Ideas for the Linac3 Source ML Analysis

Max Mihailescu

This document provides a collection of ideas and topics for the analysis of the Linac3 Ion Source from a ML viewpoint. I will try to collect descriptions and links to interesting papers and summarize the results I had when trying out some of them.

Clustering

Given a collection of data points, clustering is to group together points that are similar under some kind of similarity metric. Usually, this is an unsupervised technique, meaning that no reference labels are known. There exist a variety of different algorithms, and each algorithm can produce very different results on the same set of data. Therefore it is crucial to somehow evaluate the results.

For Linac3 we performed a Clustering Analysis with the goal, to see, if certain settings of the source would lead to a stable beam current. For the resulting report please contact Detlef Küchler (CERN BE-ABP-HSL).

The clustering algorithm we used is called *Optigrid* and is described in the paper "Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering" by Alexander Hinneburg and Daniel A. Keim [8].

Matrix Profile

The Matrix Profile is a tool for efficient motif discovery in time series, i.e. for discovery of repeated or "conserved" patterns. It can also be used to find time series discords, i.e. anomalies. A large number of papers was published by the same working group, you can find the projects website here [9]. For a good introduction with example applications read the first paper [24].

What is the Matrix Profile?

First, let's understand what a subsequence is. Consider a ordered sequence of real values $T = T_1 ... T_n$ of length n (what we typically call a time series, although the ordering can also be something else, like a DNA sequence), and fix a number $m < n$ (typically $m \ll n$). Now, a subsequence of length m starting at i is the continuous block $T_i ... T_{i+m}$. By sliding a window of size m over T we can get all $n - m + 1$ subsequences of length m.

Now, we can define the distance between two subsequences. The most obvious choice is the euclidean distance $dist(Q,T) = \sqrt{(Q_1 - T_1)^2 + \cdots + (Q_m - T_m)^2}$. However, for the matrix profile we work with the so called *z-normalized* euclidean distance. It is the euclidean distance of the z-normalized subsequences, i.e. we subtract the mean of the subsequence and divide the difference by the subsequence' standard deviation, and only then take the euclidean distance. By this we rescale all subsequences to make them more compareable. Note that under some circumstances this might be undesirable, for example when search for motifs in the call of a bird where the pitch could be relevant, and not only the shape of a signal.

The Matrix Profile P is a meta time series, that for every subsequence stores the smallest distance to some other subsequence. Furthermore, the Matrix Profile Indices I is another meta time series, that for every subsequence stores the starting index of the subsequence, that has the smallest distance to it (its nearest neighbor). If you imagine the subsequences being points in a m dimensional Space, then the matrix profile index of a subsequence is the starting index of the closest neighbor and the matrix profile value is the distance to the closest neighbor.

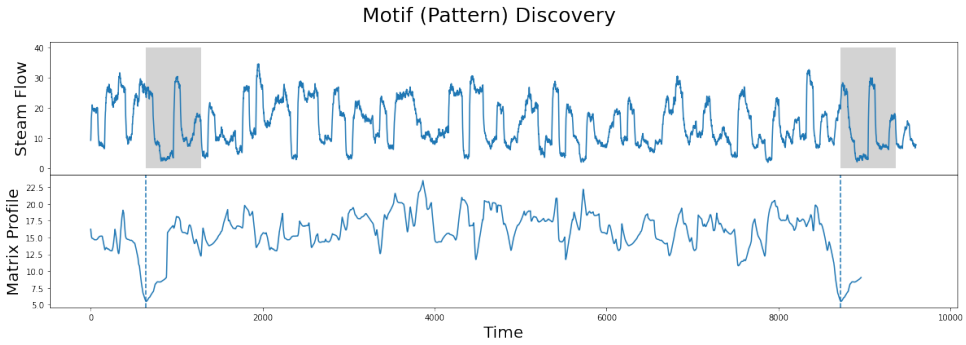


Figure 1: Example of the matrix profile on the steamgen dataset.

So, what does this tell us? In the example above you can see a visualization of the Matrix Profile for the Steamgen Dataset. In the upper plot you see the data set, and in the lower plot you see a plot of the Matrix Profile. The two dashed lines represent the lowest values in the Matrix Profile. This means the two subsequences that start at each of these lines have a very small (the smallest) distance from each other, hence they are very similar. This means, that by visually inspecting the Matrix Profile we can immediately see the most repeated pattern. There are various other things to discover and better overview can be found on the UCR page [9]. There exist also generalizations to more dimensions, see [23].

There exists a very good Python library called *stumpy* for computing the Matrix Profile that also has very good support by the author. The Github page can be found here [14].

How could the MP be used?

As described above, the MP profile is a tool that can help to discover repeating or anomalous patterns in time series data. Hence we can aim to apply it to any of the many time series produced by the source.

1. Patterns in the BCT currents for prediction: One could try to discover repeating patterns in the BCT currents and see if they can be used to predict the future development. For example if a pattern indicates a degradation of the current in the near future, it could be used to alert the operators in time. There exists also a real time version of the MP, where it gets updated with every arriving data point. For this the SDTS algorithm [22] built on top of the MP could be interesting.
2. Pattern in the BCT current for analysis: Likewise, one could try to link patterns in the BCT with patterns/actions of other parameters. For example, often times a slow increase of the HT current leads to a slow degradation of the BCT current.
3. Motifs of different parameter combinations shifted in time: When computing the multidimensional matrix profile, to see if there are motifs in more than on dimension, one could shift one of the time series in time, to see for example how a change of the gas voltage affects the current in one hour.
4. Meta time series: Instead of looking at the original time series, one could first calculate a new series with the rolling mean or standard deviation. I.e. one would slide a window of fixed length over the original series, and store the means/standard deviations of the window in a new sequence. Then one could look for repeated patterns there and try to understand what influences the motifs there.

Difficulties

The MP is built under the assumption, that repeated patterns or motifs are an effect of a regular event in the generating process. One example from the Papers is Seismology. There the time series is the recording of a Seismograph, which can have very long periods of "random" data, where nothing happens. However, an earthquake would show up with a very distinctive shape.

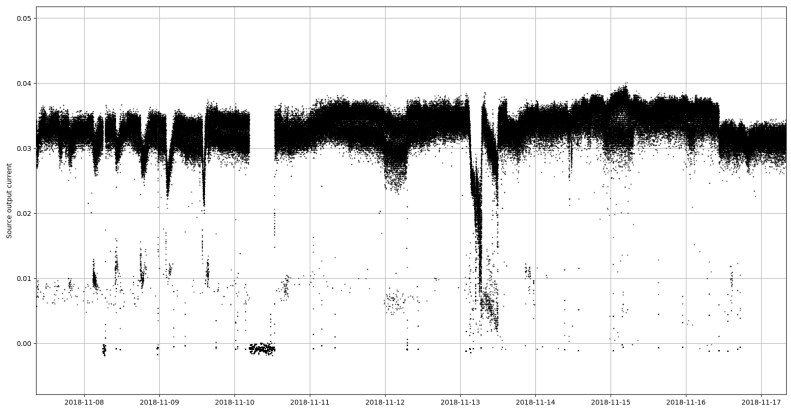


Figure 2: Example of the BCT25 current

In our case however, the BCT signals are mostly flat with oscillations (see Figures 2 and 3), hence motif discovery with z-normalized subsequences is very insufficient. If you have two flat signals with a lot of added noise, their z-normalized euclidean distance will be very large, even if one might expect it to be small as they are visually similar. There are some ideas to mitigate the problem in [16], but I didn't get any satisfying results.

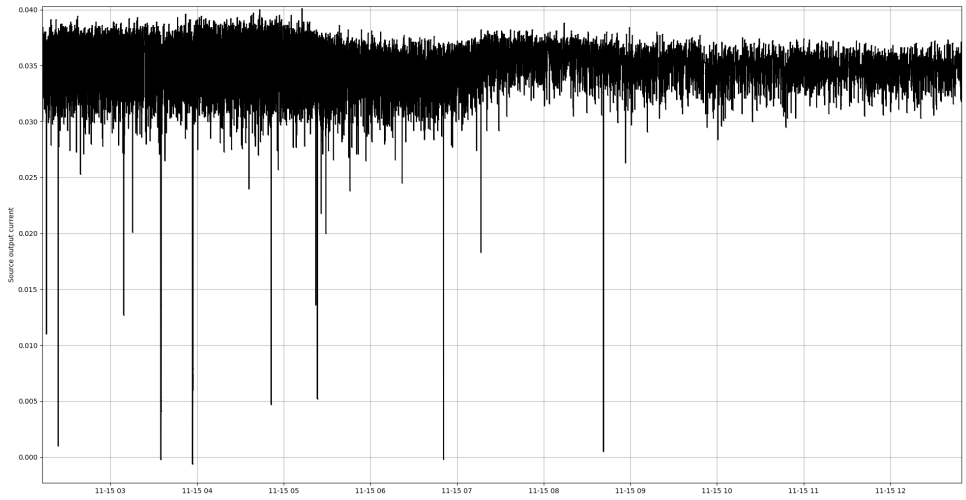


Figure 3: Example of the BCT25 current, smaller window

Results

Instead of trying to use a matrix profile with removed noise I flattened the signal over some minutes and found several links of parameter changes to BCT current. The results where achieved using the MSTOMP algorithm [23], a multi dimensional generalisation of the matrix profile calculation.

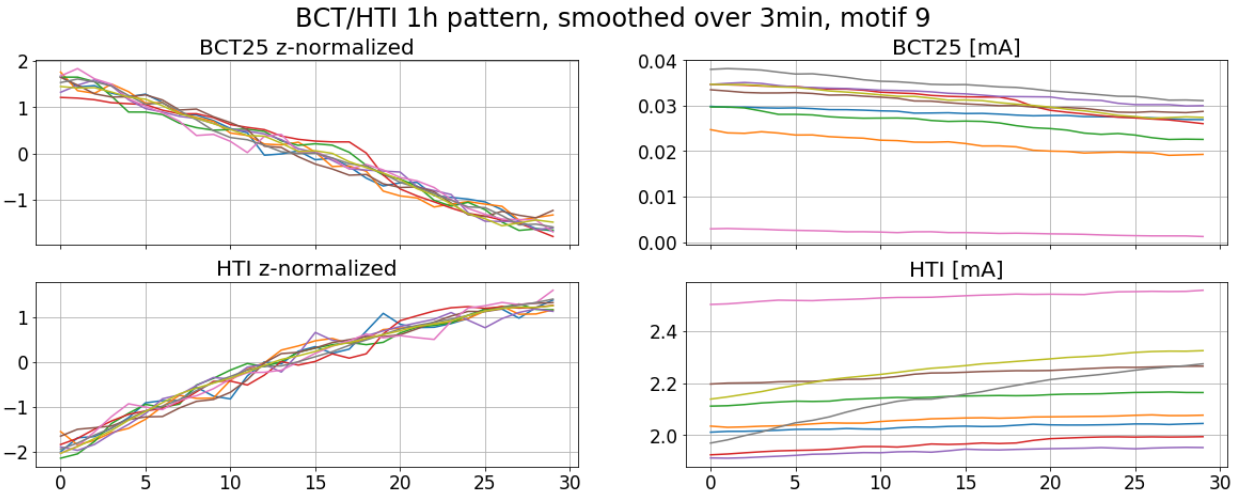


Figure 4: Rise of HTI (bottom) and degradation of BCT25 current (top).

A rising HTI often coincides with a degrading BCT25 current, see Figure 4, so this seems to confirm this theory. However it is not a proof that an opposite effect (e.g. rising HTI and rising BCT current) doesn't also exist. But we didn't observe it in the time frame we considered (August and September 2016).

The reverse can also be seen in Figure 5. Drops of the HTI correlate with jumps in the BCT25 current.

One possible explanation we explored is that the behavior in Figure 5 can often time be achieved by increasing the Oven Power. This can be seen in Figure 6.

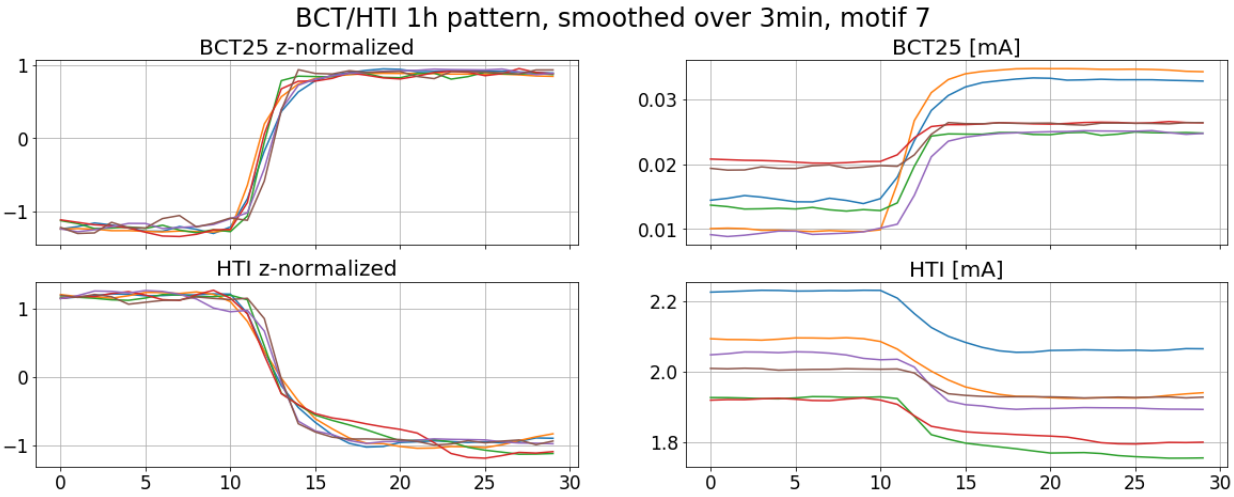


Figure 5: Drops of the HTI and jumps in the BCT25 current.

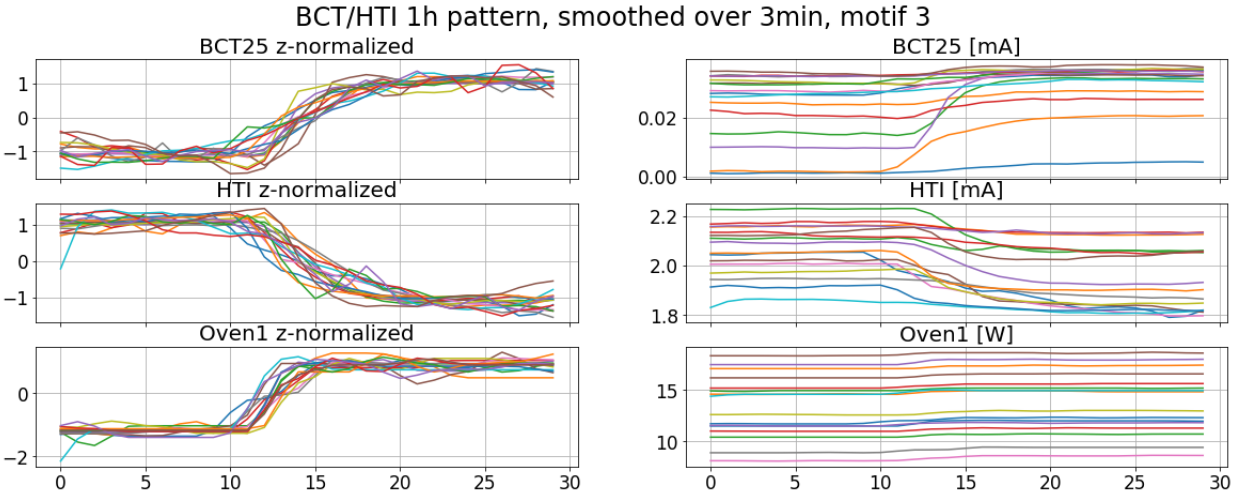


Figure 6: A common pattern when including BCT25 current, HTI and Oven1 power.

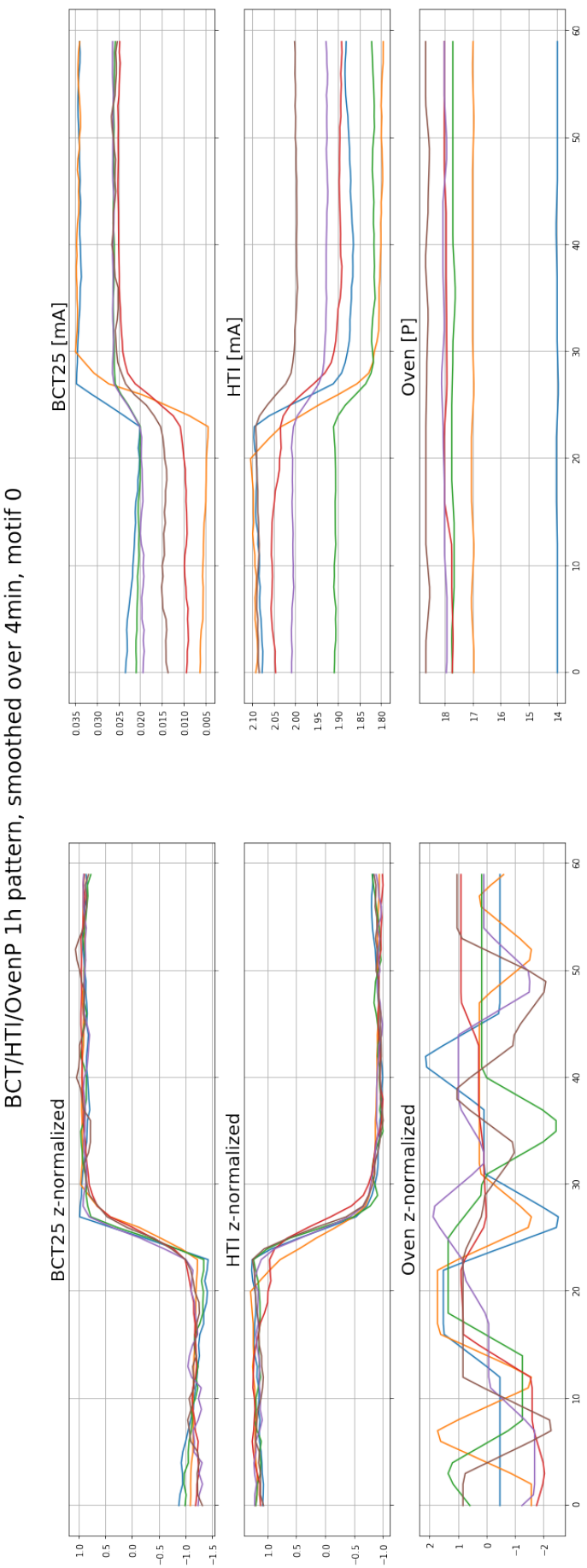


Figure 7: Jumps in BCT25 current and HTI without changes of the Oven power.

However, it appears that there are cases where the oven remains unchanged, cases can be seen in Figure 7. Further invastigation is necessary, but it could be that the power of Oven 2 was increased, as this is only the data for Oven 1.

We could also see that an increase of the oven power is often accompanied with a decrease of the gas voltage, see Figure 8.

We didn't see any meaningful motifs when jointly looking at the Bias Disc Voltage and the BCT25 current.

Discretization

One problem of using the data from CALS/NXCALS is that for all setting only acquisitions are logged. This means, that we typically don't know the exact values a setting was changed to, see Figure 9. It shows one day (05.11.2018) of Oven1 power acquisition from NXCALS, where only one data point every five minutes is logged. From the logbook we can learn when the oven1 power was chaged precisely, however on the plot in Figure 9 we can see some oscillations (The times in the plot are UTC, so you have to count +01h when comparing to the elogbook excerpt).

The same occurs with other settings, and raises the problem that we cannot directly say when a change of a certain setting happened. So I tried to discretize the raw acquisition values and get back the true setting where possible. The main assumption I had to make is that a setting remained constant over time, unless somebody changes it. This appears reasonable in most cases, but in some extreme cases some information might be lost (see below in Figure 10).

Under this assumption we can model a setting as a step function [20] with added noise, and the problem is to find the step function. I will call the step function *discretization*, because we separate our time series into discrete states of a fixed setting.

There are several techniques that could be used to solve such a problem, and we will discuss some of them below in more detail for a different use case. For this use case I combined a simple rolling window approach with a decision tree regressor. Some results can be seen below.

As one can see, the discrete approximation, our attempt at finding the true step function, follows the acquisition signal very closely and most changes are modeled correctly. This can also be seen when comparing the results with entries in the elogbook (especially for the oven, since here most changes are noted in the logbook). As can be seen in the figures, during some periods no discrete approximation is plotted. This is the case when the source was off (BCT05 current 0A), because the method does not work well when there are sections with a non-step function like signal as during an oven restart, so I cut them out.

Explanation

As described above, the process involves using a Decision Tree Regressor (see [17]). A decision tree partitions the input data by sequentially applying if-then-else rules. It can

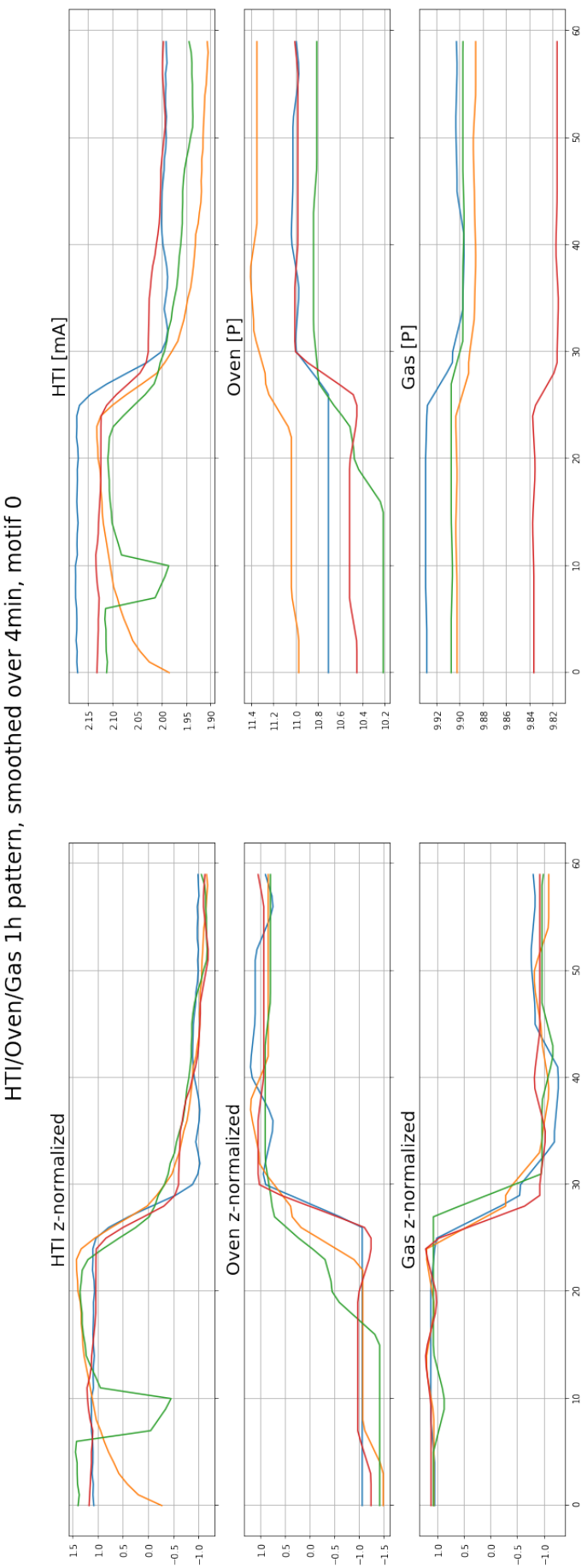
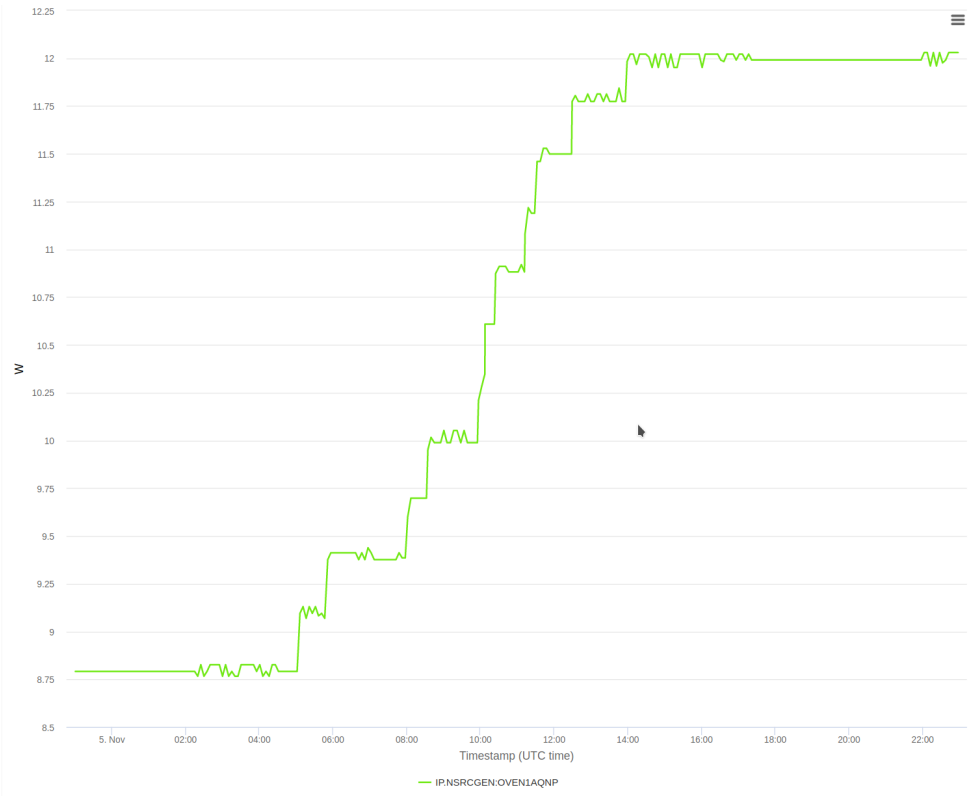


Figure 8: Correlation of gas voltage decreases with oven power increases.



05/11/2018 06:05	DAY LINAC 3
Oven to 9.1W [rs]	

05/11/2018 06:50	DAY LINAC 3
Tuning I don't find anything better. Increase oven as suggested earlier -> 9.4W. [rs]	

05/11/2018 09:01	DAY LINAC 3
oven1 9.7W [dk]	

05/11/2018 10:56	DAY LINAC 3
oven1 10.3W [dk]	

05/11/2018 11:07	DAY LINAC 3
oven1 10.6W [dk]	

05/11/2018 11:24	DAY LINAC 3
ooven1 10.9W [dk]	

Figure 9: Top: Screenshot of Oven1 Acquisition, Bottom: Excerpt from the elogbook on this day. Times are +01:00 in comparison to UTC on the plot.

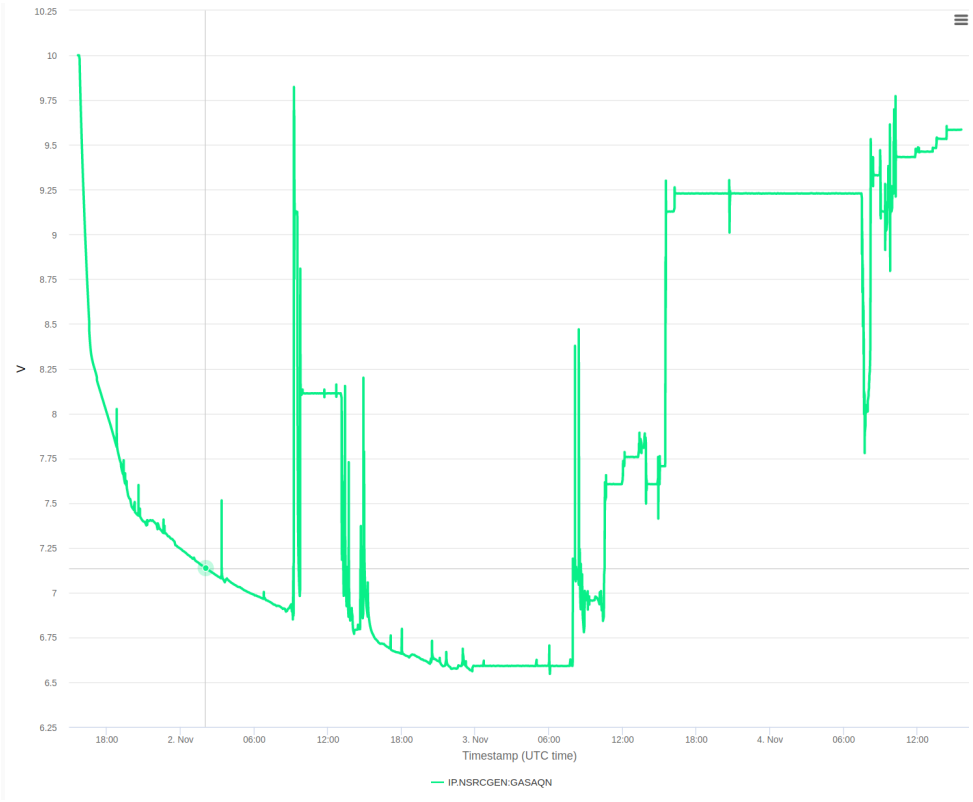


Figure 10: GASAQN voltage on the second and third November 2018. This was during an oven refill where the gas pressure changes not as a step function.

be thought of as a directed graph, were every node is one of these rules. End nodes, so called leaves, that return the class the input is belonging to. Training a decision tree means finding an optimal set of rules, that explains the training data as good as possible. Decision trees are a supervised learning method, meaning that for each training input a output class is specified, and the algorithm tries to learn this relationship. Decision trees can be used for regression. If you want to regress a function $\mathbb{R}^n \to \mathbb{R}; \quad (x_1, \ldots, x_n) \mapsto y$ you pass $(x_1, \ldots, x_n)$ as input and y as the desired class. In the case of an one dimensional function for example, a decision tree classifier could learn that for $x >= 5$ and $x <= 10$ it should output $y = 5$. So, by the nature of a decision tree, the regression result is a step function, that looks like the result as much as possible.

One very common problem with decision trees is over fitting. If they are allowed to grow too much, they are not regressing any more, but copying. For example suppose that all your input data points are the natural numbers. By building a tree with the rules $(x >= 0.5, x < 1.5)$, $(x >= 1.5, x < 2.5)$, $(x >= 2.5, x < 3.5)$, … the resulting regressor could perfectly replicate the input function, but it would learn all small oscillations, what is not what we are typically interested in. However, one thing that can be controlled is how many leaf nodes the tree can have, i.e. in the 1D case into how many intervals the real axis can be split at most.

In our case it would hence be useful to know how many discrete levels, or number of constant segments, the function we want to model consist of. Then, we could regress it using a tree with this maximum of leaf node, because by minimizing the error (i.e. difference from the original function) it would find the best stepwise approximation to the input data without over-fitting.

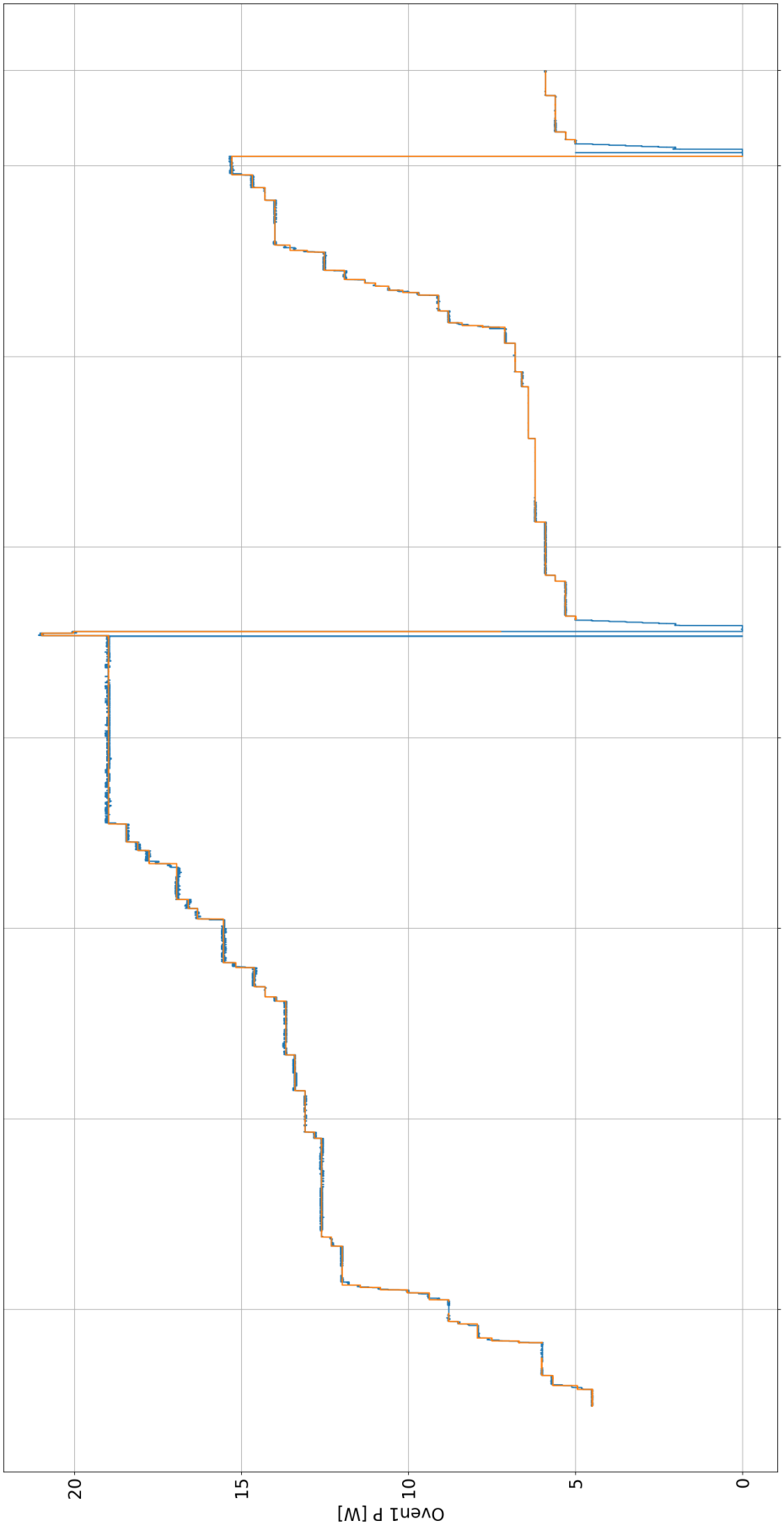


Figure 11: Discretization of Oven Power for November 2018. In blue the original data is plotted, the orange line represents the discrete approximation.

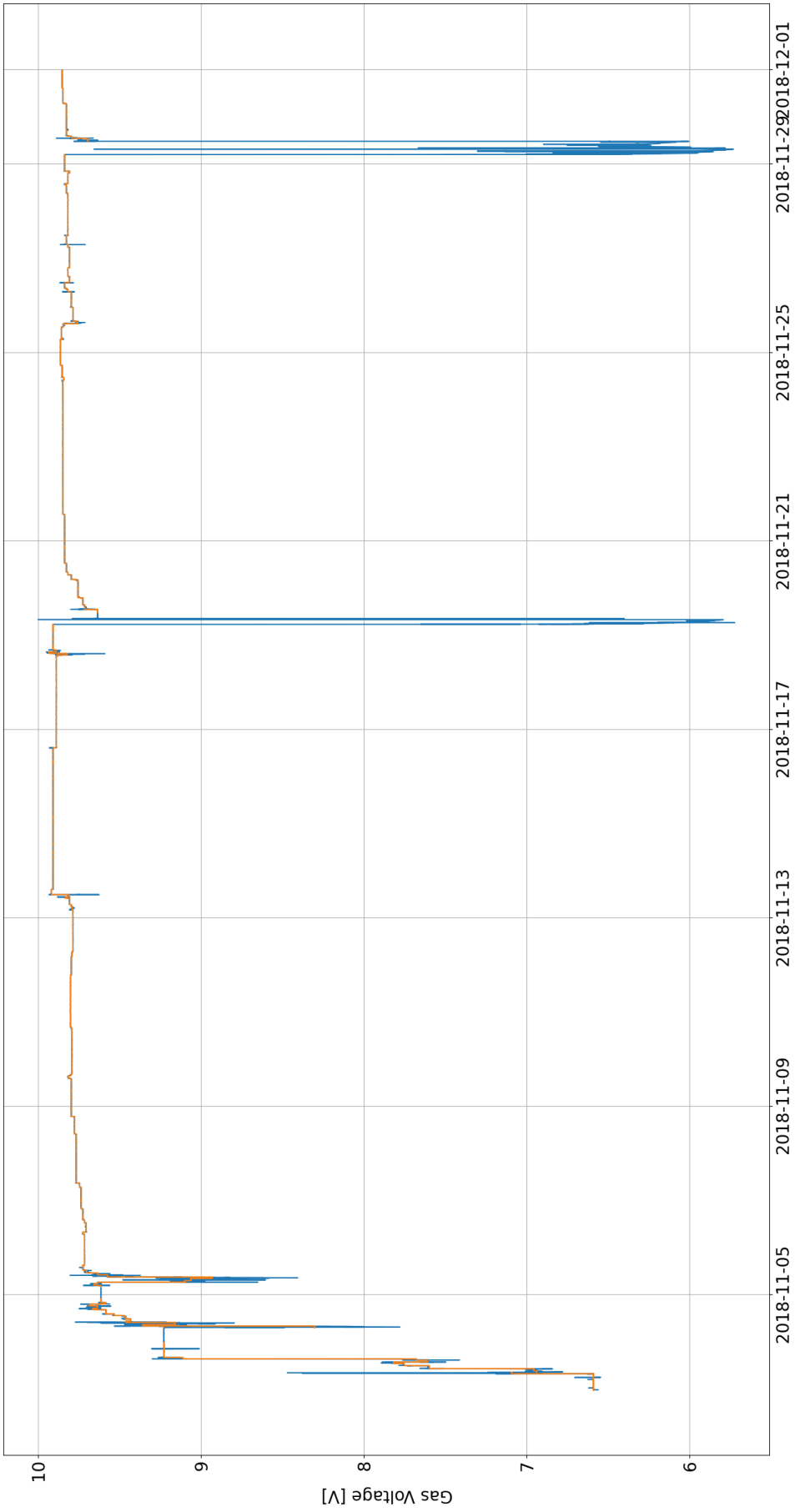


Figure 12: Discretization of Gas Voltage for November 2018.

Finding the number of constant segments

This leads to the topic on change point detection which I will describe more in detail in the next chapter. However, if we are only interested in the number of step jumps, we can do it a bit easier. We can use two adjacent windows in which we compute the means and then calculate the quotient. If it is close to one, the means before and after the touching point of the windows are the same, if the quotient is not one, then somewhere around the touching point a jump has happened. In particular the jump happens very close to local extrema. Hence we only need to count the number of local extrema to get a very good approximation of the number of jumps that happened. However, it should be tried to do this analysis only in regions where the data actually has a step function character.

Change Point Detection

Changepoints are abrupt changes in time series data that can for example be an indicator of some change in the underlying generating process. Finding changepoint can be useful in a lot of cases when it is interesting to calculate exact times when the nature of the data changed. On Linac3 example of a changepoint could be a sudden drop in the beam intensity. This could be an indicator of a change of some source parameter, and detecting such a changepoint early could be used to alert the operators.

There is a variety of changepoint detection Literature available, one overview can be found in [2]. Generally, one distinguishes between offline (or batch) and online analysis. In the offline setting, all data is available a priory, while in the online setting the algorithm decides if a changepoint occurred every time a new data point arrives.

Typically there are two approaches to changepoint detection: Either, one specifies some cost function that is to be minimized. This could be for example the L2-Distance when the number of changepoints is known an changes in mean should be detected (i.e. signal is a step function possibly with some noise and one wants to see when there was a jump). If the number of changepoints is not known, one needs to introduce a penalty that somehow depends on the number of changepoints, to avoid overfitting. For different cost functions in the offline setting and what kind of changes they capture see [19]. The second approach is to view a changepoint as a point, where some parameters of a probability distribution (or the distribution itself) change. Then one would do a hypothesis test with the H_0 hypothesis that the distributions before and after a given point x are equal. If the H_0 hypothesis is rejected, then there is a changepoint at x. A method which builds on this probabilistic setting is detailed below.

Bayesian Online Change Point Detection

Bayesian Statistics

Bayesian statistics is a branch of statistics which makes use of Bayes' theorem

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}.$$

In conventional statistics, one has n observations of a random variable $x = (x_1, ... x_n)$ coming from some probability distribution with unknown parameter θ (which can be multidimensional, e.g. $\theta = (\mu, \sigma^2)$ for the normal distribution). This parameter is treated as an unknown, but fixed variable. In Bayesian statistics however, the parameter is treated as a random variable as well, so we can give meaning to $P(\theta \mid x)$. It is the probability of θ being the parameter of the distribution of the x_i, given all our observations. Hence, we have by Bayes theorem

$$P(\theta \mid x) = \frac{P(x \mid \theta) P(\theta)}{P(x)} \propto P(x \mid \theta) P(\theta) \tag{1}$$

$P(x \mid \theta)$ is called the likelihood, and it encodes our model of the reality, in how we think that x depend on θ. For the (stable) beam current we could for example assume that it is normally distributed with some parameters μ, σ were we estimate these values by looking at data from the last hour. $P(\theta)$ is called the prior distribution. It contains the knowledge we have about our parameter θ, by telling us, how likely a parameter is. For example an average beam current in the order of 1A is very unlikely, whereas an average current around 100mA is much more likely). Finally, $P(\theta \mid x)$ is called the posterior distribution. It can be seen as an update of our observations on θ with new knowledge coming from x. For a more thorough introduction see [18, 3].

Algorithm

Based on this idea, Adams and MacKay proposed an online changepoint detection algorithm [1]. A great introduction can be found at [7] and a python implementation at [13] They introduce the concept of *run lengths*. At a given point in time t the run length r_t gives the number of time steps that passed since the last changepoint happened. During one run, we assume that all observations come from the same distribution.

If the run length is known, at $t + 1$ only two things are possible: $r_{t+1} = r_t + 1$ (no changepoint happened) or $r_{t+1} = 0$ (there was a changepoint). The algorithm computes at each step t the run length distribution $P(r_t \mid x_1, ..., x_t)$, i.e. the probability of having a run length r_t having observed everything that happened up to time t, see figure 13.

By definition of the conditional probability we have ($x_1, ..., x_t \equiv x_{1:t}$)

$$P(r_t \mid x_{1:t}) = \frac{P(r_t, x_{1:t})}{P(x_{1:t})}$$

and the nominator can be computed recursively (see equation (3) in [1]) as

$$P(r_t, x_{1:t}) = \sum_{r_{t-1}} P(r_t \mid r_{t-1}) P(x_t \mid r_{t-1}, x_t^{(r)}) P(r_{t-1}, x_{1:t-1}) \tag{2}$$

while the denominator can be seen as a scaling factor. $x_t^{(r)}$ denotes the set of observations associated with the run r_t (The notation in the paper is slightly confusing. To be precise $x_t^{(r)} \equiv (x_{t-r_t}, ..., x_{t-1})$, so the most recent observation is explicitly not included!). To apply the algorithm it helps to understand some technicalities, which I try to explain

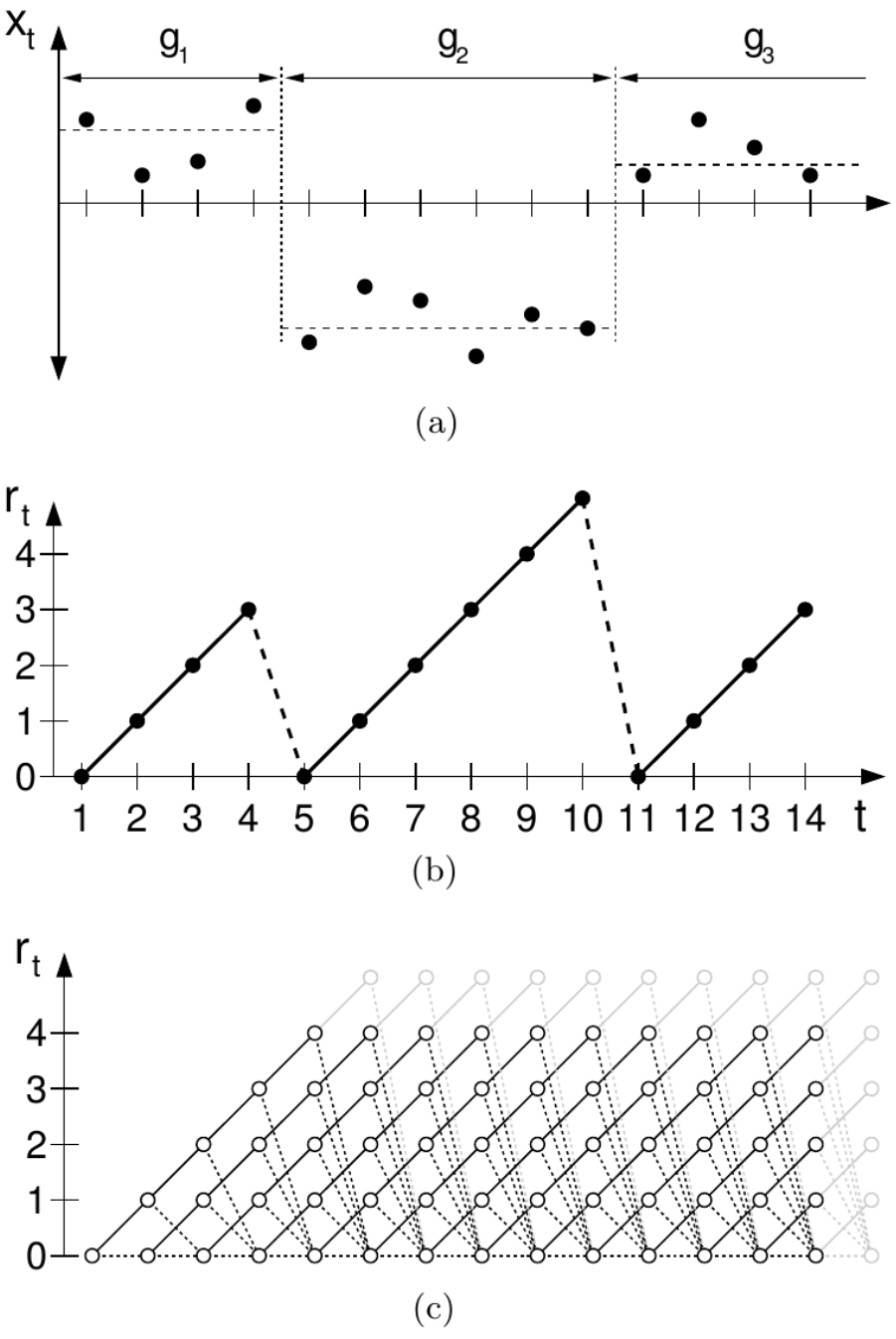


Figure 13: Explanation of the run length, figure taken from [1]. In (a) you can see a time series with two changepoints at $t = 5$ and $t = 11$. In (b) the true run length can be seen. Notice that when a changepoint occurs, the run length drops to zero. (c) shows an excerpt of possible run lengths and the edges of the graph show how run lengths can evolve with increasing time. Note that at any point, the run length for the next time point can be only either one greater or zero. The algorithm introduced in [1] computes the probability of each node in the graph.

below. However I recommend reading a little bit (for example on wikipedia) about Hazard functions and more importantly conjugate priors.

Hazard function

The first term in the sum, $P(r_t \mid r_{t-1})$, is called the changepoint prior and it encodes the information or intuition we have about the distribution of the changepoints. At any given time t, we can have only $r_t = r_{t-1} + 1$ or $r_t = 0$. We introduce a hazard function H that encodes the probability that a changepoint happened at a given run length. Then one can write

$$P(r_t \mid r_{t-1}) = \begin{cases} H(r_{t-1}+1) & \text{if } r_t = 0, \\ 1 - H(r_{t-1}+1) & \text{if } r_t = r_{t-1}+1 \text{ (no changpoint)}, \\ 0, & \text{else} \end{cases}$$

For the ion source it probably makes sense to model the changepoints distribution as an exponential distribution, meaning that at each time step there is a change of probability p for a new regime to start. However, one could improve this by making the hazard function dependent for example on a rising on the HT current, as a rising current seams to make drops in intensity more likely. But for the easy case without any assumption, only a sensible value for p needs to be determined. As for the exponential distribution we have $\mu = \frac{1}{p}$, one can choose p as one over the average time between changepoints, e.g. 4 hours. The algorithm doesn't seem to be to be very sensible to the exact value of p, as long as it is chosen reasonably. In this exponential case, the hazard function is just $H(y) \equiv p$, so it is constant.

Conjugate Priors

The first term in the sum of equation (2), $P(x_t \mid r_{t-1}, x_t^{(r)})$, gives the probability of observing an new value x_t, given that we have run length r_{t-1} and that for this run length we observed the values $x_t^{(r)}$. To calculate this, we need to decide how our data is distributed during one run.

In case of the beam current, it makes sense to assume it is normally distributed with unknown mean μ and variance σ^2. Using the first few points of the run, we could estimate these two parameters, call them $\bar{\mu}$ and $\bar{\sigma}^2$, and then say that for the whole run we have $x_t \mid r_{t-1}, x_t^{(r)} \sim Normal(\bar{\mu}, \bar{\sigma}^2)$. This would be one very likely Normal distribution the data could come from, but as we only estimated the parameters, there is some uncertainty involved. Hence it would make sense to use a weighted average distribution of all possible parameters.

$$P(x_t \mid r_{t-1}, x_t^{(r)}) = \int_\theta P(x_t \mid \theta) P(\theta \mid r_{t-1}, x_t^{(r)}) d\theta \tag{3}$$

This integral can be very difficult to calculate and can be typically only been simulated numerically. But, for some combinations of likelihoods and priors, the result has an

algebraic representation making random sampling unnecessary. If this is the case, one calls the prior a *conjugate prior*. There can be found lists (eg. [6]) of such pairs and how the parameters need to be updated. It is important to still have in mind that the likelihood and prior should be chosen to reflect reality.

Here is a short example where the likelihood (the first term in the integral in (3)) is a normal with unknown mean μ and variance σ^2, as this is used in many implementations of the Algorithm and makes sense for the ion source data. For a more complete derivation see [15]. The conjugate prior in this case is the *Normal-Inverse-Gamma Distribution*.

This distribution has four parameters that we will explain here. Assume that we used $\nu = 100$ data points to calculate a mean of $\hat{\mu} = 10$ and $2\alpha = 100$ (so $\alpha = 50$) observations to calculate a variance of $\hat{\sigma}^2 = 0.1$. We can thus calculate the sum of squares

$$2\hat{\beta} := \sum_{i=1}^{100}(x_i - \hat{\mu})^2 = 2\alpha * \hat{\sigma}^2 = 10 \Rightarrow \hat{\beta} = 5.$$

As μ and σ^2 are unknown, it makes sense to encode our uncertainty using distributions. We want the pair (μ, σ^2) to be distributed with the Normal-Inverse-Gamma distribution, so the following has to hold

$$\sigma^2 \mid \alpha, \beta \sim InverseGamma(\alpha, \beta)$$
$$\mu \mid \sigma^2, \hat{\mu}, \nu \sim Normal(\hat{\mu}, \frac{\sigma^2}{\nu}).$$

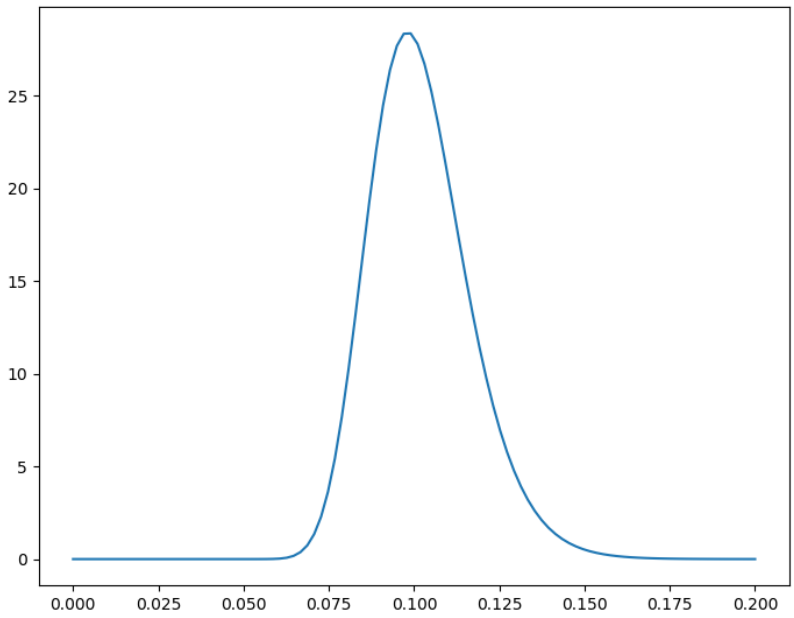


Figure 14: Probability density function of an Inverse Gamma distribution with parameters $\alpha = 50$ and $\beta = 5$. This can be seen as an uncertainty (likelihood) of a estimated variance of 0.1 using 100 samples.

If one looks at the probability distribution of the Inverse Gamma with these parameters, one can see that it is zero except for a bump around our estimated variance $\widehat{\sigma}^2$, so it is a reasonable choice to encode our uncertainty about our estimate (see Figure 14). Similarly, the distribution of μ as a normal centered around $\hat{\mu}$ shows our uncertainty about this estimate, and if the variance of our measurements increase, this uncertainty increases, while the more observations (ν) we use to estimate the mean, our uncertainty decreases. So this choice of prior seems to have a reasonable real world interpretation and we can hence use it in our calculations.

In this case (for derivation see e.g. [15]) we know that

$$x_t \mid r_{t-1}, x_t^{(r)} \sim t_{2\alpha}\left(\hat{\mu}, \frac{\hat{\beta}(\nu+1)}{\alpha\nu}\right), \tag{4}$$

that is a scaled and shifted Student-t distribution with 2α degrees of freedom. Since we know the distribution, we do not need to calculate the Integral in (3) anymore. The fact that the resulting distribution a a Student-t has also a very nice interpretation. The Student-t is a smeared out Gaussian (it has more weight on the tails) which converges to a Gaussian as its degrees of freedom increase. So the higher α is, the closer it resembles our likelihood, which makes sense because we get a better approximation of the variance. Also, as κ increases, the resulting Gaussian (in the limit if α were very large) gets narrower, because with increasing κ there is less uncertainty in the estimation of the mean.

Since the algorithm is a streaming algorithm, we sequentially observe new values. Every time a new value x comes in, we update the parameters using the following rules, which follow directly from what the parameters represent:

$$\begin{aligned}
\nu' &\leftarrow \nu + 1 \\
\mu' &\leftarrow \frac{\nu\mu + x}{\nu + 1} \\
\alpha' &\leftarrow \alpha + \frac{1}{2} \\
\beta' &\leftarrow \beta + \frac{\nu}{\nu+1}\frac{(x-\mu)^2}{2}
\end{aligned}$$

Generalizations

There are various resources that extend the BOCPD algorithm that could be interesting.

In [10] a unification the Algorithm by Adams and MacKay ([1]) with [4] to create a method that allows selecting different models for each segment. Furthermore they present a method to extend BOCPD to multiple dimensions.

In [11] an approach to mitigate the problem of outliers is presented. Bayesian Interference can be seen as a minimization problem over all distributions (see the cited paper) and by changing the metric that is used an alternative definition that is more robust can be used. More motivation on this topic can be found in [12].

The problem of outliers is also tackled in [5], however this is not a Bayesian setting anymore.

One very straight forward improvement is also suggested in the original paper [1], and it is to prune the tail probabilities. This means that you look at the probabilities of the very long runs (the tail of the run length probability distribution) and if they are small, e.g. below 10^{-4} you set them to zero. Hence you can ignore all run lengths greater than a certain threshold in the recursion (2) and thus save computation time.

Applying BOCPD

As the basic version of the algorithm does not require setting any hyper parameters, it can be applied easily. In the setting of the Linac3 Source we decided to model the current as normally distributed variables, which are independent and identically distributed in the segments between two changepoints with unkown mean and variance (i.e. for any segment between two changepoints, a "run", the random variables are independent and identically distributed with unknown mean and variance). We model the time between changepoints a an exponential variable. For the parameter of the Hazard function values in the order of multiple hours produce good results, and in this range the algorithm is not very sensible.

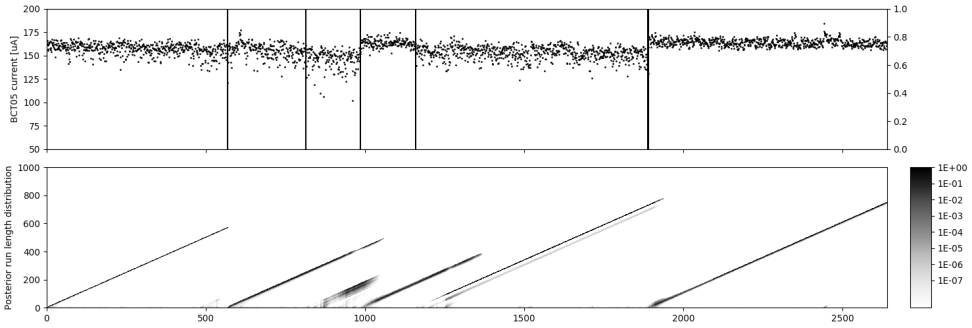


Figure 15: The BOCPD Algorithm applied to around seven hours of BCT05 current from September 2018 (sampled at 10s), where drops due to voltage breakdowns were removed. The black lines in the upper plot show the determined changepoints. In the upper plot the posterior run length distribution is plotted on a logarithmic color scale. Darker pixels indicate a higher probability.

The paper [1] does not propose a strategy to transform the run length probabilities into changepoint locations. There are several ways how this could be done.

One method is to set a delay of d time steps. After processing a new data point, one compares the probability of being in a run with length d (i.e. the value r_d) to a threshold, and if the probability is high enough it is likely, that a changepoint occurred d time steps ago.

Another method is based on the Maximum-a-Posteriori estimate (MAP) and is used for example in [10]. If at time step t the most probable run length is for example 100, and at $t+1$ the most probable run length is 101, then no changepoint occurred. However, if the most probable run length at $t+1$ is 20, then there probably was a changepoint at time $(t+1)-20$.

A third method, which in my opinion has the advantage of being more intuitive, is to at each time step look at the probability again. For example you sum up the probabilities

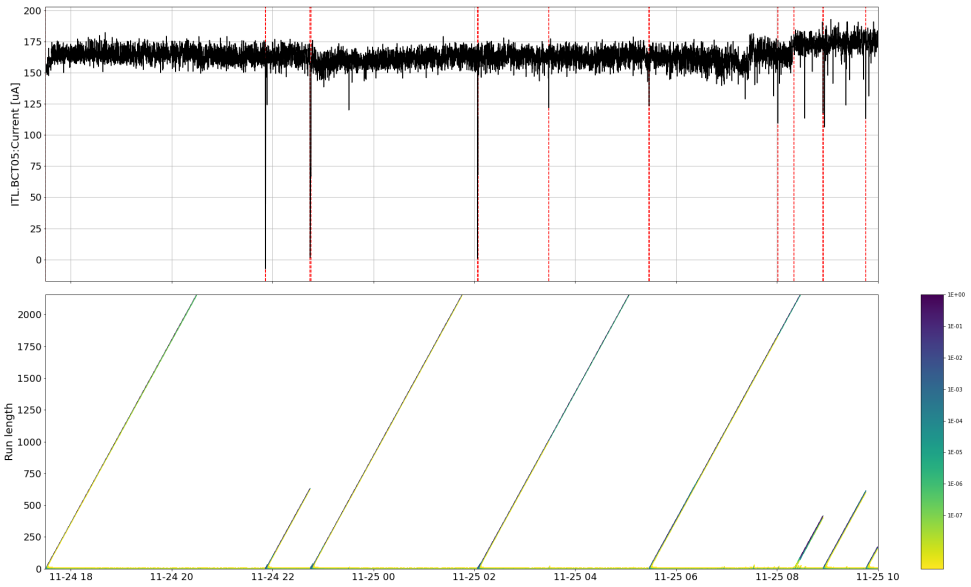


Figure 16: The BOCPD Algorithm applied to a few hours from November 2018 without removing voltage breakdowns, sampled at ten seconds. Here, the run length posterior is color coded differently to show some nuances. Yellow indicates a lower probability while purple indicates a higher run length probability.

of the 20% longest possible run length, and if the sum is smaller than a certain threshold, i.e 0.1, you assume a changepoint occurred. This makes sense visually: Consider Figure 15 and look at the bottom plot at $x = 700$. The longest possible run length is around 150 (Here we assumed that we pruned a after the changepoint at $x = 550$ was found, so all probabilities above the black line are zero). There, almost all probability mass is concentrated around the most probable run length, so we are likely still in this old run. However around $x = 900$, the run length probability distribution gets a second peak for run lengths of around 100. The tail probability there is small, so the probability of being in the run that started around $x = 550$ is small. Hence, we should detect a new changepoint at around $x = 800$.

However, there might be better methods of deciding if a changepoint occurred, visually looking at the run length probabilities produces less false positives, so the intuition could maybe somehow be better encoded.

Suffix arrays

One more interesting approach to explore are suffix arrays. A suffix array is a data structure that can be used to efficiently search for substrings in a string or to find the most often occurring substring of a given length in a longer string. An introductory series of videos can be found on youtube [21].

To use this approach, we need to encode the information we have into a string. The idea is to first discretize the setting Acquisitions, i.e. to get the points when a certain setting was changed. Then, assign a letter a to each setting and concatenate the letters fo all changes. For example assume that at 14:00 the oven (o) was increased, at 14:01 the gas (g) decreased and at 14:02 increased again. Then a string representation would be ”ogg”. If

one wants to encode the increase/decrease, one could choose to represent them with plus and minus, yielding "o+g-g+". Also, it might be useful to encode the duration between changes somehow.

Once a string representation of the changes is found, it could be interesting to combine it with changepoint detection. One could take some interval before a changepoint happens and create a string that contains the information of all changes that happened in these intervals, to then look for common substrings. This would be one way to find out if there are common tuning patterns that affect the beam in a way that changes some property (thus leading to a changepoint).

References

[1] Ryan Prescott Adams and David J.V. MacKay. "Bayesian Online Changepoint Detection". In: *arXiv* (2007). URL: https://arxiv.org/pdf/0710.3742.pdf.

[2] Samaneh Aminikhanghahi and Diane J. Cook. "A survey of methods for time series change point detection". In: *Knowledge and Information Systems* 51 (2017). URL: https://link.springer.com/content/pdf/10.1007/s10115-016-0987-z.pdf.

[3] Jonny Brooks-Bartlett. *Probability concepts explained: Bayesian inference for parameter estimation.* Online Article. accessed 02.07.2020. Jan. 2018. URL: https://towardsdatascience.com/probability-concepts-explained-bayesian-inference-for-parameter-estimation-90e8930e5348.

[4] Paul Fearnhead and Zhen Liu. "On-line inference for multiple changepoint problems". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 69.4 (2007), pp. 589–605. DOI: 10.1111/j.1467-9868.2007.00601.x. URL: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9868.2007.00601.x.

[5] Paul Fearnhead and Guillem Rigaill. "Changepoint Detection in the Presence of Outliers". In: *Journal of the American Statistical Association* 114.525 (2019), pp. 169–183. DOI: 10.1080/01621459.2017.1385466. URL: https://doi.org/10.1080/01621459.2017.1385466.

[6] Daniel Fink. *A Compendium of Conjugate Priors*. 1997. URL: https://pdfs.semanticscholar.org/0894/42f59e3f4afb920479a7115c3dc57fb14757.pdf.

[7] Gregory Gundersen. *Bayesian Online Changepoint Detection*. Aug. 2019. URL: http://gregorygundersen.com/blog/2019/08/13/bocd/.

[8] A. Hinneburg and D. A. Keim. "Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering". In: *Proceedings of the 25th International Conference on Very Large Databases* (1999), pp. 506–517. URL: https://kops.uni-konstanz.de/bitstream/handle/123456789/5790/vldb99.pdf.

[9] Eamonn Keogh. *The UCR Matrix Profile Page*. URL: https://www.cs.ucr.edu/~eamonn/MatrixProfile.html.

[10] Jeremias Knoblauch and Theodoros Damoulas. "Spatio-temporal Bayesian On-line Changepoint Detection with Model Selection". In: *International Conference on Machine Learning (ICML)* (2018). URL: https://arxiv.org/abs/1805.05383.

[11] Jeremias Knoblauch, Jack Jewson, and Theodoros Damoulas. "Doubly Robust Bayesian Inference for Non-Stationary Streaming Data with -Divergences". In: *Neural Information Processing Systems (NeurIPS)* (2018). URL: https://arxiv.org/pdf/1806.02261.pdf.

[12] Jeremias Knoblauch, Jack Jewson, and Theodoros Damoulas. *Generalized Variational Inference: Three arguments for deriving new Posteriors*. 2019. arXiv: 1904.02063 [stat.ML]. URL: https://arxiv.org/pdf/1904.02063.pdf.

[13] Johannes Kulick. *Bayesian Changepoint Detection Code*. Github. URL: https://github.com/hildensia/bayesian_changepoint_detection.

[14] Sean M. Law. "STUMPY: A Powerful and Scalable Python Library for Time Series Data Mining". In: *The Journal of Open Source Software* 4.39 (2019), p. 1504. URL: https://github.com/TDAmeritrade/stumpy/.

[15] Kevin P. Murphy. *Conjugate Bayesian analysis of the Gaussian distribution*. Oct. 2007. URL: https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf.

[16] Dieter De Paepe, Olivier Janssens, and Sofie Van Hoecke. "Eliminating Noise in the Matrix Profile". In: *ICPRAM*. 2019. URL: https://biblio.ugent.be/publication/8605188/file/8605190.pdf.

[17] scikit-learn. *Decision Trees*. Documentation. URL: https://scikit-learn.org/stable/modules/tree.html#tree.

[18] D. Spiegelhalter and K. Rice. "Bayesian statistics". In: *Scholarpedia* 4.8 (2009). revision #185711, p. 5230. DOI: 10.4249/scholarpedia.5230.

[19] Charles Truong, Laurent Oudre, and Nicolas Vayatis. "Selective review of offline change point detection methods". In: *Signal Processing, 167:107299* (2020). URL: https://arxiv.org/pdf/1801.00718.

[20] Eric W Weisstein. *Step Function*. From MathWorld–A Wolfram Web Resource. URL: https://mathworld.wolfram.com/StepFunction.html.

[21] WilliamFiset. *Suffix Array Playlist*. URL: https://www.youtube.com/watch?v=zqKlL3ZpTqs&list=PLDV1Zeh2NRsCQ_Educ7GCNs3mvzpXhHW5.

[22] Chin-Chia Michael Yeh, Nickolas Kavantzas, and Eamonn Keogh. "Matrix Profile IV: Using Weakly Labeled Time Series to Predict Outcomes". In: *VLDB 2017* (2017). URL: https://www.cs.ucr.edu/~eamonn/WeaklyLabeledTimeSeries.pdf.

[23] Chin-Chia Michael Yeh, Nickolas Kavantzas, and Eamonn Keogh. "Matrix Profile VI: Meaningful Multidimensional Motif Discovery". In: *ICDM 2017* (2017). URL: http://www.cs.ucr.edu/%7Eeamonn/Motif_Discovery_ICDM.pdf.

[24] Chin-Chia Michael Yeh et al. "Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets". In: *IEEE ICDM 2016* (2016). URL: https://www.cs.ucr.edu/~eamonn/PID4481997_extend_Matrix%20Profile_I.pdf.