of data points an exceptional slope sublevel requires not to get deleted. It cannot be negative, and if

it is set to 0, then this check of the minimum number of data points for exceptional slope sublevels is discarded and other tests are used to check whether to keep or discard this sublevel.

exceptionalSlope_MinHeightStdOfMinDiff

- Default Value: 2.2
- Increasing it increases the threshold that the minimum of sublevel current estimate delta between the current and the previous and current and the next sublevel must cross to pass this check for being an exceptional slope sublevel.
- Decreasing it decreases this threshold.

Pass 5: Merge Small Current Steps (Repeat)

(Same as Pass 3)

Pass 6: Clear Baseline

baselineStdThreshold

- Default Value: 1.5
- Increasing it increases the threshold under which any sublevel before the approximate start value and after the approximate end value is merged to the baseline.
- Decreasing it decreases this threshold for deleting non-primary fits in the event.

Pass 7: Backtrack

(No hyperparameters used.)

Post-Processing

directionalThreshold

- Default Value: 0.5
- Increasing it increases the threshold a sublevel has to cross for the ratio of maximum between the count of increasing and decreasing values and the number of points in the sublevel for its sublevel current estimate to be set as the mean of the last 50% of data points in the sublevel.
- Decreasing it decreases this threshold for the sublevel current estimate of this threshold to be set as the mean of the last 50% of data points in the sublevel.

Sublevel Current Estimation Function

shortSublevelDefinition

- Default Value: 2% event length
- Increasing it raises the threshold under which a sublevel is termed as short and its sublevel current estimate is set as the extreme value in the sublevel, and over this, it is calculated using the mean of the last 50% of the sublevel. Increasing it to a very high value forces all the sublevel current estimates in the fit to be set to the extreme values in the corresponding sublevels.
- Decreasing it lowers this threshold, and when set to a very low value forces all the sublevel current estimates to be calculated using the mean of the last 50% of the data in each sublevel.

B.3 Hyperparameter importance

There are several hyperparameters in Nano Trees used throughout several passes, but not all of them need to be tuned every time. This section covers the importance of tuning each hyperparameter based on various situations (Table B.1 and B.2). The hyperparameters can be divided into two categories based on their importance in the fit as follows:

- Required Hyperparameters – these parameters cannot be left at their default values and need to be specifically tuned to each experimental context.
- Optional Hyperparameters – these hyperparameters are used for minute adjustments to the fit and in most cases the default values are suitable or can be disabled for fast results.

The tuning of optional hyperparameters can be skipped for quick results and/or debugging, but essential hyperparameters control the essence of the fitting procedure and can end up making or ruining a perfect fit. The hyperparameters are also segregated based on specificity:

- Setup specific hyperparameters – these hyperparameters can remain consistent throughout most experiments done using the same setup. This does not mean that these should not be changed, but if multiple datasets are recorded using the same setup and the fitting works well on some and not on others, then it is highly unlikely that these hyperparameters need adjustment. They could be adjusted, but the user should first focus on adjusting other hyperparameters and settings.
- Experiment-specific hyperparameters – these hyperparameters require a good adjustment for different experiments based on what biomolecules are used in the experi-

ment, expected sublevels, baseline/noise characteristics, etc. Any major issue while setting these hyperparameters can majorly degrade the fit quality.

Table B.1: Description of the combination of required, optional, setup-specific, and experiment-specific hyperparameters.

	Setup specific hyperparameters	**Experiment specific hyperparameters**
Required Hyperparameters	Less tuning is required but tune them well at least once for every new setup.	Change these for every experiment to get the best fit. These need to be debugged first if the fits are bad.
Optional Hyperparameters	Set them once or use the default ones for most cases.	Much tuning is not required but they can greatly affect the fit quality if tuned well.

Table B.2: Hyperparameters classified into 4 categories based on Table B.1 descriptions.

	Setup specific hyperparameters	**Experiment specific hyperparameters**
Required Hyperparameters	numberOfStdAboveAndBelow, baselineStdThreshold	approxSubLevelEstimate, minDataPointsToBeSubLevel, shortSublevelDefinition
Optional Hyperparameters	oneSidedPercentParity, directionalThreshold, exceptionalHeightBaseMaxDiffForHeightRefresh	adaBoostRegressorNEstimators, minDataPointsToBeBoosted, Exceptional Slope Parameters, Exceptional Peak Parameters, Approximate Event Location

B.4 Debugging

Nano Trees is a brand new algorithm. Although we have done quite some testing on it, and documented it fairly well, there will always be a situation where the hyperparameter description and change descriptions are not sufficient to fix an issue in the fits produced by Nano Trees. This section covers several such cases; flow charts describing what to do if the user gets stuck in a certain situation. The flow charts have been designed thoroughly to address those situations, but they obviously do not cover all possible situations that could go wrong. Thus, for those scenarios "Manual Debugging" is required. This means the user has to go through the algorithm description, hyperparameter descriptions and changes (mentioned in Section 4, Appendix B.1 and Appendix B.2) to debug the issue.

Figure B.1 describes the steps of the most common issue, which is that peaks or slopes are missing in the fit. This flow chart is extended to describe specific steps to debug *approxSubLevelEstimate* hyperparameter in Figure B.2. The confidence parameter debugging is extended in Figure B.3. Similarly, the exceptional parameters for peaks are described in Figure B.4 and for slopes in Figure B.5. Another major issue is missing Normal sublevels, Figure B.6 describes how to debug this issue. Finally, overfitted fits where extra sublevels are fitted in the final fit can be debugged using the flow chart in Figure B.7. A slot for "Manual Debugging" is left in several flow charts where the flow charts cannot help further.

Figure B.1: Flow chart describing debugging steps to improve the hyperparameters in case slopes or peaks are missing in the fit.

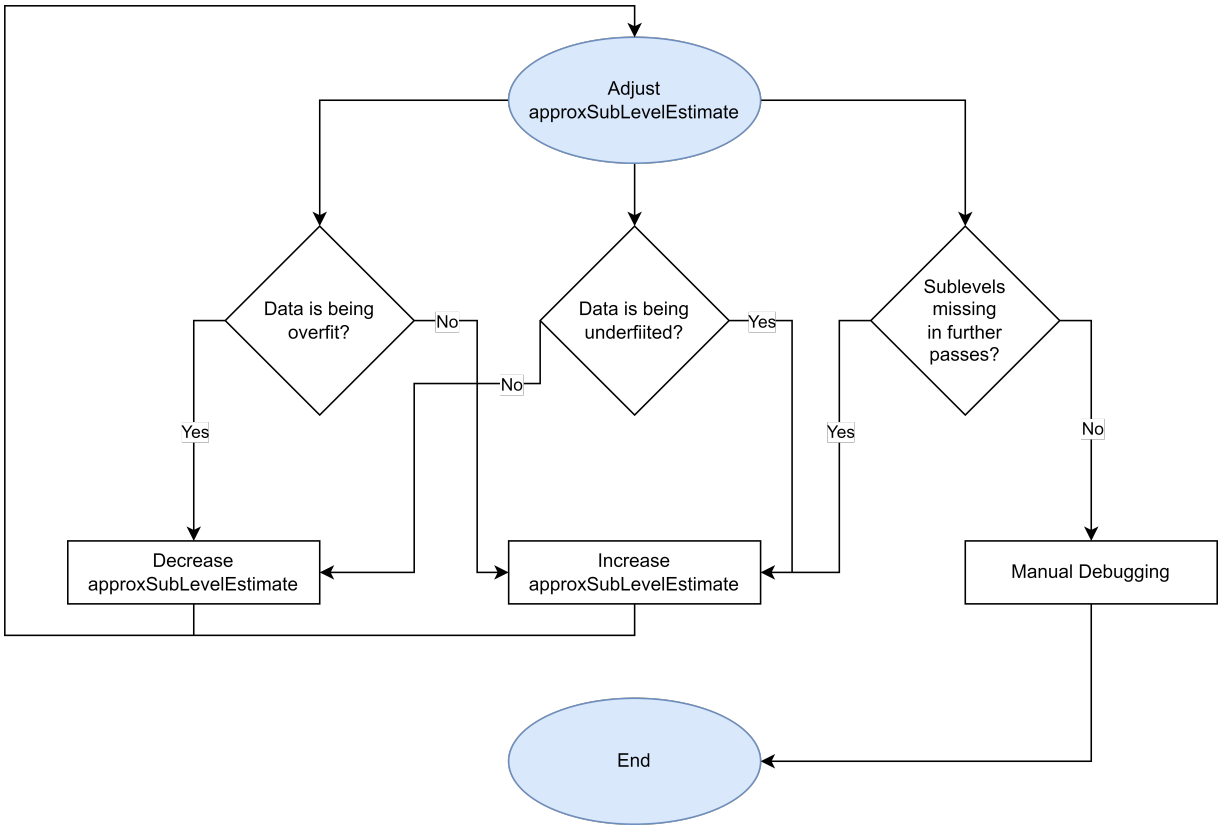

Figure B.2: Flow chart describing debugging steps to adjust approxSubLevelEstimate (blue).

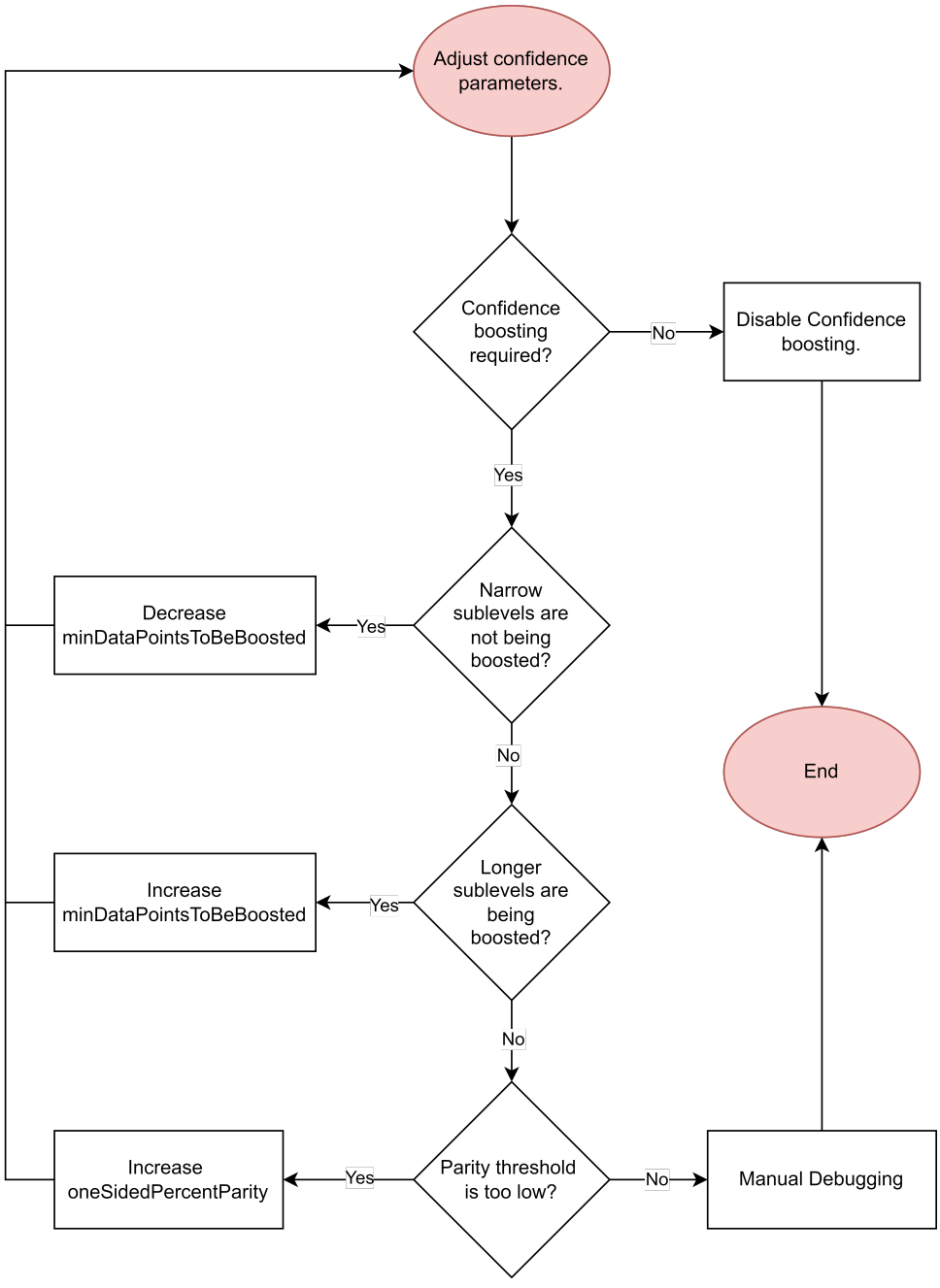

Figure B.3: Flow chart describing debugging steps to adjust confidence parameters (Red).

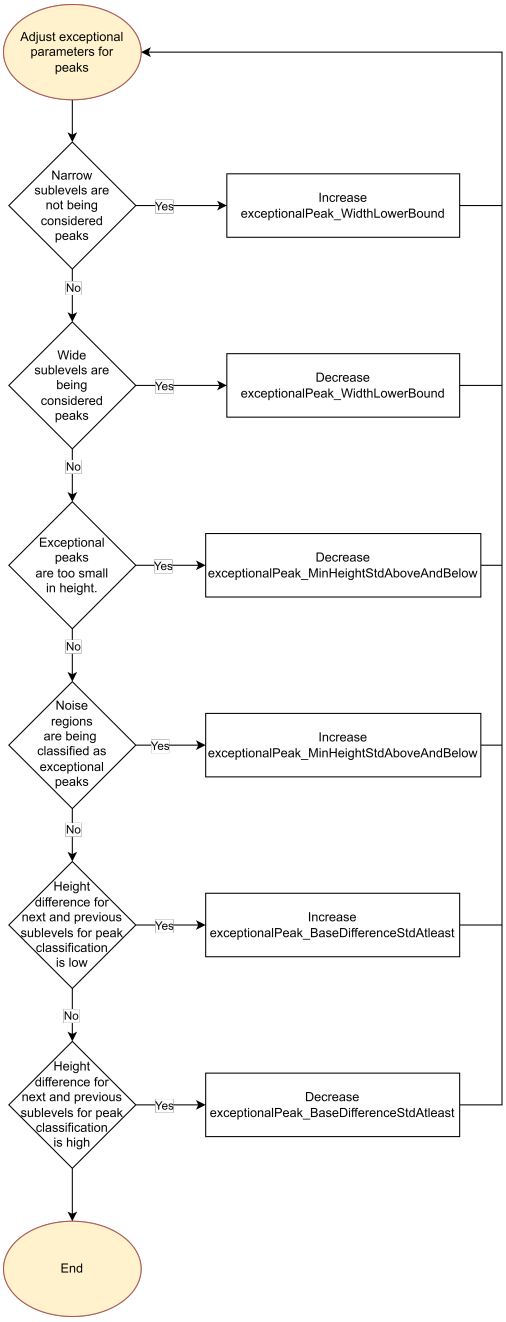

Figure B.4: Flow chart describing debugging steps to adjust exceptional parameters for peaks (Yellow).

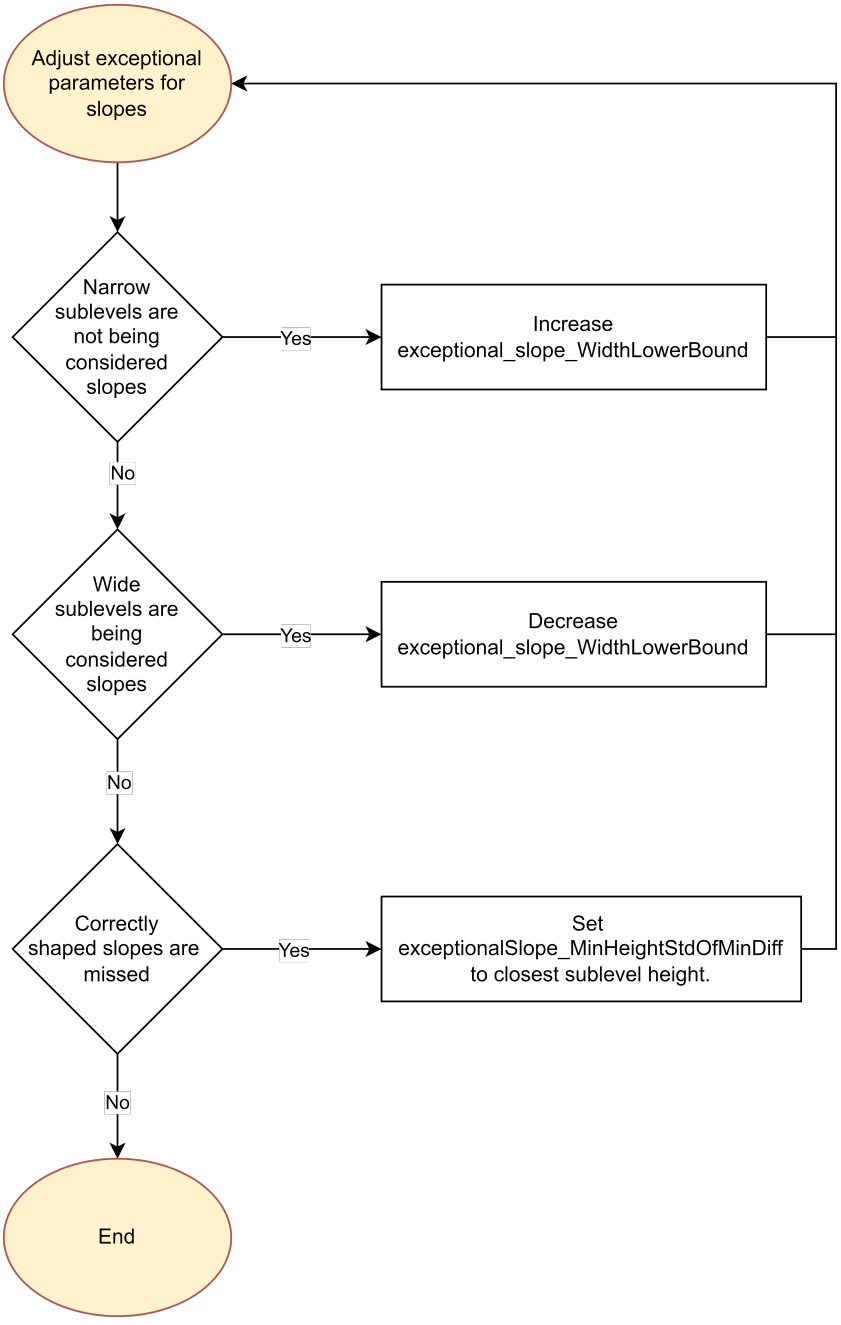

Figure B.5: Flow chart describing debugging steps to adjust exceptional parameters for slopes (Yellow).

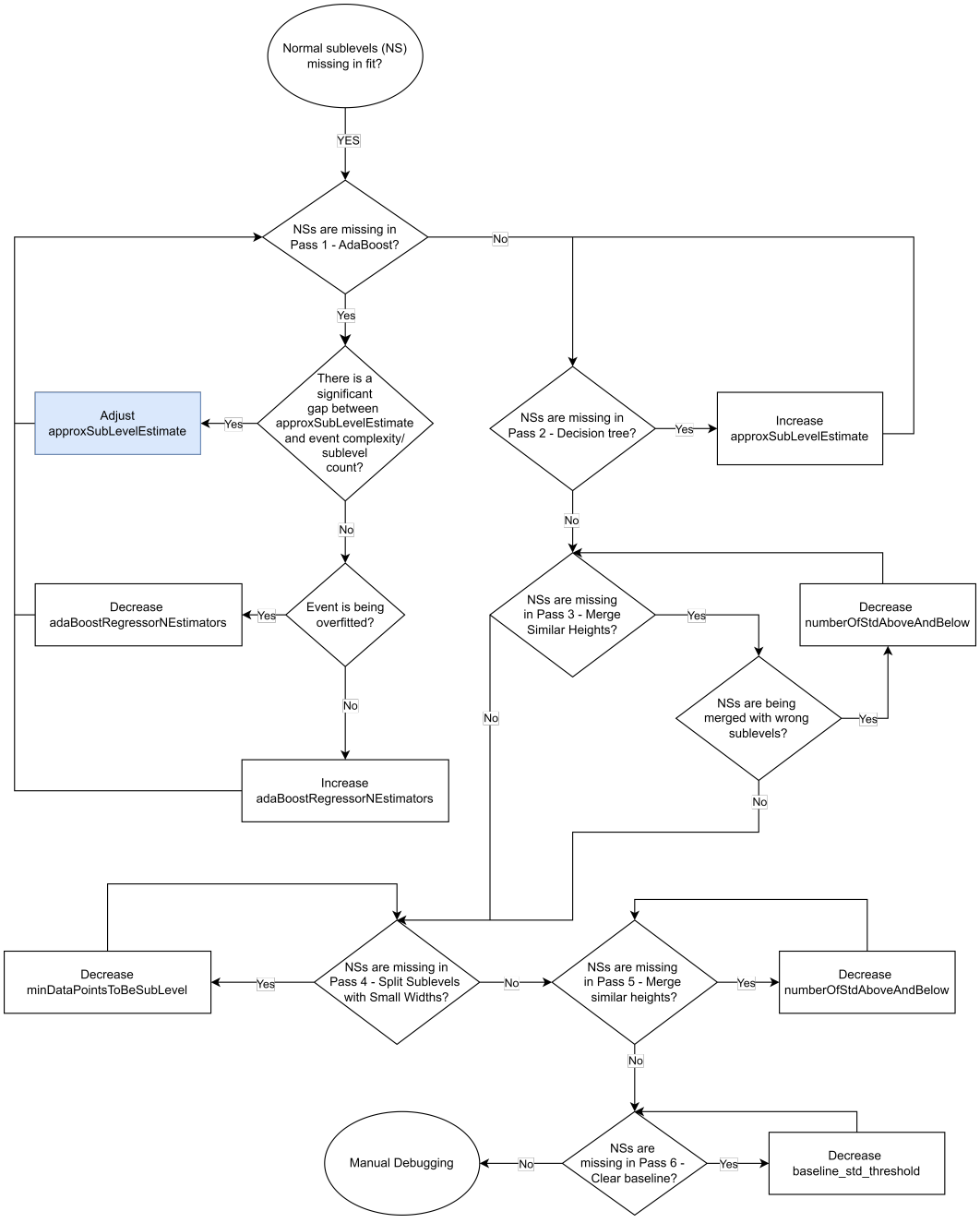

Figure B.6: Flow chart describing debugging steps to improve the hyperparameters if normal sublevels are missing in the fit (Blue).

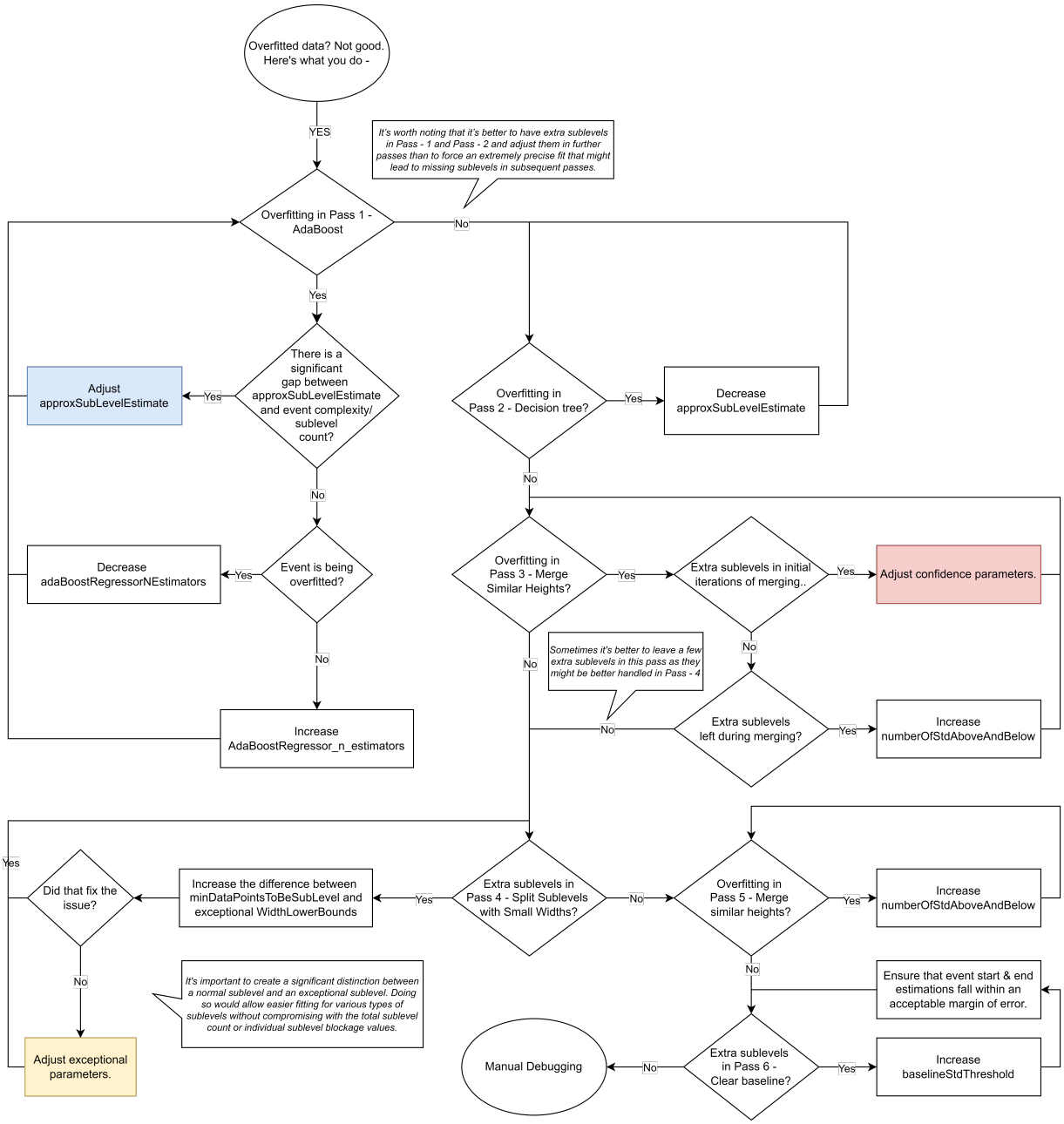

Figure B.7: Flow chart describing debugging steps to improve the hyperparameters if the fit is overfitted, which means there are extra physically possible sublevels in the fit.

B.5 Pass Name : Function Name

Each pass in Nano Trees corresponds to a function used in the Fitting Blocks Project 4.3 to build Nano Trees. These function names are different from the pass names used to describe Nano Trees functioning. This section correlates these two and also highlights how they have been used in the code. Figure B.8 represents the use of these names with the corresponding hyperparameters:

- Pass 1 Adaptive Boosting - ensemble_adaboost_decession_tree
- Pass 2 Decision Trees - single_decession_tree
- Pass 3 Merge Small Current Steps - mergeSimilarHeights_withIterativeHeightUpdates
- Pass 4 Categorize and Correct Sublevels with Short Durations - splitSublevelWithSmallWidths_withExceptionalSmallButTallSublevels
- Pass 5 Merge Small Current Steps (Repeat) - mergeSimilarHeights
- Pass 6 Clear Baseline - clear_baseline_using_longest_event_only
- Pass 7 Backtrack - backtrack_last_point_directional
- Post-Processing - slope_height_adjust
- Sublevel Current Estimation Function - l50_max_height or height function

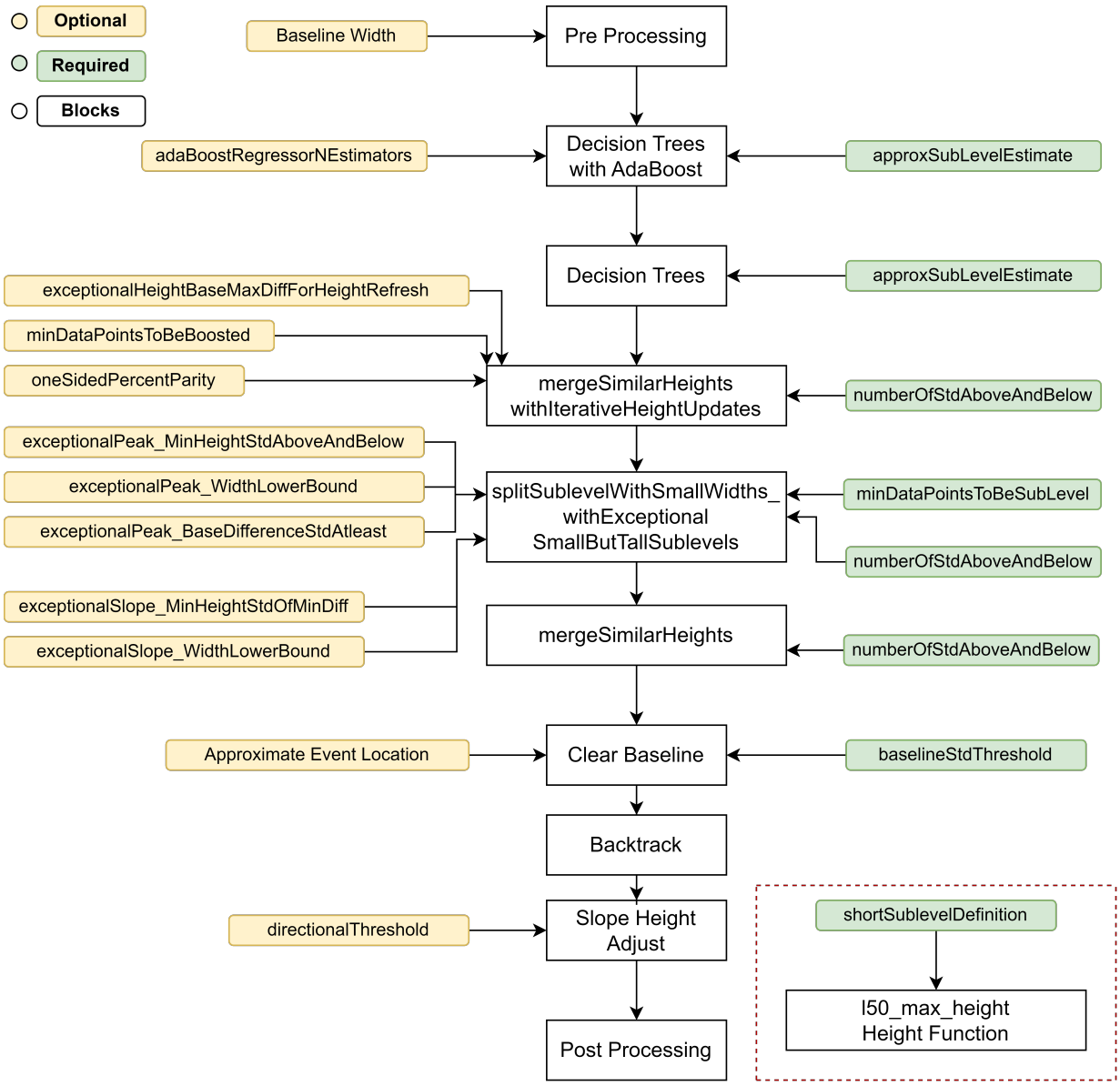

Figure B.8: Nano Trees pipeline with all functions and corresponding hyperparameters