AI 4 Science Discovery Network+

Group: 8
Challenge: Event Detection in Nanopore Data
AI4SD ML Summer School Report
20-24th June 2022

Project Team: Wole Ademola Adewole (University of Southampton), Halil Ibrahim Aysel (University of Southampton), Stephen Gow (University of Southampton), Zheng Jiang (University of Southampton), Dimitrios Stamatis (University of Southampton)

Report Date: 01/07/2022

AI4SD-SummerSchoolSeries:Report-6

Group: 8
Challenge: Event Detection in Nanopore Data
AI4SD-SummerSchoolSeries:Report-6
Report Date: 01/07/2022
DOI: 10.5258/SOTON/AI3SD0249
Published by University of Southampton

Network: Artificial Intelligence and Augmented Intelligence for Automated Investigations for Scientific Discovery
This Network+ is EPSRC Funded under Grant No: EP/S000356/1

Principal Investigator: *Professor Jeremy Frey*
Co-Investigator: *Professor Mahesan Niranjan*
Network+ Coordinator: *Dr Samantha Kanza*

Contents

1 Project Details

Group Number	8
Challenge Name	Event Detection in Nanopore Data
Project Dates	20-24th June 2022

2 Project Team

Names	Wole Ademola Adewole, Halil Ibrahim Aysel, Stephen Gow, Zheng Jiang, Dimitrios Stamatis
Employer name	University of Southampton
Work Emails	apa1g16@soton.ac.uk hia1v20@soton.ac.uk srg1f20@soton.ac.uk z.jiang@soton.ac.uk ds2e19@soton.ac.uk

2.1 Challenge Description

Translocation of DNA molecules through special membrane nanopores - under the application of electric current - results in change of the nanopore current [1,2]. The produced signals have characteristic patterns that depend on the type of event that takes place, i.e. the configuration of the DNA molecule that passes through the pore. The focus of this challenge is the development of methods for:

1. detecting the events
2. classifying the events based on the signal pattern
3. finding an appropriate measure for quantifying the success of the model

The provided data for this challenge consist of nine simulation sets produced by tuning the signal-to-noise ratio (SNR) at 100, 50, 10, 5, 4, 3, 2, 1.5, and 1. Each set (same SNR level) consists of five simulations of electric current time series, resulting in a total of forty-five simulations. An example plot of the data with the maximum SNR of 100 is shown in Figure 1, where the spikes in the data correspond to events. Zooming in on the graph at each of these spikes, it is possible to identify a range of three patterns corresponding to the different classes of event. As the SNR decreases, the true signals become more difficult to distinguish from the natural variation resulting from the noise. Therefore, devising a method to identify events when the SNR is low is a particularly important challenge.

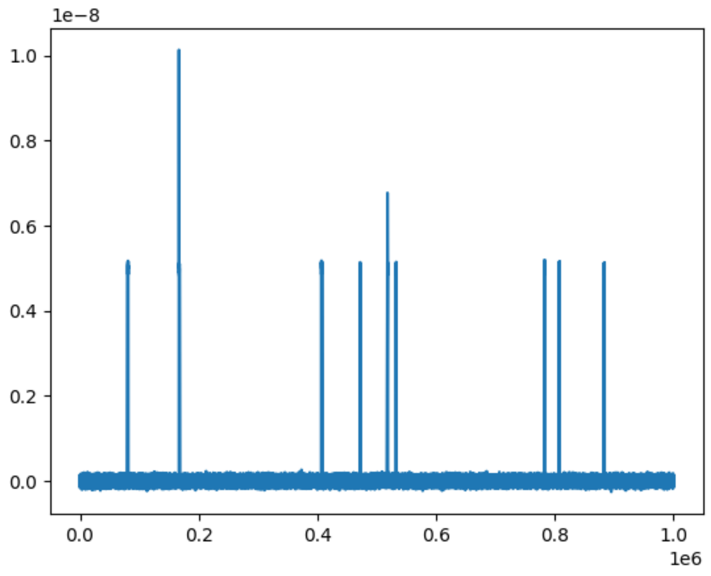

Figure 1: Plot of electrical current against time for a simulated data set with SNR=100.

3 Lay Summary

Detection of signal peaks (events) with the use of a static threshold is a simple and reliable method for cases with high SNR, i.e. when the peaks are well-separated from the noise. Nevertheless, in cases where SNR is lower than 10, it is easy to see that the performance of this method quickly deteriorates. In this work, we developed a detection method based on dividing the simulation into small, overlapping windows in time and determining when an event occurs by considering the number of points per window in which a threshold is exceeded, and combining this information between nearby windows to identify the start and end points of each detected event. This method requires several parameters, which were approximately optimised manually and tested on different data sets with the same SNR. Across four test data sets at the lowest SNR of 1, the method detects the majority of events present with a false positive rate close to 0, and an Area Under the Receiver Operating Characteristic curve (ROC AUC) in the range 0.83 to 0.89.

4 Methodology

Since a static threshold is only appropriate for data with high SNR, the idea of an adaptive detection cutoff that takes into account both the noise and signal levels may seem a suitable solution, especially for cases where signal and noise peaks might be separated by a fine line. To implement this method, we considered using an adaptive threshold of the form:

$$T = N_{SD} + P \times (M - N_{SD})$$

where T is the threshold, M is the maximum peak value, N_{SD} is the standard deviation of the noise measured on the first 10% of the time series, and P is an adjustable value. For our tests, we used P values of $0.05, 0.1, 0.2$ and 0.3. Another idea explored was detecting the start and end of events by considering the change in output between consecutive time steps. Due to the presence of the signal, we would expect to see a large increase in the output at the start of each event, and a large decrease at the end. An event could then be defined by "matching" sharp increases and decreases which occur in a specified period of time. The risk of false positives would thus be mitigated, as it is unlikely that both an increase and a corresponding decrease would both occur due to the effect of noise alone. This would thus allow a relatively low threshold to be chosen so that the probability of detecting an event would be increased. The code to implement this is as follows:

```
#Vector of instantaneous changes
data1 = data[0:ndata-1]
data2 = data[1:ndata]
instchange = data2-data1

#Threshold on the change
thresh = 2.9e-10
change_est = np.zeros(ndata - 1)
for i in range(ndata-1):
    if instchange[i] > thresh:
        change_est[i] = 1
    elif instchange[i] < -thresh:
        change_est[i] = -1
```

Our final approach was based on a method that splits the given data into multiple overlapping windows in time. The underlying idea is that, while noise may be responsible for individual large observations outside of an event or low observations during an event, a collection of several high observations in quick succession is likely to indicate that an event is indeed taking place. Therefore, for each window, the number of observations above an appropriate (static) threshold was recorded. Then, windows with a large number of high observations were further processed to remove potential false positives and allow event durations to be determined.

The practical techniques required to do this are somewhat involved and merit a full discussion. First, a simple thresholding algorithm is set up with a significantly lower threshold than would be optimal if this were to be the end of the process. This gives a vector of ones and zeroes corresponding to the locations of time steps where the observed value was above or below the threshold respectively. Next, given a window size and distance between windows, the space is divided into the overlapping windows and the number of points above the threshold in each window is counted. If this exceeds the minimum number deemed worthy of further investigation, the locations of every point in the window in which the observed value exceeds the threshold are recorded, and the first and last time steps at which this occurred are added to vectors of possible event start and end locations. This process, which is still only half of the complete algorithm, is shown in the code below.

```
##Step 1 - basic thresholding with a low threshold
thresh = 1e-10
st_est = np.zeros(len(data))
for i in range(len(data)):
    if data[i]> thresh:
        st_est[i]=1

##Step 2 - identify windows where many points exceed threshold
winsize = 700
stepsize = 50
winthresh = 70
nwins = (ndata - winsize)//stepsize
poss_event_wins = []
poss_starts = []
poss_ends = []
for i in range(nwins):
    curr_win = st_est[i*stepsize:(i*stepsize+winsize)]
    if(np.count_nonzero(curr_win)>winthresh):
        poss_event_wins = np.append(poss_event_wins, i)
        highpts = curr_win > thresh
        highlocs = [i for i, val in enumerate(highpts) if val]
        poss_starts = np.append(poss_starts, i*stepsize+min(highlocs))
        poss_ends = np.append(poss_ends, i*stepsize+max(highlocs))
```

Having identified a set of possible event start and end points, we now need to determine which start and end point should be "paired" to make real events. The nature of the windows means that the first and last points detected may not be the true start or end of an event, as it is possible that there are more points within the same event which lie outside of the window in question. To remove some of these, we begin by filtering out all such points which occur in only one window, as any true change point will appear several times. This is done using the Counter function and a frequency dictionary.

```
##Step 3 - distinguish "real" event start/endpoints from artefacts

#Identify which possible endpoints occur in more than
    one window and remove others

from collections import Counter
FrqDict1 = Counter(poss_starts)
int_starts = []
for key in FrqDict1:
    if FrqDict1[key] > 1:
        int_starts.append(key)

FrqDict2 = Counter(poss_ends)
int_ends = []
for key in FrqDict2:
    if FrqDict2[key] > 1:
        int_ends.append(key)

int_starts = np.array(int_starts, int)
int_ends = np.array(int_ends, int)
```

The final step to identify real change points is to group the remaining points across time. This requires an additional parameter, the maximum possible event width. All remaining points are grouped into a single vector, which is then sorted and split into several entries based on the distance between neighbouring points. The resulting subvectors are checked for length, and those with length 1 are discarded. These are "isolated" start or end points for which no corresponding point of the opposite type was detected. Since no event can be associated with these, they are removed from further consideration. (Note that the loop required to do this counts backwards from the end of the vector, as counting forwards can lead to overflow errors when entries are removed.) For those subvectors with more than one point, the first and last entries are considered to be the start and end of an event, with everything in between being part of that event.

```
#Split possible changepoints by similar values and pick extremes
    of each group to remove spurious intermediates

all_changepts = np.concatenate((int_starts, int_ends))
all_changepts = np.sort(all_changepts)
all_changepts = np.split(all_changepts,
    np.reshape((np.where(np.diff(all_changepts)>2000)), -1) + 1)

for i in range(len(all_changepts)-1, -1, -1):
    if len(all_changepts[i]) < 2:
        all_changepts.pop(i)

nevents = len(all_changepts)
eventlocs = np.empty((nevents, 2))

for i in range(nevents):
    eventlocs[i, 0] = min(all_changepts[i])
    eventlocs[i, 1] = max(all_changepts[i])
```

Finally, for consistency with earlier methods, the resulting locations are mapped into a vector of ones and zeroes corresponding to the presence or absence of an event.

```
#Step 4 - convert start/ends identified into vector of 0s and 1s
    giving when events take place

event_est_meth2 = np.zeros(len(data))
for i in range(len(eventlocs)):
    startcurr = int(eventlocs[i, 0])
    endcurr = int(eventlocs[i, 1])
    event_est_meth2[startcurr:endcurr] = 1
```

The developed method relies on several input parameters: the detection threshold, the window size, the distance between windows, the minimum number of detections per window to be regarded as a candidate for a real event, and the maximum event width. These parameters were set manually, using information from the analysis of the events in the synthetic data (see

Results). The parameters required to achieve a reasonable performance were found to vary strongly with the SNR. For the hardest case of SNR of 1, the parameters used were as follows:

- Detection threshold: 1×10^{-10}
- Window size (time steps): 700
- Window spacing (time steps): 50
- Minimum number of detections: 70
- Maximum event width: 2000

It should be noted that these values were determined manually based on a single data set; several performance metrics were used in this calculation, including the ROC AUC score, the number of true events detected in some form, and the number of time steps which were categorised incorrectly. It is possible that better parameter sets may exist, and an automated method to determine the optimal values would be of substantial value to the effectiveness of the method.

The topic under consideration in this report has been the focus of substantial recent research, for example [3]. However, our research was conducted without considering the pre-existing body of work, as we wished to develop a novel approach to the problem.

5 Results

Some parameters for the detection method were tuned based on the characteristics of the synthetic data time series. In particular, analysis of the baseline and peak intensities shows that the standard deviation (SD) of noise remains similar across the simulations, while the intensity of the highest peak drops by more than tenfold as we move from simulations with SNR of 100 to data with SNR of 1 (Figure 2, left). This plot confirms that it is easy to separate signal peaks from noise for simulations with SNR as low as 10 (corresponding to simulation indexes 0-15). For lower SNR, the peaks are increasingly within three SDs of the noise, making the choice of an appropriate cutoff value rather difficult. Nevertheless, values close to 1×10^{-10} seem a reasonable choice. Additionally, all the signal peaks in the data have an average duration of 1005 ± 100 time steps (Figure 2, right).

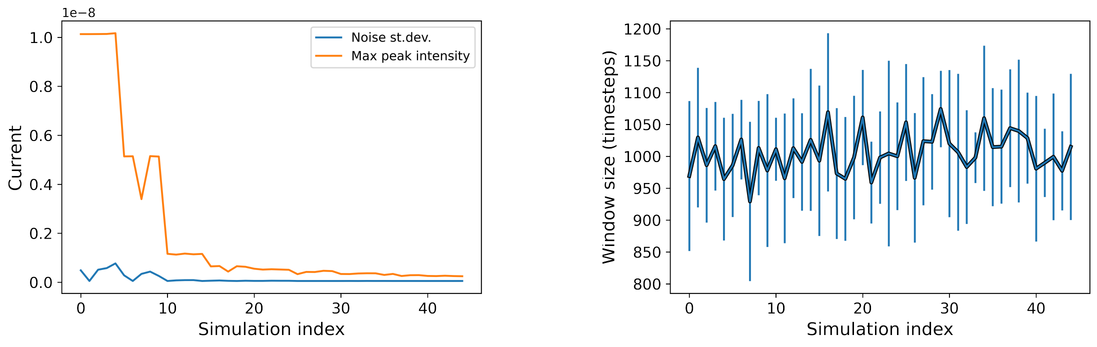

Figure 2: Maximum signal intensity and SD of noise (left), and mean±SD window size of event signals (right) across simulations.

By using an adaptive threshold (see Methods), we found that the success of signal detection does not deteriorate as fast as when using a static threshold (Figure 3). Nevertheless, by taking a closer look at the ROC AUC scores for low SNR simulations (indexes 16-45) and $P \leq 0.3$, we see that the relatively good scores of 0.62 or above arise from a degeneration of the ROC AUC metric, rather than an actually good performance of this detection method. For this reason, comparisons of the performance of the other methods - while considering ROC AUC score as one possible metric - also take into account other factors such as the number of true events detected in some form, and the number of time steps at which an incorrect classification was made.

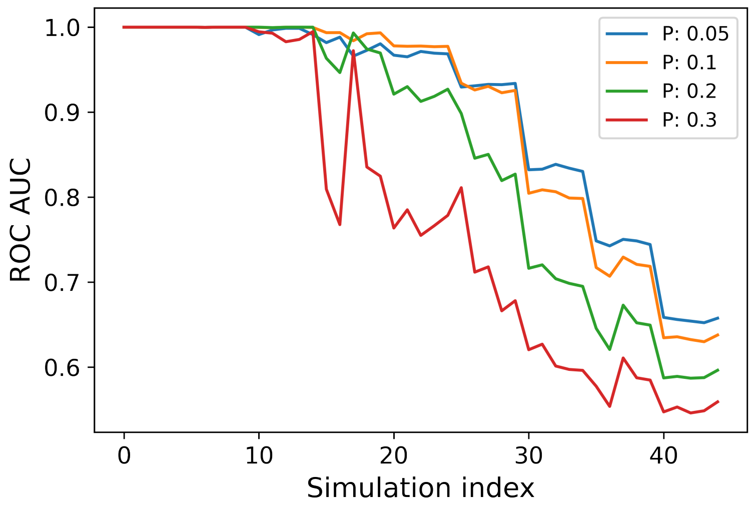

Figure 3: ROC AUC scores across all simulations using adaptive thresholds with different P factors.

The method of detecting events by searching for their start and end points was not found to be an effective method. Despite the reduced probability of false positives, the chance of missing real events is substantially higher than that of a simple thresholding approach, as there are now two ways in which an event may fail to be detected. This is depicted in Figure 4. Although it is possible to identify a few events from corresponding "starting" and "ending" signals, the vast majority of events were not correctly detected. Any further decrease in the threshold to identify more possible change points permits too many false positives, obscuring the effect of the true events.

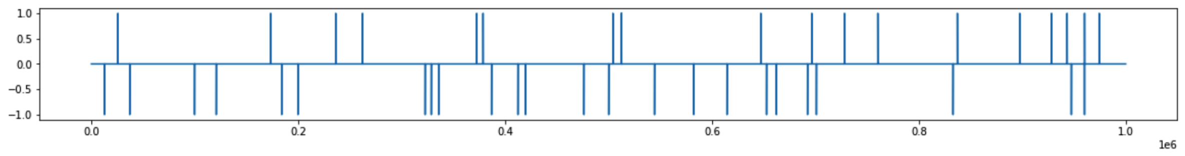

Figure 4: Large instantaneous changes in output for an example data set with SNR=1. Points with value 1 represent large increases; points with value -1 represent large decreases.

The method of overlapping windows was substantially more successful, managing to identify a majority of events even when the SNR was low. With parameters trained on a single data set with SNR=1, the method was able to generalise with surprising effectiveness to unseen data, achieving ROC AUC scores of between 0.831 and 0.887 on the other four data sets with the same SNR.

Consider for example the fifth data set with SNR=1. Given the parameters optimised (by hand) on the first data set at this level, the observed ROC AUC is 0.887 - significantly better than the best results available using a static threshold alone (ROC AUC close to 0.5, depending on the used threshold). Of the nine events in the simulation, the method identified seven, and produced plausible (although slightly incorrect) boundaries for their durations. In terms of detecting wider events, this method produced no false positives at all, as we would hope given the relatively strict process of identifying clusters of points where the threshold is exceeded.

It is informative to view the results of this approach graphically. In Figure 5, we plot the underlying data in blue, and the boundaries detected by the method in red. Given that all seven events detected were correct (at least in approximate location), this appears something of an impressive achievement, as due to the relatively high noise there is visually little difference between the locations of these events and the remainder of the simulation.

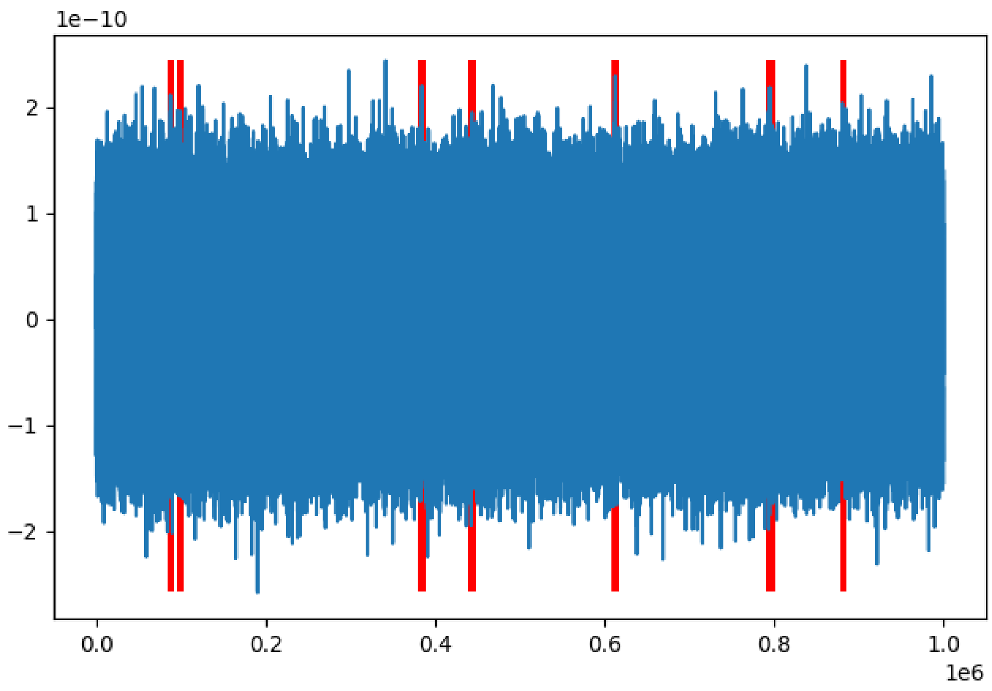

Figure 5: Detected events and observed data for a simulation with SNR=1.

It is important to note that the observed results are still far from perfect. Two events were missed altogether, and of the events detected, none were reproduced perfectly; in every case the start or end of the event (or both) were located a short distance from the true location. In most cases, the boundaries of the event were too wide. This could be due to a large observation (caused by noise) being located close to a true event and classified by our method as part of the event itself. Overall, the method made around $5,500$ detection errors across the entire simulation - small compared to the 10^6 total time steps, but relatively large in the context of only $9,134$ time steps during which an event was in fact taking place. Nonetheless, this is a substantial advance over simple thresholding, which achieves its best performance on this metric by predicting that there are no events at all during the simulation.

6 Conclusions & Future Work

The reliable detection and classification of signals is an extremely difficult task for cases with SNR of 5 or lower. To tackle this challenge, we attempted to develop several detection methods, such as an adaptive threshold algorithm, an instantaneous change detection algorithm, and an algorithm relying on overlapping windows. The last approach proved to be the most successful, as it resulted in the highest ROC AUC score, the smallest number of total classification errors, and a large proportion of correctly identified events even at the lowest SNR of 1.

Despite the reported success, there are many limitations to this method, and several ways in which it could perhaps be improved. Firstly, the five parameters of the method were chosen manually, and are likely to be quite different from the best possible configuration to maximise the effectiveness of the method. An automatic method to search for the best set of parameters would represent a significant advance, and may be possible using machine learning or design of experiment techniques. Additionally, one of the most common errors made by the method is overestimating the duration of events due to large values in the electrical current data close to - but not during - the events. This could perhaps be mitigated by introducing a second layer of smaller windows to determine when such anomalous points exist and consequently narrow the boundaries of a detected event.

Finally, the developed method does not help with the problem of classifying signals by event type. This challenge would need a different approach, for example by considering the behaviour of the data within an identified event. One way to do this could be to search for extreme changes to distinguish non-linear events from the more common linear events - for example a very large but short-lived peak in the case of a looped molecule. This of course depends on the boundaries of the event being well-defined, and would thus also rely on further improvements to the event detection method. Other approaches could involve the use of algorithms that match patterns of complex signal features, such as the peak area and/or the areas of amplitude increase and decrease, to those of peaks with shapes representative of the event types of interest.

References

[1] Raillon C, Granjon P, Graf M, Steinbock LJ, Radenovic A. Fast and automatic processing of multi-level events in nanopore translocation experiments. Nanoscale. 2012;4:4916-24. Available from: http://dx.doi.org/10.1039/C2NR30951C.

[2] Wang C, Sensale S, Pan Z, Senapati S, Chang HC. Slowing down DNA translocation through solid-state nanopores by edge-field leakage. Nature communications. 2021;12(1):1-10.

[3] Wen C, Dematties D, Zhang SL. A Guide to Signal Processing Algorithms for Nanopore Sensors. ACS sensors. 2021;6(10):3536-55.

List of Figures