Supplementary Information:

Nano Trees: Nanopore signal processing and sublevel fitting using Decision Trees

Deekshant Wadhwa,[1] Philipp Mensing,[2] James Harden,[2] Paula Branco,[1] Vincent Tabard-Cossa[2], and Kyle Briggs[2*]

[1]Department of Computer Science, University of Ottawa, Canada. [2]Department of Physics, University of Ottawa, Canada

*Corresponding author: kbriggs@uOttawa.ca

Supplementary Section S1: Nano Trees Pipeline

Nano Trees is a modular algorithm that ca be implemented in programming language, but the version in this research has been implemented using Python 3.10.11 and scikit-learn 1.2.2.

Algorithms Used

Adaptive Boosting

The Adaptive Boosting Algorithm (a.k.a. AdaBoost)[1,2] is an ensemble learning technique. It is a meta-algorithm that learns by training several base learners and improves itself by learning from their mistakes. In the domain of machine learning, base learners or weak learners are models that perform slightly better than random chance. These 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, it tries to reduce the weighted sum of mean square errors of each component decision tree by updating the weight of these weak learners in the overall model according to their individual performance.

Decision Trees

Decision Trees[3,4] is a supervised machine learning algorithm that can be used both for classification and regression and is used in several places in the Nano Trees pipeline. Decision Trees work by grouping data into sections that are represented by a single value (a constant sublevel ionic current blockage, in this context), which is then iteratively split into smaller sublevels until the mean-squared error arising from this local approximation is minimized. Initially, we have a single root node (or subevent), 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 is met. 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. By itself, this algorithm produces a high number of unphysical, false positive sublevels, which is desirable in an initial pass which has the goal of avoiding any false negatives.

The input to this pass is a vector representing normalized current sampled at a constant time interval. The output is a piecewise constant vector of current values that simplifies the signal by locally grouping data points into regions of constant current based on proximity.

Sublevel Current Estimation Function

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 the ways in which our measurement distorts the signal. Because the duration of a sublevel affects the extent of this distortion, we use a hyperparameter "*shortSublevelDefinition*" to split sublevels into two categories based on whether they last long enough to reach a steady state.

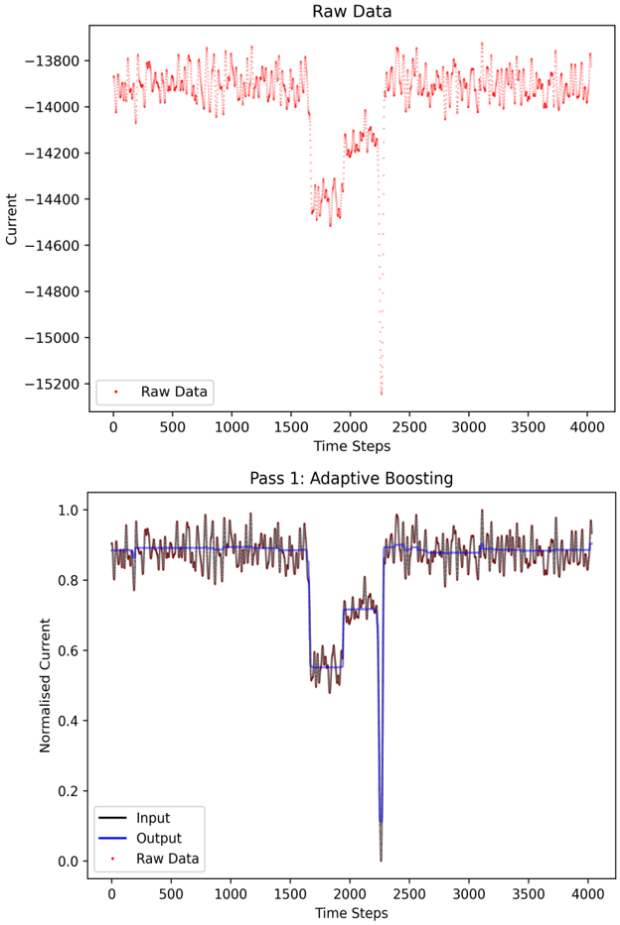
For sublevels that have fewer data points than *shortSublevelDefinition*, the extreme values are used as the current estimate. This means that the current value that is furthest from the previous sublevel in absolute value is used as the estimate of the sublevel current, effectively using local noise 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. The specifics of this algorithm are discussed in Supplementary Section S2. In a few places in the Nano Trees pipeline a different method is used to calculate the sublevel currents, which are described when applicable.

Equipped with a high-level description of the algorithms used, we next present the Nano Trees analysis pipeline of nanopore data.

Data Preprocessing

Before fitting, the current is normalized to [0,1]. This ensures that the hyperparameters used in each pass have a similar effect on each event. It is important to note that the normalization is done for each event individually.

Pass 1: Adaptive Boosting

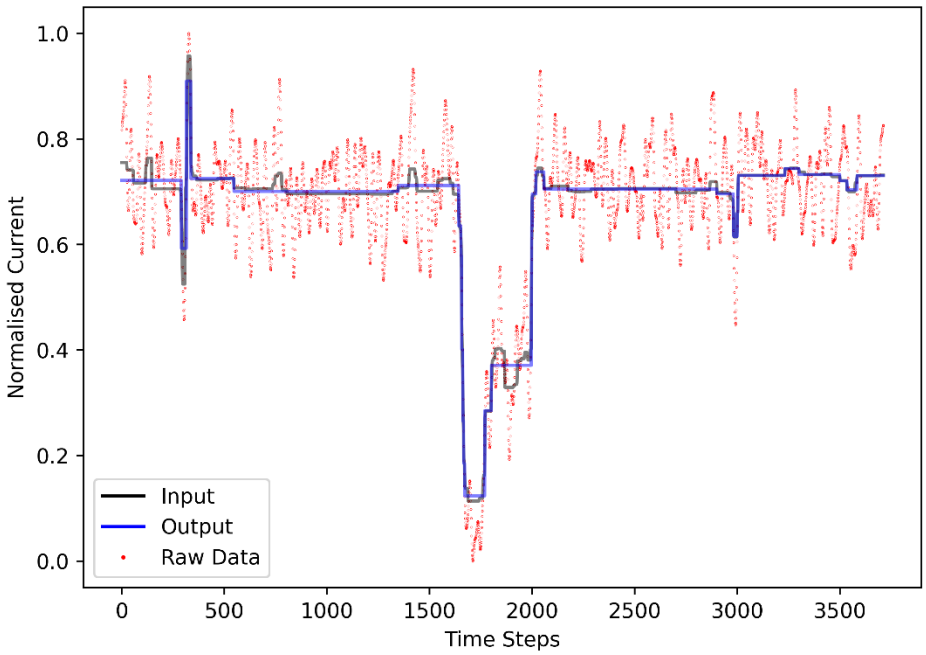


Supplementary Figure S1: a) Example current trace generated from passing 12-arm star nanostructures attached to a 2kbp dsDNA carrier through a ~13 nm pore (3.6 M LiCl, 75 mV) with a measurement bandwidth of 1 MHz (sampled at 4.17 MHz) and digitally low-pass filtered for fitting at 250 kHz taken from reference [5]. b) The input (black line) and output (blue line) from the first pass, consisting of the raw data (red dots) normalized to (0,1) and an overfitted piecewise constant approximation using the AdaBoost algorithm, respectively.

We first employ the AdaBoost algorithm using Decision Trees as our weak learners from the scikit-learn package [6]. Note that these algorithms are not trained in the sense that the phrase is usually used in deep learning, and do not require extensive training data as a result. Instead, during prediction, AdaBoost 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, which otherwise would have been made by individual iterations of Decision Trees.

The process is shown in Supplementary Figure S1, starting with the raw data in Supplementary Figure S1a. After normalizing and optionally filtering the current signal, the input to this pass is a vector of indices from 0 to n-1, where n is the number of data points that make up a single event. It is important to note that we are using a separate AdaBoost model for each individual event, rather than applying a single model across multiple events, as the size and characteristics of each event can vary, and different signals may correspond to distinct physical processes occurring within the nanopore. The expected output for this model is the current signal, normalized to [0,1. The training is done using the default parameters in the scikit-learn package and the values set by the user to limit the number of weak learners and their tree height, as discussed in Supplementary Section S6. We do not expect the model to give a perfect piecewise constant function at this step; rather, we want an overfit from this pass so as to not lose any physical sublevels while still simplifying the input. This is shown in Figure 3b.

Pass 2: Decision Trees

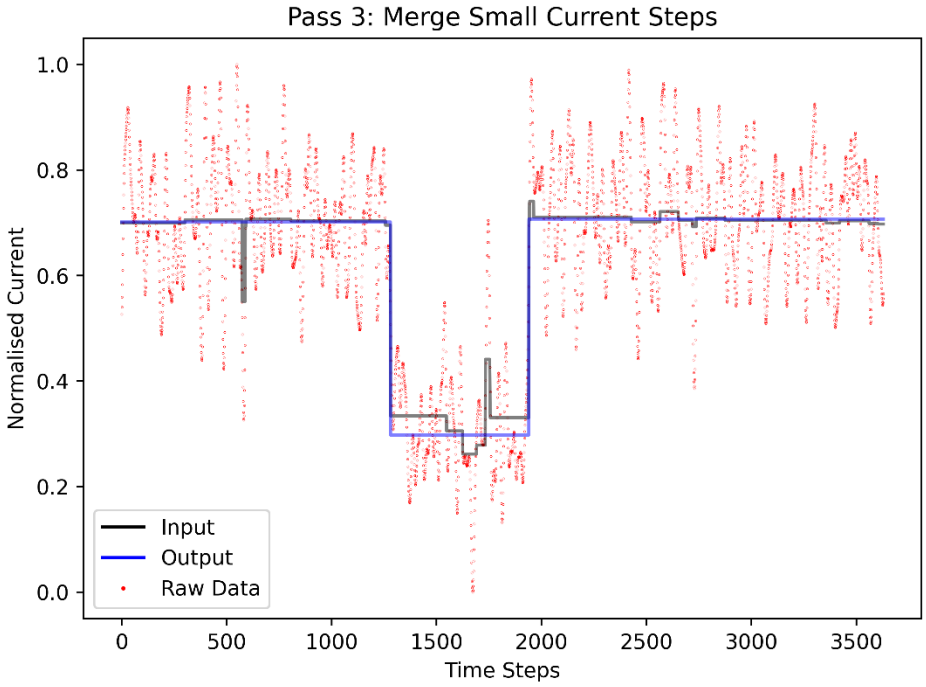


Supplementary Figure S2: Example current caused by the passage of a partly folded 2 kbp dsDNA polymer. Input and output of Pass 2 – Decision Trees. The input to pass 2 (black line) is the output from pass 1. The output (blue line) is an overfitted but improved fit to the normalized raw data (red dots).

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 vector from AdaBoost as our input, we apply a single decision tree from the scikit-learn package[6] to further reduce the noise in the signal to represent the piecewise constant function more closely with the number of pieces in it being as close to the real count of sublevels as possible in the same manner as Pass 1, using a vector of indices for training and prediction, but this time the output of pass 1 is the expected outcome during the training of a single decision tree model per event.

AdaBoost with decision trees in Pass 1 provides a gradual smoothing to not remove any sharp spikes in the signal that could correspond to a real sublevel. But it was observed that after a fine smoothing in Pass 1, a single decision tree in Pass 2 with a stronger ability to generalize performed better in reducing even more noise, which is now even more clearly distinguishable from actual narrow sublevels because of Pass 1. The output is close to our final fit visually, but usually still contains some nonphysical sublevels which are identified and corrected in subsequent passes. Supplementary Figure S2 shows an example of this pass.

Pass 3: Merge Small Current Steps



Supplementary Figure S3: Example normalized current (red dots) caused by the passage of an unfolded 2 kbp dsDNA polymer. Input and output of Pass 3 – Merge Small Current Steps. The input (black line) is the output from pass 2, while the output (blue line) is a smoothed version of the fit with sublevels that have unphysically small steps merged between them.

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 Supplementary Figure S3. This pass performs 3 major tasks, listed below.

Subpass 3.1 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 about 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 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 minimal asymmetry in the sign of the resulting errors.

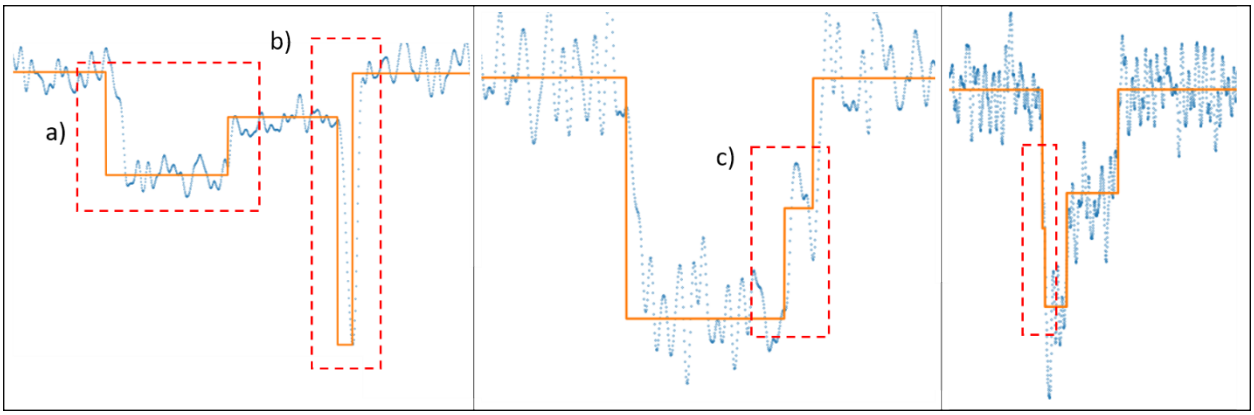
Subpass 3.2 Merging

Up to this point, the algorithm still tends to produce overfits. These are smoothed over by combining neighboring 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. The algorithm for this procedure is discussed in detail in Supplementary Section S3.

Subpass 3.3 Refresh 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 locally fitted current estimate by more than this value, using the default sublevel current estimation function, as detailed in Supplementary Section S2.

Pass 4: Categorize and Correct Sublevels with Short Durations

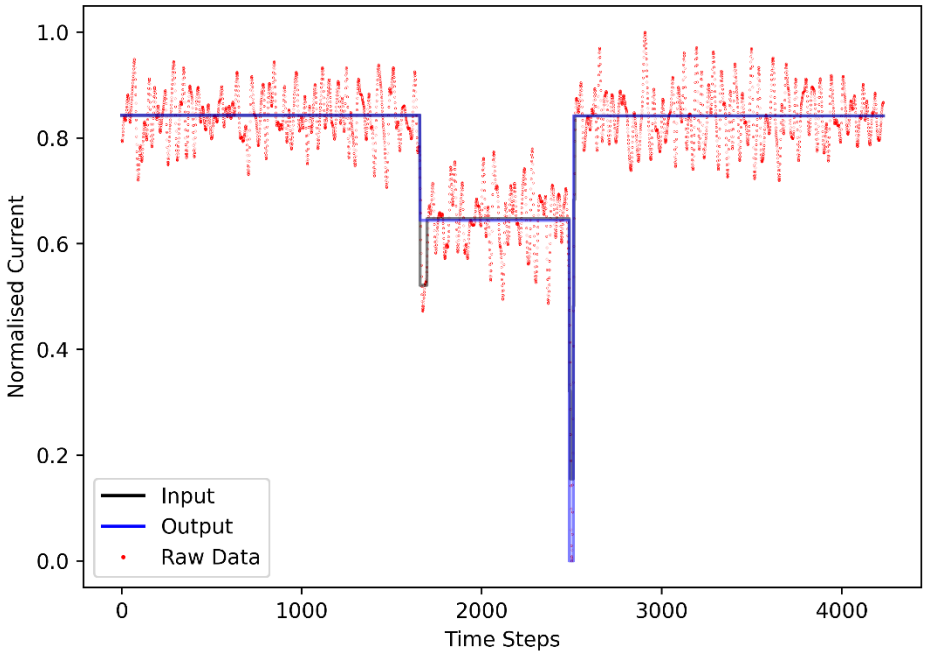


Supplementary Figure S4: Examples of all four types of sublevels defined in nanopore data. a) Normal sublevel (lasting long enough to reach a steady state), b) Peaked sublevel (not lasting long enough to reach a steady state between current steps in opposite directions), c) Sloped sublevel (not lasting long enough to reach a steady state between two current steps in the same direction), and d) Bad sublevel (not classified into one of the previous categories).

It is often challenging to identify whether a short sublevel represents a physical change. Simply removing them is not a solution as physical sublevels can often have 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 (Supplementary Figure S4a), and do not require correction in this pass. This is determined by the "*minDataPointsToBeSubLevel"* hyperparameter.
2. Peaked sublevels: Sublevels that do not last long enough to reach a steady state between current steps in opposite directions (Supplementary Figure S4b).
3. Sloped sublevels: Sublevels that do not last long enough to reach a steady state between two current steps in the same direction (Supplementary Figure S4c).
4. Bad sublevel: Any sublevel not classified into one of the previous categories (Supplementary Figure S4d).

Full details are available in Supplementary Section S4.



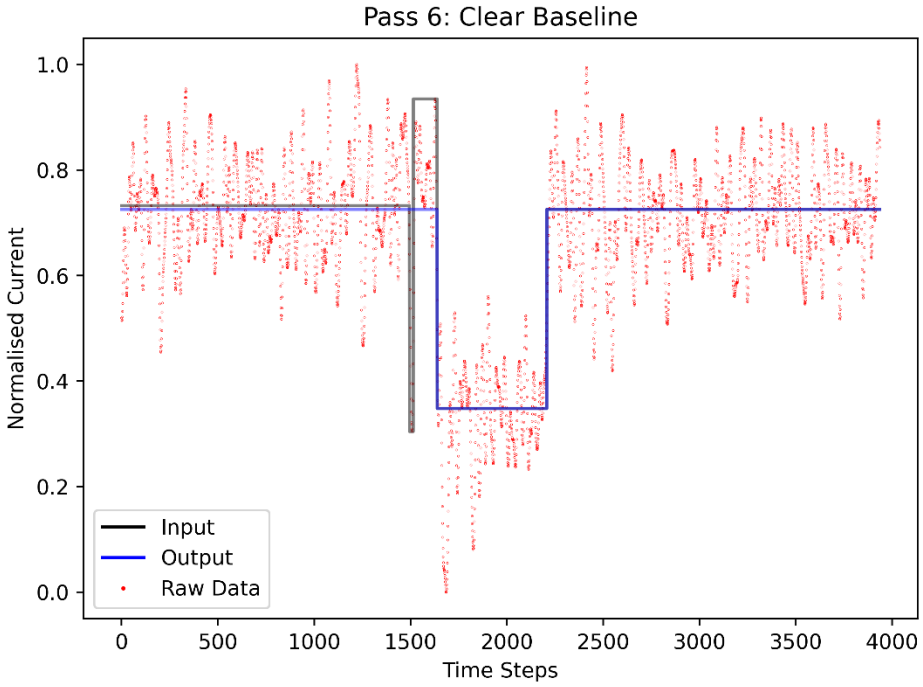
Supplementary Figure S5: Input and output of Pass 4 – Categorize and Correct Sublevels with Short Durations. The example current trace (red dots) is generated from passing 12-arm star nanostructures attached to a 2 kbp dsDNA carrier through a ~13 nm pore (3.6 M LiCl, 75 mV) with a measurement bandwidth of 1 MHz (sampled at 4.17 MHz) and digitally low-pass filtered for fitting at 250 kHz. The input (black line) is the output from pass 3, while the output (blue line) is an improved fit that corrects underestimates of current steps for sublevels that approach the time resolution of the system.

After labelling as above, bad sublevels are merged with neighboring sublevels using the procedure discussed in Supplementary Section S5. The hyperparameter "*numberOfStdAboveAndBelow*" is used to decide exactly where such sublevels are split. Some peaked and sloped sublevels are also merged, as determined by the additional hyperparameters discussed in Supplementary Section S3. Supplementary Figure S5 presents an example of these steps.

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 the Pass 3 function.

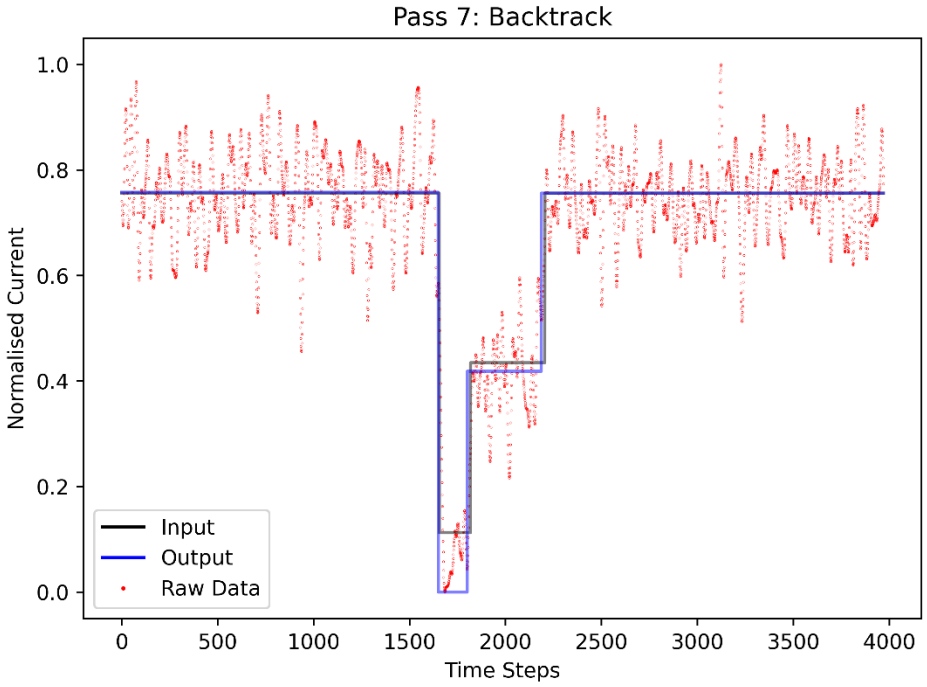
Pass 6: Clear Baseline



Supplementary Figure S6. Input and output of Pass 6 – Clear Baseline. Example normalized current trace from the translocation of unfolded 2 kbp dsDNA. The input (black line) is the output from pass 5, while the output (blue line) has suppressed any sublevels that were detected before the beginning or after the end of the event.

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 known start and endpoint of the event, any sublevels that fall outside of these bounds are merged into a single baseline sublevel. Supplementary Figure S6 presents an example of this step.

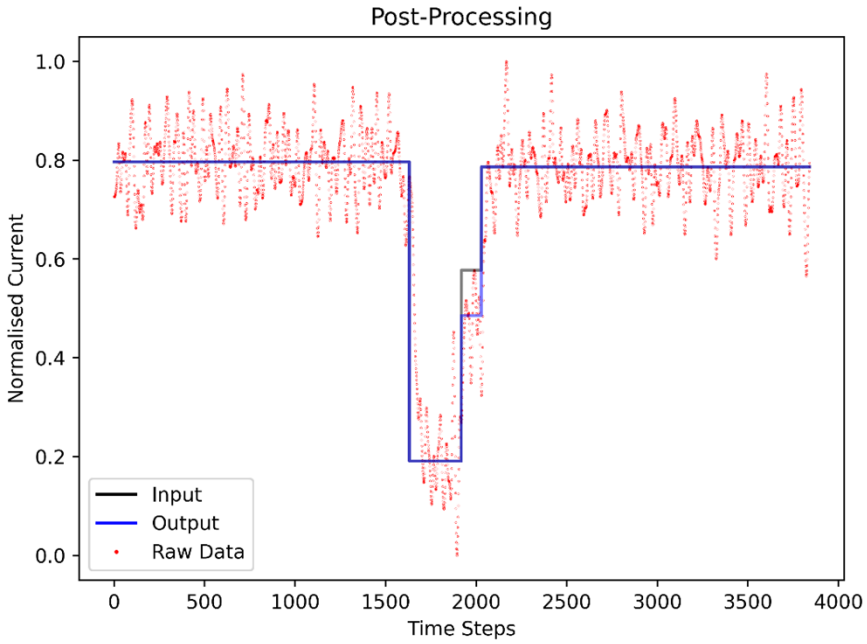
Pass 7: Backtrack



Supplementary Figure S7: Input and output of Pass 7 – Backtrack. Example normalized current trace from the translocation of folded 2 kbp dsDNA. The input (black line) consists of the output from pass 6, while the output (blue line) corrects minor errors in the time-indices of the changes between sublevels that arose from previous passes.

Having now identified all physical sublevels in the event, we must correct their start and end time 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 present 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 present 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 Supplementary Figure S7.

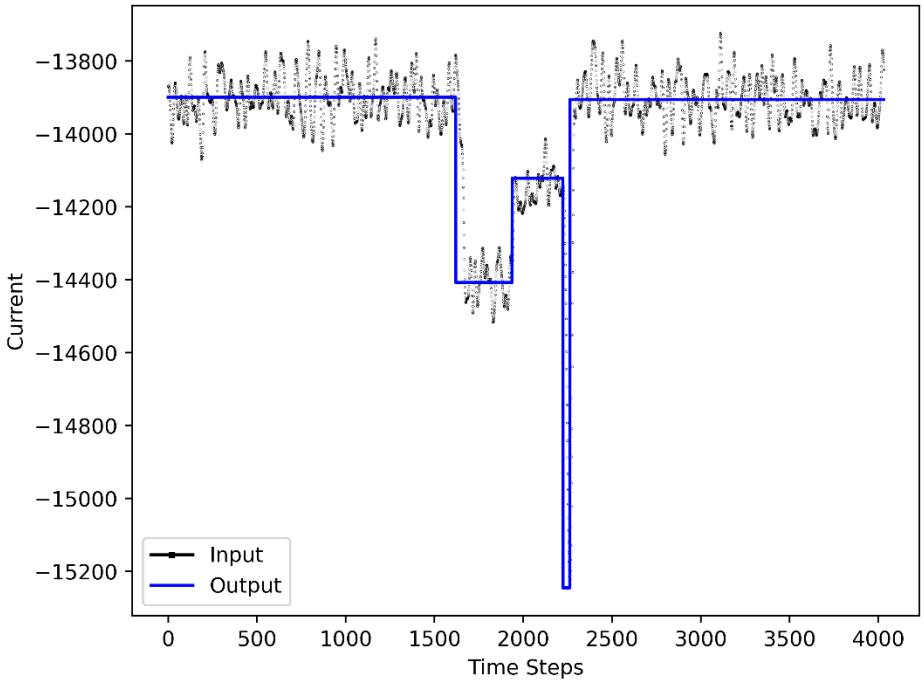
Post-Processing



Supplementary Figure S8: 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.

For the results presented in this work, we finish the fitting process 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 Supplementary Figure S8. The event is then denormalized using original maximum and minimum values. Finally, event-based information, including the number of sublevels, sublevel changepoints, and sublevel current estimates, is extracted and stored for further testing and analysis. Supplementary Figure S9 represents a sample fit of this entire algorithm.



Supplementary Figure S9: A sample event fit by the Nano Trees algorithm for the same current trace as in Figure 3.

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 approximation (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.

Supplementary Section S2: Updating current estimates

Algorithm 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: $sublevelHeight$

Supplementary Section S3: Merging small current steps.

Algorithm 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: $sublevels$

Supplementary Section S4: Sublevel categorization

Algorithm 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
$exceptionalPeak\_MinHeightStdAboveAndBelow$
$exceptionalPeak\_WidthLowerBound$
$exceptionalPeak\_BaseDifferenceStdAtleast$
$exceptionalSlope\_MinHeightStdOfMinDiff$
$exceptionalSlope\_WidthLowerBound$
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 $minDataPointsToBeSubLevel$
    $checkC5 \leftarrow |nextHeight\_nonExceptional - prevHeight|$
    $c5 \leftarrow checkC5 > exceptionalBaseDifferenceStdAtleastHeight$
    if $c5$ **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.

Supplementary Section S5: Splitting sublevels

Algorithm 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(*sublevel*1, *sublevel*2)
    Apply Algorithm 4.1 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**
        $check \leftarrow |sublevels[i].height - sublevels[i+1].height|$
        if ALGORITHM 4.3($sublevels[i]$) = $false$ **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

Supplementary Section S6: Glossary of hyperparameters

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 ± 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 is it 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 only 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 end point.)

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.

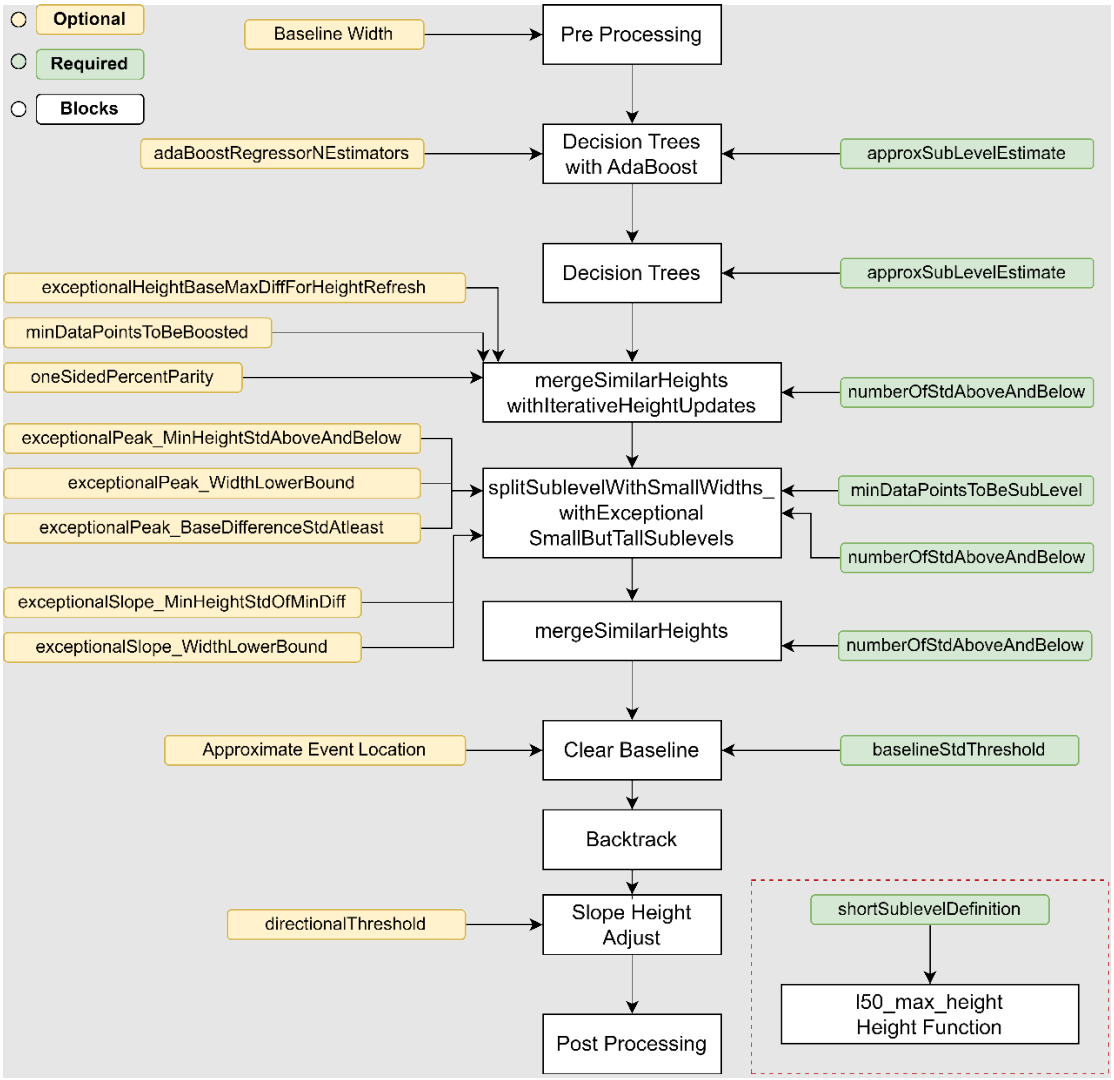
Supplementary Section S7: Hyperparameter tuning

The simplest way to tune hyperparameters is to select a few representative example events and set the parameters while visualizing the resulting fits. This process is cumbersome, and development is ongoing to simplify the process. 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 value, 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.

For the purpose of correlating function names used in the code and the pass names used to refer them in this research, the following mapping can be used, Supplementary Figure S10 represents the use of these names with the corresponding hyperparameters used along with their importance:

- 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



Supplementary Figure S10: Nano Trees pipeline with all functions and corresponding hyperparameters.

The hyperparameters can be divided into two categories based on their importance in the fit as follows:

1. Required Hyperparameters – these parameters cannot be left at their default values and need to be specifically tuned to each experimental context.
2. 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:

1. 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 the 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.
2. Experiment specific hyperparameters – these hyperparameters require a good adjustment for different experiments based on what biomolecules are used in the experiment, expected sublevels, baseline/noise characteristics, etc. Any major issue while setting these hyperparameters can degrade the fit quality significantly.

Hyperparameters used in this work are broken down according to these descriptions in Supplementary Tables S1 and S2.

Supplementary Table S1: Description of combination of required, optional, setup specific, and experiment specific hyperparameters.

Required Hyperparameters	Less tuning required but tune them well at least once for every new setup.	Change these for every experiment to get the best fits. 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.
	Setup specific hyperparameters	Experiment specific hyperparameters

Supplementary Table S2: Hyperparameters classified into 4 categories based on Table S1 descriptions.

Required Hyperparameters	numberOfStdAboveAndBelow baselineStdThreshold	approxSubLevelEstimate minDataPointsToBeSubLevel shortSublevelDefinition
Optional Hyperparameters	exceptionalHeightBaseMaxDiffForHeightRefresh oneSidedPercentParity directionalThreshold	adaBoostRegressorNEstimators minDataPointsToBeBoosted exceptionalPeak_MinHeightStdAboveAndBelow exceptionalPeak_WidthLowerBound exceptionalPeak_BaseDifferenceStdAtleast exceptionalSlope_MinHeightStdOfMinDiff exceptionalSlope_WidthLowerBound (Approximate Event Location)
	Setup specific hyperparameters	Experiment specific hyperparameters

Default Values & effect of increasing and decreasing each hyperparameter during tuning-

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 baseline standard deviation, so if 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. Since parity determines the ratio of datapoints above and below the current sublevel estimate, a higher threshold means fewer 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 sublevel 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 "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 threshold for the minimum number of data points an exceptional slope sublevel requires to not get deleted. A sublevel falls to the category of being exceptional if it has less than "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.

<u>Pass 6: Clear Baseline</u>

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.

Post-Processing

directionalThreshold

- Default Value: 0.5
- Increasing it increases the threshold a sublevel must cross for the ratio of maximum between the count of increasing and decreasing values and the number of points in the sublevel for its 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.

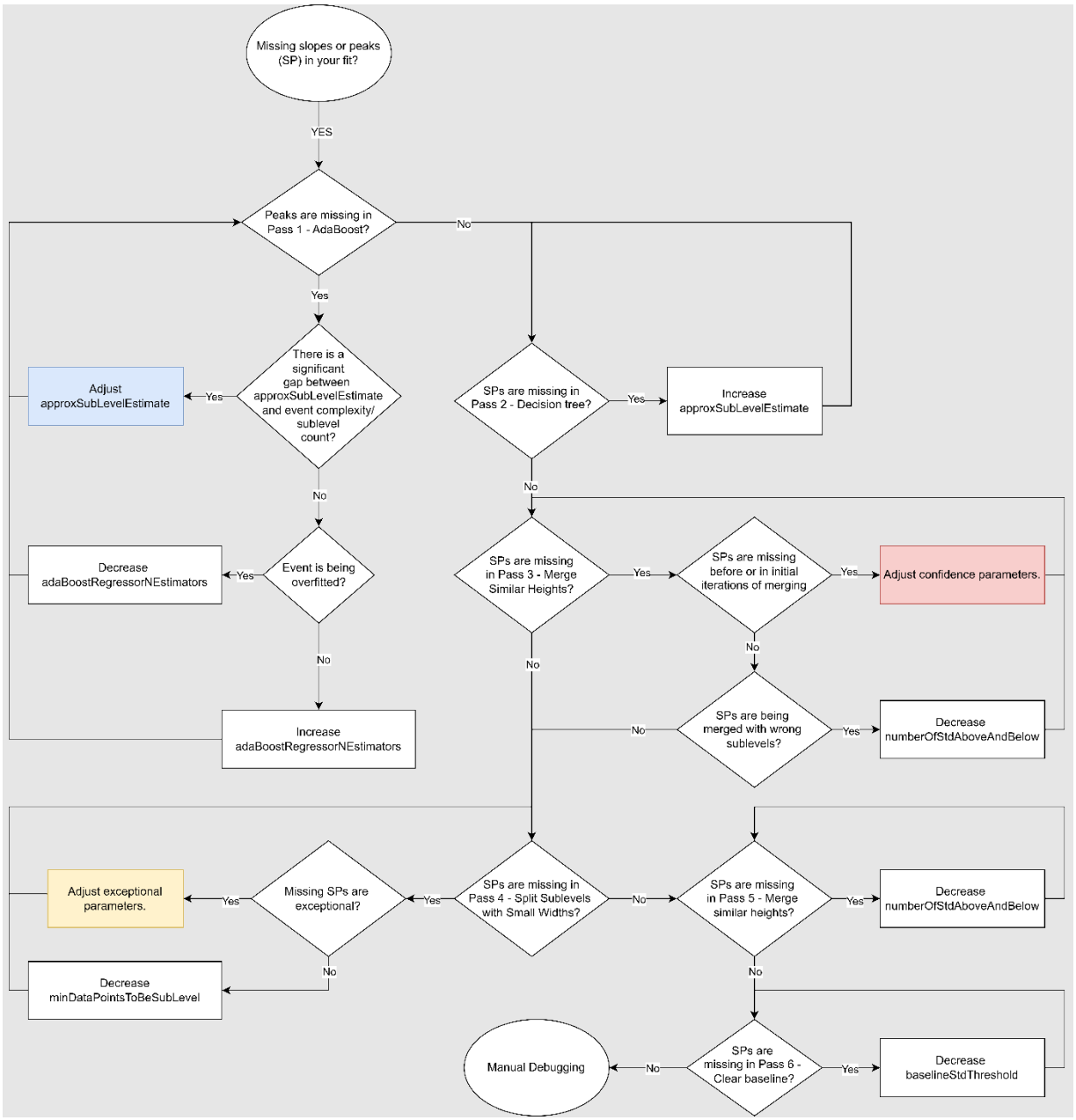
Sublevel Current Estimation Function

shortSublevelDefinition

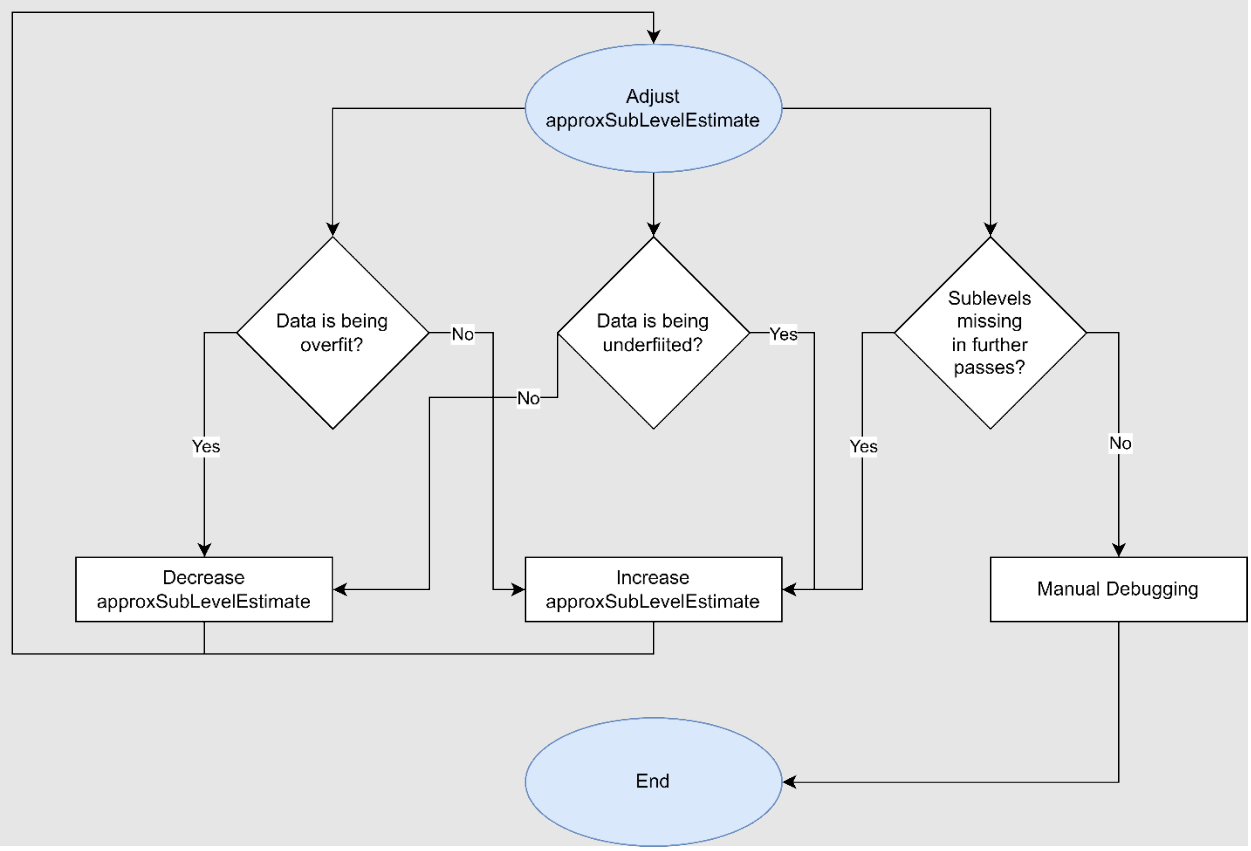
- Default Value: 2% event length
- Increasing it raises the threshold under which a sublevel is termed as short and its sublevel current estimate is set as the extreme value in the sublevel, and over this it is calculated using the mean of last 50% of the sublevel. Increasing it to a very high value 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.

Pass 5 has the same configuration as Pass 3 and Pass 7 does not have any hyperparameters. Debugging hyperparameters to fix specific issues can be a complicated task, Supplementary Figures S2-S8 describe steps for some common hyperparameter tuning procedures.

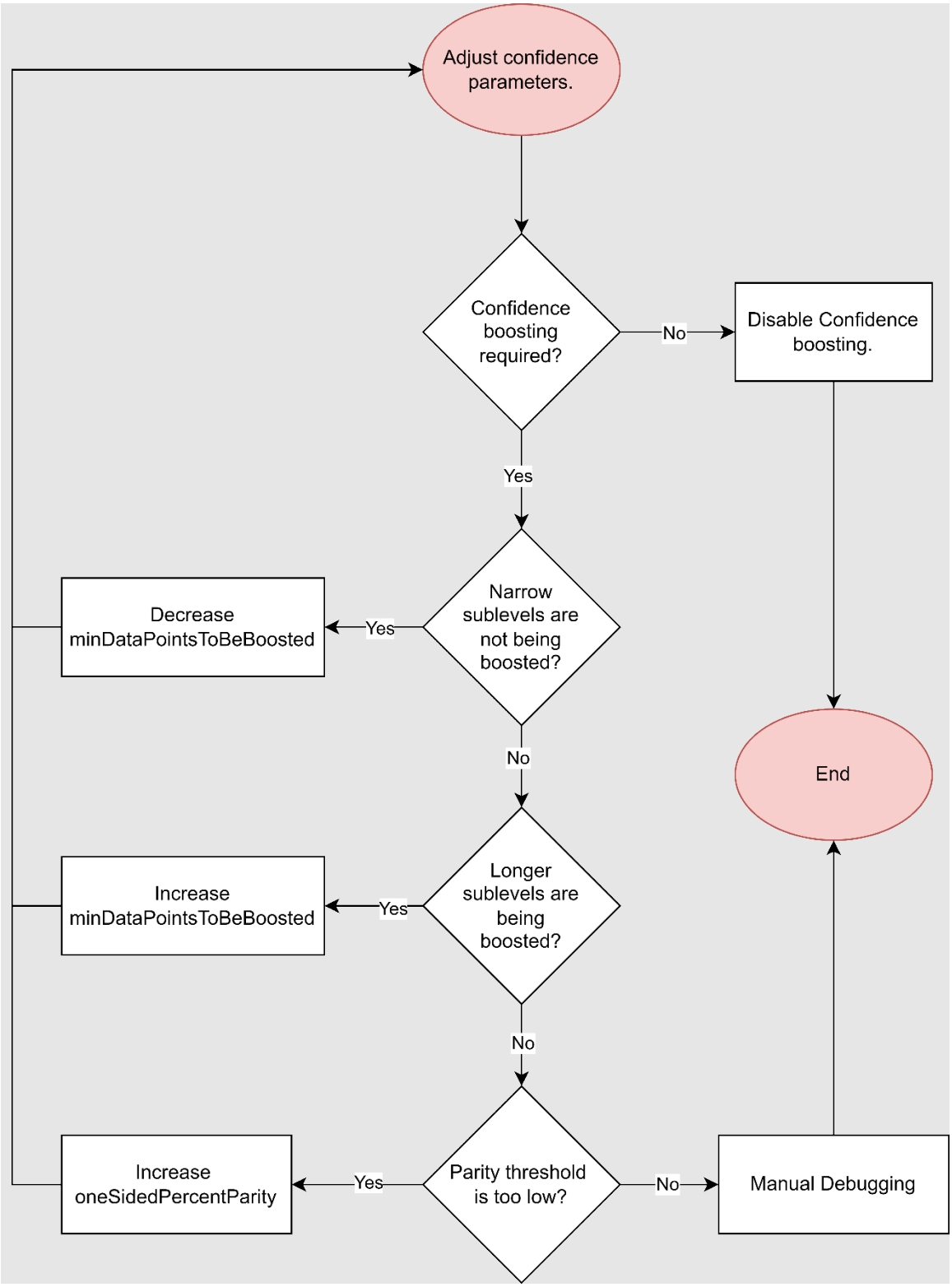
Supplementary Figures S11-S17 present flow charts for procedural optimization of fit parameters in new experimental contexts.



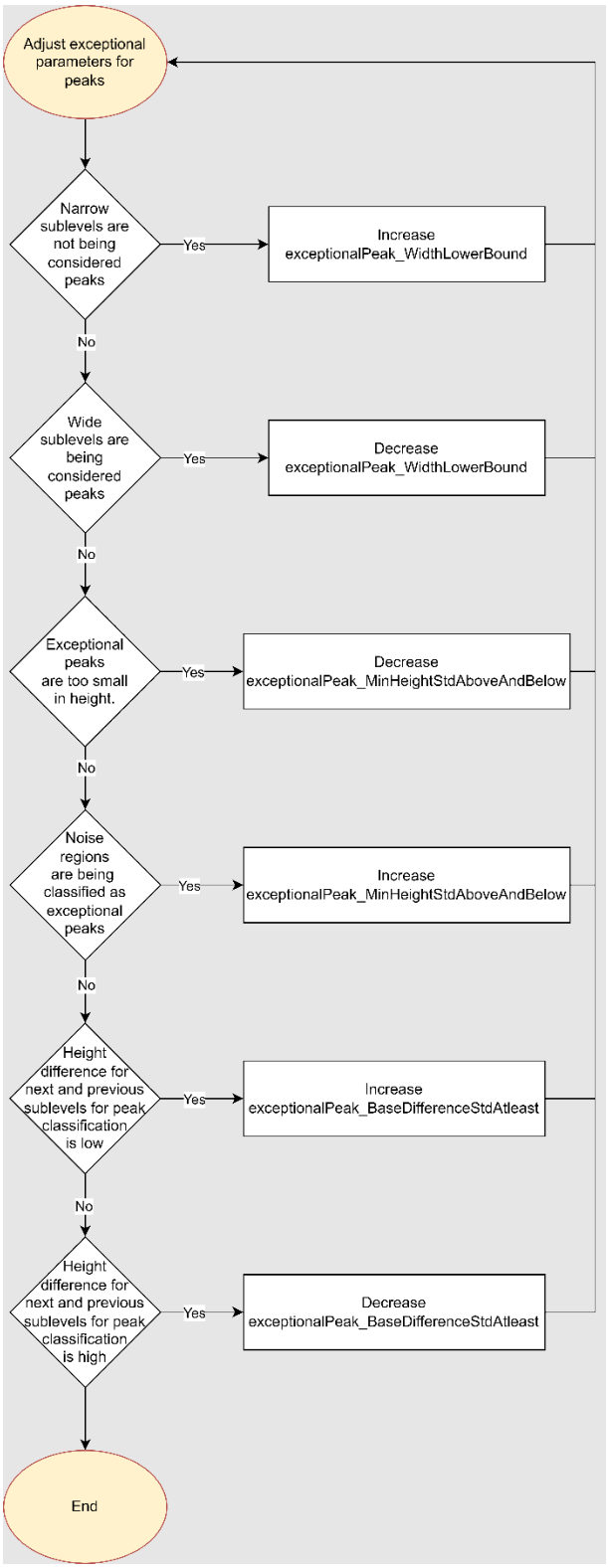
Supplementary Figure S11: Flow chart describing debugging steps to improve the hyperparameters in case slopes or peaks are missing in the fit.



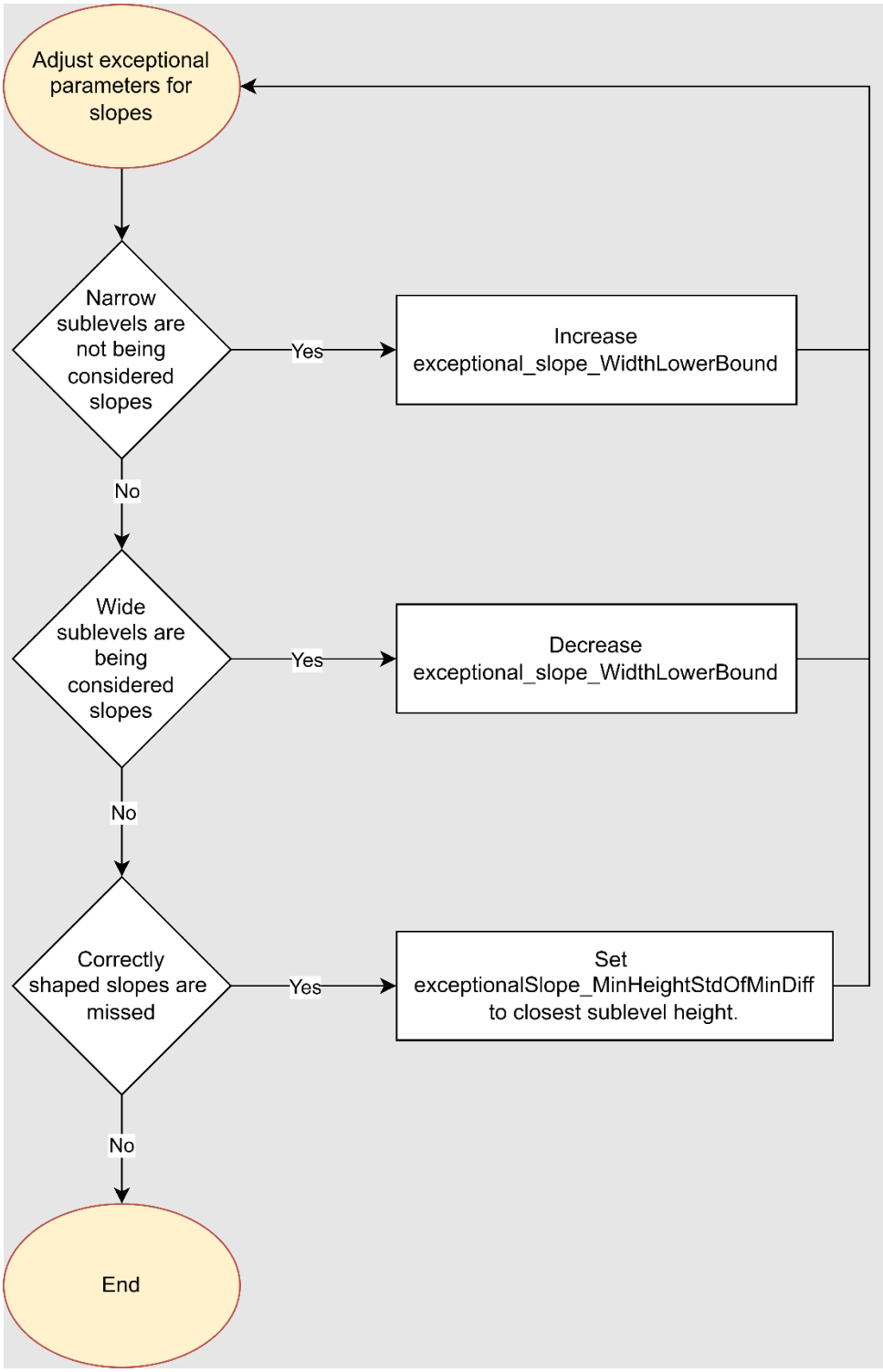
Supplementary Figure S12: Flow chart describing debugging steps to adjust approxSubLevelEstimate.



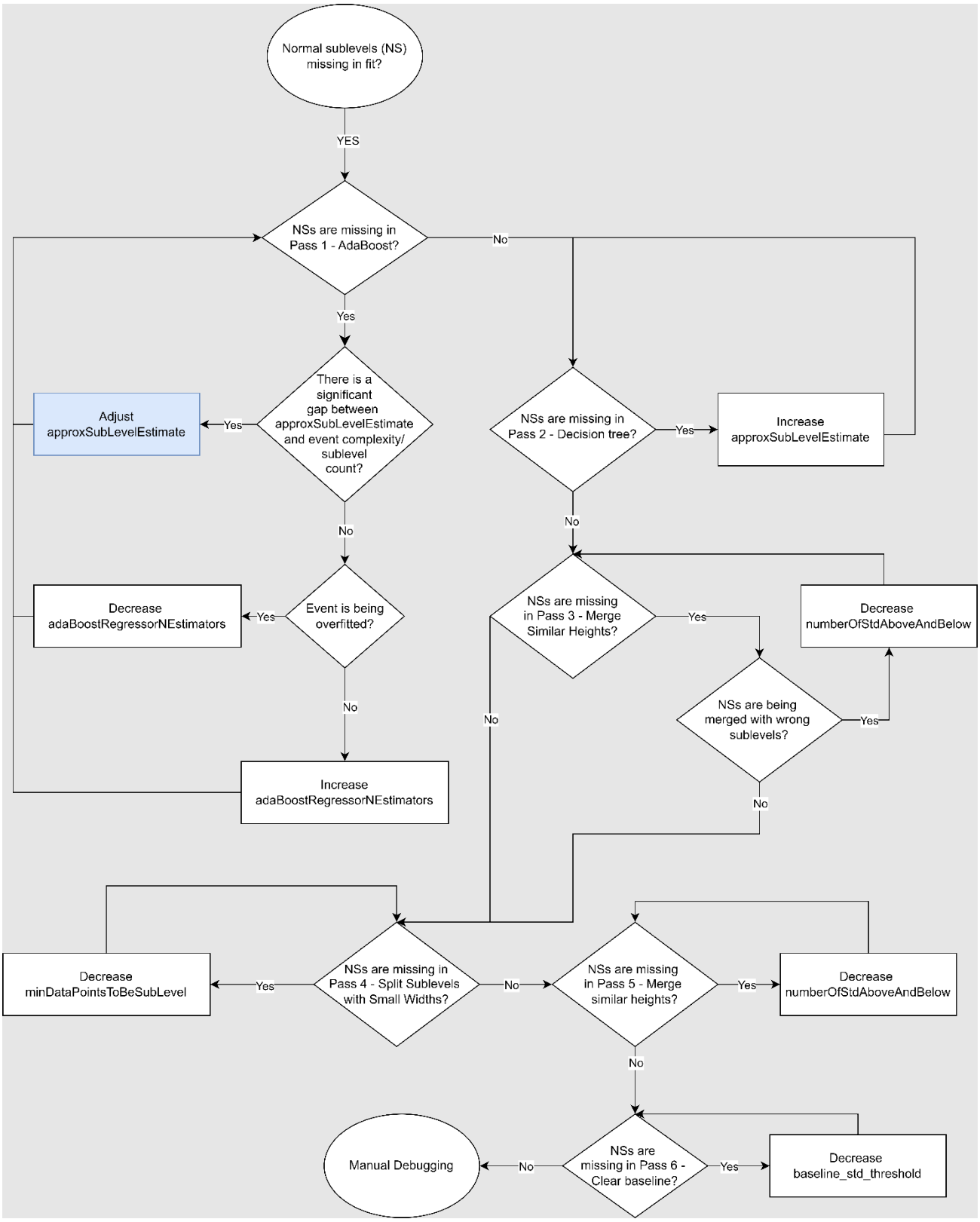
Supplementary Figure S13: Flow chart describing debugging steps to adjust confidence parameters.



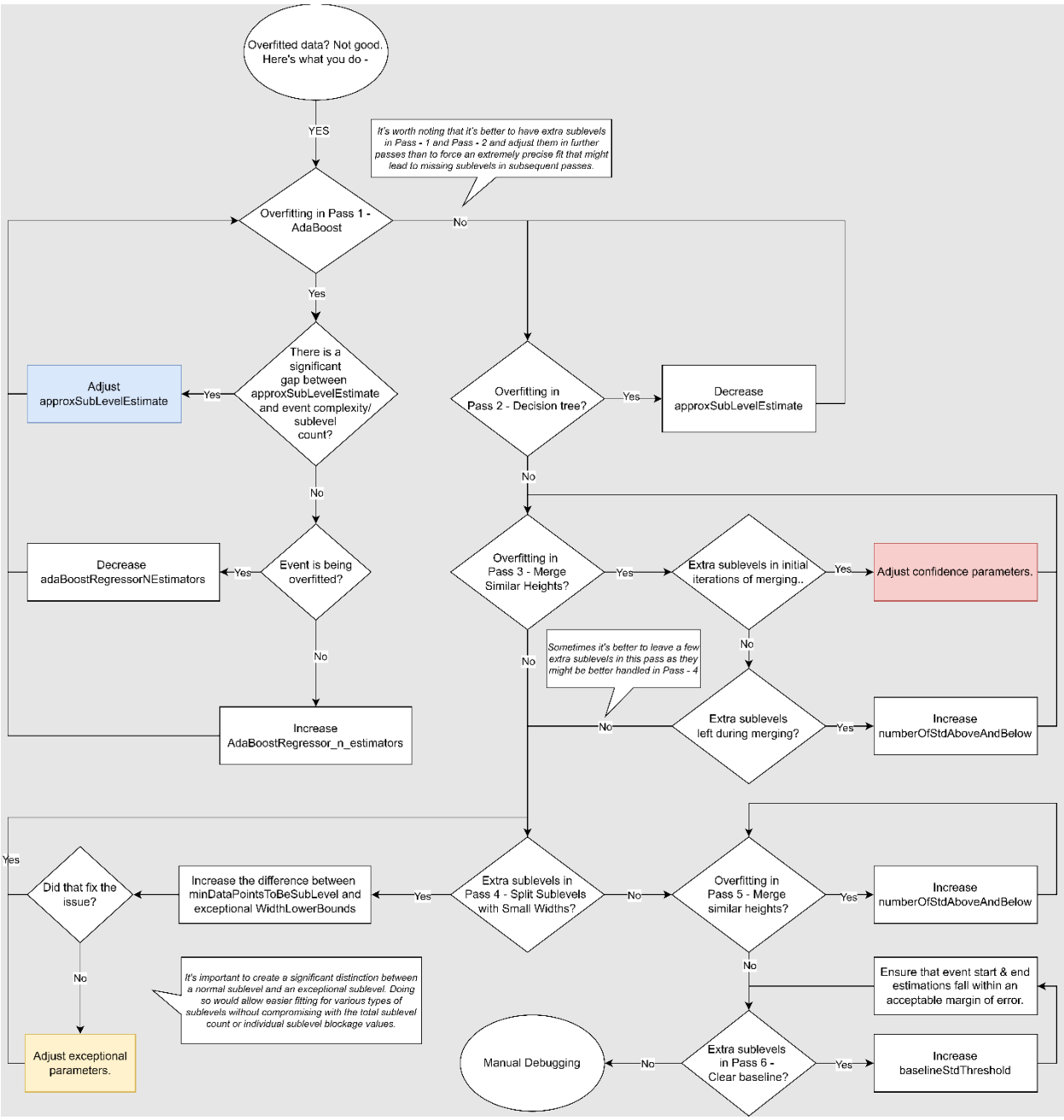
Supplementary Figure S14: Flow chart describing debugging steps to adjust exceptional parameters for peaks.



Supplementary Figure S15: Flow chart describing debugging steps to adjust exceptional parameters for slopes.



Supplementary Figure S16: Flow chart describing debugging steps to improve the hyperparameters if normal sublevels are missing in the fit.



Supplementary Figure S17: 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.

Supplementary Section S8: Hyperparameters used in this work.

Synthetic Dataset

Pass 1: Adaptive Boosting

- approxSubLevelEstimate = 10
- adaBoostRegressorNEstimators = 800

Pass 2: Decision Trees

- approxSubLevelEstimate = 10

Pass 3: Merge Small Current Steps

- numberOfStdAboveAndBelow = 2.5
- oneSidedPercentParity = 0.2
- minDataPointsToBeBoosted = 2% event length
- exceptionalHeightBaseMaxDiffForHeightRefresh = 0.35

Pass 4: Categorize and Correct Sublevels with Short Durations

- minDataPointsToBeSubLevel = 8
- numberOfStdAboveAndBelow = 2
- exceptionalPeak_MinHeightStdAboveAndBelow = 3
- exceptionalPeak_WidthLowerBound = 8
- exceptionalPeak_BaseDifferenceStdAtleast = 0
- exceptionalSlope_MinHeightStdOfMinDiff = infinity
- exceptionalSlope_WidthLowerBound = infinity

Pass 5: Merge Small Current Steps (Repeat)

- numberOfStdAboveAndBelow = 3.2

Pass 6: Clear Baseline

- baselineStdThreshold = 1.5

Post-Processing

- directionalThreshold = 0.5

Sublevel Current Estimation Function

- shortSublevelDefinition = 16

DNA Nanostructures Barcodes

Pass 1: Adaptive Boosting

- approxSubLevelEstimate = 4
- AdaBoostRegressorNEstimators = 500

Pass 2: Decision Trees

- approxSubLevelEstimate = 4

Pass 3: Merge Small Current Steps

- numberOfStdAboveAndBelow = 2
- oneSidedPercentParity = 0.2
- minDataPointsToBeBoosted = 20
- exceptionalHeightBaseMaxDiffForHeightRefresh = 0.35

Pass 4: Categorize and Correct Sublevels with Short Durations

- minDataPointsToBeSubLevel = 60
- numberOfStdAboveAndBelow = 2
- exceptionalPeak_MinHeightStdAboveAndBelow = 3
- exceptionalPeak_WidthLowerBound = 10
- exceptionalPeak_BaseDifferenceStdAtleast = 0
- exceptionalSlope_MinHeightStdOfMinDiff = 2.2
- exceptionalSlope_WidthLowerBound = infinity

Pass 5: Merge Small Current Steps (Repeat)

- numberOfStdAboveAndBelow = 4.5

Pass 6: Clear Baseline

- baselineStdThreshold = 1.5

Post-Processing

- directionalThreshold = 0.5

Sublevel Current Estimation Function

- shortSublevelDefinition = 80

Supplementary Section S9: Dataset Summary

Raw data used in this work is provided in the Federated Research Data Repository, available at https://doi.org/10.20383/103.01212.

Synthetic Dataset

Synthetic data was generated using Python's numpy==1.26.3 and scipy==1.11.4 libraries in Python 3.11 to simulate nanopore signal behavior. A piecewise constant signal was constructed and modified using a response function based on previously published nanopore measurement models. Zero-mean white noise was added to achieve a signal-to-noise ratio of 6, ensuring distinguishable transitions. The signal was sampled at 5 MHz and low-pass filtered to 1 MHz using a 4-pole zero-phase Bessel filter to remove high-frequency components. The dataset includes 31,500 events across seven distinct classes of transient signal features, with Supplementary Script 1 provided alongside this work for reproducibility.

DNA Nanostructures Barcodes

This dataset was collected using a Chimera VC100 instrument at a sampling rate of 4.1667 MHz. It contains the translocation of specially designed molecules consisting of a double-stranded DNA backbone with specific binding sites for side chains. These side chains carry DNA nanostructures shaped like stars, with either 4 or 12 arms, which generate distinct signal blockages when passing through a nanopore. The molecules are deliberately asymmetric to allow for clear identification of their orientation during translocation. Events are labeled as "10" or "01" based on which star—4-arm or 12-arm—appears first. The recorded signals were low-pass filtered at 250 kHz using an 8-pole Bessel filter implemented via scipy.signal.bessel() and scipy.signal.filtfilt() for noise reduction.

As this dataset lacks ground truth for its piecewise constant signal levels, a manual labeling process was used to evaluate the fitting accuracy of two algorithms: Nano Trees and CUSUM+. Each event was independently evaluated by a neutral third-party reviewer who judged the fit quality of both methods with a simple "Yes" or "No" response. These responses were used to calculate fitting accuracy, where a higher proportion of "Yes" votes signified better performance. In the dataset provided, the label applied by the expert reviewer is indicated by the name of the folder in which the dataset is found.

The folder names in which an event is found corresponds the category to which the event belongs, as manually assessed by an expert human reviewer.

References

(1) Freund, Y.; Schapire, R. E. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J Comput Syst Sci* **1997**, *55* (1), 119–139. https://doi.org/10.1006/jcss.1997.1504.

(2) Drucker, H. Improving Regressors Using Boosting Techniques. *Proceedings of the 14th International Conference on Machine Learning* **1997**.

(3) Hastie, T.; Friedman, J.; Tibshirani, R. *The Elements of Statistical Learning*; Springer New York: New York, NY, 2001. https://doi.org/10.1007/978-0-387-21606-5.

(4) Breiman, L.; Friedman, J. H.; Olshen, R. A.; Stone, C. J. *Classification And Regression Trees*; Routledge, 2017. https://doi.org/10.1201/9781315139470.

(5) Roelen, Z.; Briggs, K.; Tabard-Cossa, V. Analysis of Nanopore Data: Classification Strategies for an Unbiased Curation of Single-Molecule Events from DNA Nanostructures. *ACS Sens* **2023**, *8* (7), 2809–2823. https://doi.org/10.1021/acssensors.3c00751.

(6) Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; others. Scikit-Learn: Machine Learning in Python. *the Journal of machine Learning research* **2011**, *12*, 2825–2830.