NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSc THESIS

Event Correlation and Forecasting over High Dimensional Streaming Sensor Data

Thomais V. Vasilopoulou
Sofia A. Kostakonti

Supervisor: **Stathes Hadjiefthymiades,** Professor

ATHENS

JUNE 2020

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Συσχέτιση και Πρόβλεψη Συμβάντων σε Πολυδιάστατα Δεδομένα Αισθητήρων

Θωμαΐς Β. Βασιλοπούλου
Σοφία Α. Κωστακόντη

Επιβλέπων: **Ευστάθιος Χατζηευθυμιάδης,** Καθηγητής

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2020

BSc THESIS

Event Correlation and Forecasting over High Dimensional Streaming Sensor Data

Thomais V. Vasilopoulou
S.N.: 1115201500016

Sofia A. Kostakonti
S.N.: 1115201500080

SUPERVISOR: **Stathes Hadjiefthymiades,** Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Συσχέτιση και Πρόβλεψη Συμβάντων σε Πολυδιάστατα Δεδομένα Αισθητήρων

Θωμαΐς Β. Βασιλοπούλου
Α.Μ.: 1115201500016

Σοφία Α. Κωστακόντη
Α.Μ.: 1115201500080

ΕΠΙΒΛΕΠΩΝ: **Ευστάθιος Χατζηευθυμιάδης,** Καθηγητής

ABSTRACT

As technology advances, the need to detect and predict events in real-time or near real-time intensifies. In this paper, we will discuss, what constitutes an event for every data stream and how using different algorithms, future events may be predicted. These predictions are feasible, due to the correlation between corresponding events and become more frequent as the spectrum of previous events taken into account increases. Moreover, in real-world applications, these events remain relevant as time progresses with diminishing probability all the while, which is something that the algorithms we developed take into account. Due to managing Big Data, a Python implementation was considered the best approach, since both Pandas and NumPy libraries provide ease of use and optimal run time for such problems. In order to present as realistic results as possible, a variety of variables were differentiated so as to extract the outcome with the best precision and recall.

SUBJECT AREA: Sensor Networks

KEYWORDS: event correlation, event forecasting, change detection

ΠΕΡΙΛΗΨΗ

Με την πρόοδο της τεχνολογίας, εντείνεται η ανάγκη ανίχνευσης και πρόβλεψης συμβάντων σε πραγματικό χρόνο, ή σχεδόν πραγματικό χρόνο. Στην πτυχιακή αυτή, αναλύουμε τι θεωρείται συμβάν για τις επιμέρους ροές δεδομένων και τον τρόπο με τον οποίο καθίσταται δυνατή η επιτυχής πρόβλεψη των επόμενων συμβάντων, μέσω της χρήσης ειδικά κατασκευασμένων αλγορίθμων. Οι προβλέψεις αυτές είναι εφικτές, εξαιτίας των μεταξύ τους συσχετίσεων. Επιπλέον, η ακρίβεια των προβλέψεων κορυφώνεται, όσο συμπεριλαμβάνεται ένα ευρύτερο φάσμα από παλαιότερα συμβάντα-καταστάσεις. Στον πραγματικό κόσμο, οι καταστάσεις αυτές διατηρούν μία φθίνουσα χρονικά πιθανότητα πραγματοποίησης, κάτι που οι αλγόριθμοι που υλοποιήσαμε λαμβάνουν υπόψιν. Η διαχείριση Big Data μας ώθησε στη χρήση της γλώσσας προγραμματισμού Python, σε συνδυασμό με τις βιβλιοθήκες NumPy και Pandas, προκειμένου να επιτευχθεί βέλτιστος χρόνος εκτέλεσης. Με στόχο την παρουσίαση πιο ρεαλιστικών αποτελεσμάτων, διαφοροποιήθηκε ένα πλήθος μεταβλητών του προγράμματος ώστε να επιλεχθούν οι τιμές που έχουν τη βέλτιστη ακρίβεια και ικανότητα ανάκλησης.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Δίκτυα Αισθητήρων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: συσχέτιση συμβάντων, πρόβλεψη συμβάντων, ανίχνευση αλλαγής

Στη συνάδελφό μου, για τις ατελείωτες ώρες που με ανέχτηκε.

ΕΥΧΑΡΙΣΤΙΕΣ

Για τη διεκπεραίωση της παρούσας Πτυχιακής Εργασίας, θα θέλαμε να ευχαριστήσουμε τον επιβλέπων καθ. Ευστάθιο Χατζηευθυμιάδη, για την εξαιρετική συνεργασία. Θερμές ευχαριστίες και στον υποψήφιο διδάκτορα Βασίλη Παπαταξιάρχη, μέλος της ερευνητικής ομάδας Διάχυτου Υπολογισμού (Pervasive Computing Research Group), για τη καθοδήγηση, την συνεχή ενθάρρυνση και την πολύτιμη συμβολή του στην ολοκλήρωση της.

AKNOWLEDGMENTS

We would like to express our utmost gratitude to our thesis advisor Prof. Stathes Hadjiefthymiades for the excellent cooperation. Our sincere thanks also go to Ph.D. candidate Vassilis Papataxiarhis, member of the Pervasive Computing Research Group, for the guidance, continuous encouragement, and valuable contribution during the project’s fulfillment.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Inspired by the state-of-the-art technological advances, the impending need for real-time responsive devices has been increasing the demand for event prediction, depending on event succession of past recordings. That was incentive enough for the implementation of 2 (two) algorithms that function accordingly. These algorithms, process event dataset streams and use them to calculate probabilities of next step events.

It is crucial to comprehend that this approach does not apply to every dataset one may acquire. Specifically, we focused on applications where the set of past events has predictive power over the following event succession. Our implementation, for example, was tested using streams of data from a naval environment, provided by our supervisors.

An effectively fathomable model would utilize a relatively primitive dataset presenting 5 (five) data streams: Humidity, Temperature, Light, Smoke, and Fire. If on a specific moment, the first four events occur simultaneously -humidity drops, temperature rises, light increases, smoke increases- then a well-executed and trained algorithm could predict that in a time span of a few moments, it is highly possible that the event Fire will occur. The details of said implementation, along with the detection of occurring events, are examined further on.

An event constitutes an abrupt change of behavior of a specific stream, on a specific time step. It becomes apparent that an intermediate “translator” of stream values is necessary to create the event vectors used in the example above. That is accomplished by executing a set of different algorithms to convert data streams into event vectors.

Real-time data processing is the execution of data in a short period, providing near-instantaneous output. Stream processing is a technology that allows users to query continuous data streams and detect conditions quickly, near instantaneously, after receiving the data. Both concepts adhere to our implementation and depict its importance while allowing predictions for larger systems and variable sets than already existent.

The purpose of this thesis is to contribute to the vast spectrum of Sensor Network Data Processing by acknowledging and adapting to patterns that occur in real-world applications. Through this process, we wish to contribute to this sector by creating algorithms that automate significant processes and achieve more precise results.

2. EVENT MANAGEMENT OVER MULTIVARIATE NUMERICAL DATA

When a prediction of a future event is crucial, the first step to understanding the way the data streams function as real-world conditions is to detect abrupt changes in their respective functionalities. To make our implementation as efficient as possible, we used already existent algorithms that have proven to be lucrative in time and space complexity, namely the Cumulative Sum (CUSUM) and Shewhart controllers.

Once a clear understanding of event occurrences is established, it becomes apparent that these occurrences represent correlations in real-world applications and, therefore, should be approached accordingly.

The understanding that their sequence stems from their in-between succession is what makes the development of an algorithm for their correlation plausible.

We developed two algorithms that differ on the way they correlated events, and we evaluated them based on their precision and recall percentages. These algorithms are Stepwise Correlation and Sliding Window algorithms.

2.1 Event Detection

2.1.1 CUSUM Algorithm

The first algorithm we implemented to detect events between time series is CUSUM or Cumulative Sum algorithm [1]. CUSUM evaluates the change between the data values collected and a target value. It calculates both positive and negative changes and, over time, adds them up until they reach a certain threshold, positive and negative, respectively. If the sum exceeds the threshold, an event is detected, and the cumulative sums reset. For slow changes to remain unreported as events, CUSUM also proposes a positive and a negative tolerance parameter. Below, we provide the variables as well as the algorithm for CUSUM. As input variables, we consider those mentioned above:

- μ, the target value
- k^+, the positive tolerance value
- k^-, the negative tolerance value
- $thres^+$, the positive threshold value
- $thres^-$, the negative threshold value

As output variables, we have the positive and negative detection of an event; occurrence signaled by 1 and absence by 0:

- s^+, the positive detection signal
- s^-, the negative detection signal

ALGORITHM 1: CUSUM Algorithm

Input: univariate time series x_t , target value μ , above-tolerance k^+ , below-tolerance k^-, above-threshold $thres^+$, below-threshold $thres^-$
Output: above detection signal s^+ , below detection signal s^-

```
1.  $P \leftarrow 0$ ;
2.  $N \leftarrow 0$ ;
3.  $t \leftarrow 1$ ;
4.  $\boldsymbol{while}$ ( $true$ )
5.      $s^+ \leftarrow 0$;
6.      $s^- \leftarrow 0$;
7.      $P \leftarrow \mathbf{max}\,(0, x_t - (\mu + k^+) + P)$ ;
8.      $N \leftarrow \mathbf{min}(0, x_t - (\mu - k^-) + N)$ ;
9.      $\boldsymbol{if}$ ( $P > thres^+$ ) $\boldsymbol{then}$
10.         $s^+ \leftarrow 1$;
11.         $P \leftarrow 0$ ;
12.         $N \leftarrow 0$ ;
13.     $\boldsymbol{end}$
14.     $\boldsymbol{if}$ ( $N < -thres^-$ ) $\boldsymbol{then}$
15.         $s^- \leftarrow 1$;
16.         $P \leftarrow 0$ ;
17.         $N \leftarrow 0$ ;
18.     $\boldsymbol{end}$
19.     $t \leftarrow t + 1$ ;
20. $\boldsymbol{end}$
```

This algorithm considers a given that our data follow a normal distribution, therefore all the parameters must be set according to the specific dataset and what is considered a change for a unique stream. In case of multivariate event streams [1], such as ours, CUSUM must be applied separately for each variable [2].

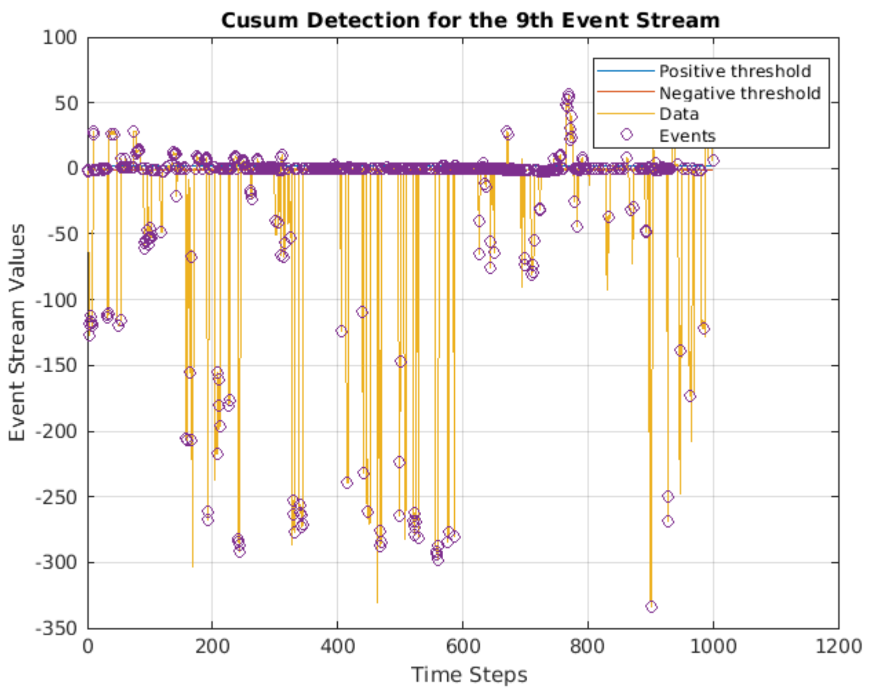


Figure 1: Event detection using CUSUM

As presented in Figure 1, the thresholds that CUSUM selects are very close to 0. Due to this, the algorithm detects changes of incoming data even for very slight fluctuations, and in turn proceeds to consider almost every stream as an event.

2.1.2 Shewhart Algorithm

Shewhart Controller algorithm consists of a recurring comparison between a function's mean value and two envelopes, labeled UCL (Upper Control Limit) and LCL (Lower Control Level), representing the statistical process's $x_t \in \mathbb{R}$ distance from said mean value.

In order to understand what aggregates a change, we define a parameter *k*, which represents the limit to x_t's fluctuation. Additionally, the standard deviation in every time step is symbolized as σ.

The detection of any event is based on a trigger that is activated every time the value of x_t fluctuates outside the two bounds, UCL and LCL. We consider these fluctuations as events, since what is considered "normal" behavior for the function is exceeded during the current time step.

To explain in mathematical terms, we define UCL and LCL as follows:

$$\begin{aligned} UCL &= \overline{x_t} + k \cdot \sigma_t \\ LCL &= \overline{x_t} - k \cdot \sigma_t \end{aligned} \tag{1}$$

where $\overline{x_t}$ represents the function's mean value in the current time step.

To implement the aforementioned algorithm with Python, the desideratum is to translate a dataset into a vector of 0's and 1's. Those values represent whether, on that particular time step, an event occurred or not. Loading the selected dataset as a Pandas' dataframe and converting it to a NumPy array, we can now compare the incoming values, draw the function and decide whether the current stream constitutes an event or not.

For every incoming stream in the given dataset, we evaluate the mean value of the function and calculate the standard deviation and using a fixed k (event assessment threshold) to find the asymptotes. A simple conditional statement on whether UCL or LCL are exceeded is enough to decide whether the current stream constitutes an event.

The algorithm described above is:

ALGORITHM 2: Shewhart Algorithm
Input: univariate series x_t, tightness k
Output: detections signal s

```
1.  $\overline{x_0} \leftarrow 0$ ;
2.  $\sigma_0 \leftarrow 0$ ;
3.  $t \leftarrow 1$ ;
4.  $\boldsymbol{while}$ ( $true$ )
5.          $\overline{x_t} \leftarrow \overline{x_{t-1}} + \frac{\overline{x_t - x_{t-1}}}{t}$;
6.          $\sigma_t \leftarrow \sqrt{\frac{1}{t}((t-1)\sigma_{t-1}^2 + (x_t - \overline{x_t}) \cdot (x_t - \overline{x_{t-1}}))}$;
```

```
7.        $UCL \leftarrow \overline{x_t} + k \cdot \sigma_t$ ;
8.        $LCL \leftarrow \overline{x_t} - k \cdot \sigma_t$ ;
9.        if $((x_t > UCL)$ or $(x_t < LCL))$ then
10.           $s \leftarrow 1$;
11.       else
12.           $s \leftarrow 0$;
13.       end
14.       $t \leftarrow t + 1$ ;
15.   end
```

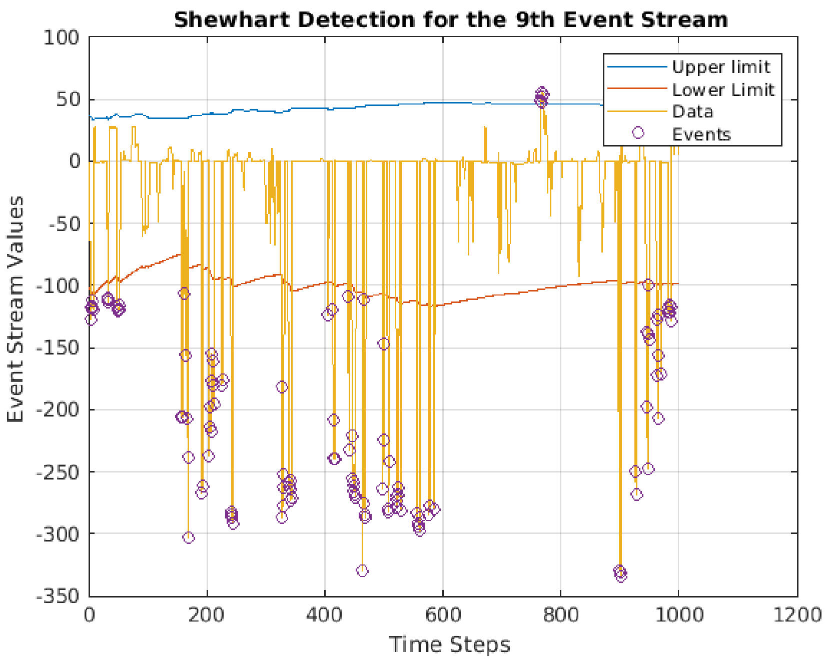


Figure 2: Event Detection using Shewhart

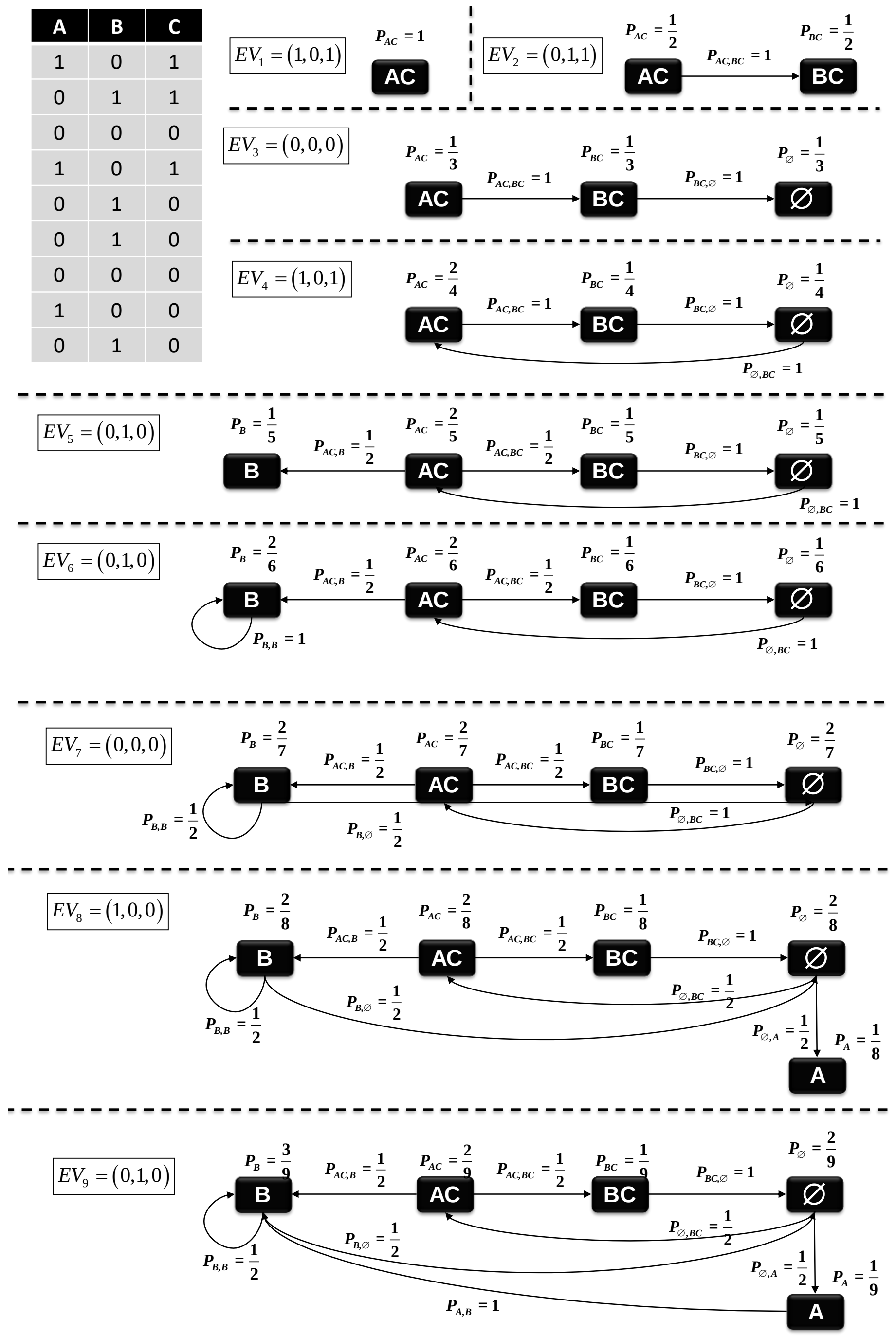
This controller [2] for Shewhart is less populated than CUSUM’s respective one. Interestingly, upper and lower limits are now more distinguishable, allowing for more realistic representation of occurring events, since now slight fluctuations are considered normal under real circumstances rather than detecting some new similar event.

2.2 Event Correlation

2.2.1 Stepwise Algorithm

The basis of the Stepwise Correlation Algorithm [3] lies in the creation of a directed graph of the occurring events. In case the event detected in this current time step has already been added to the graph, then with a set probability, we can predict what the next occurring event will be.

To facilitate the algorithm’s functionality, let’s present an example [3]; suppose we have 3 streams of events A, B, C.



A	B	C
1	0	1
0	1	1
0	0	0
1	0	1
0	1	0
0	1	0
0	0	0
1	0	0
0	1	0

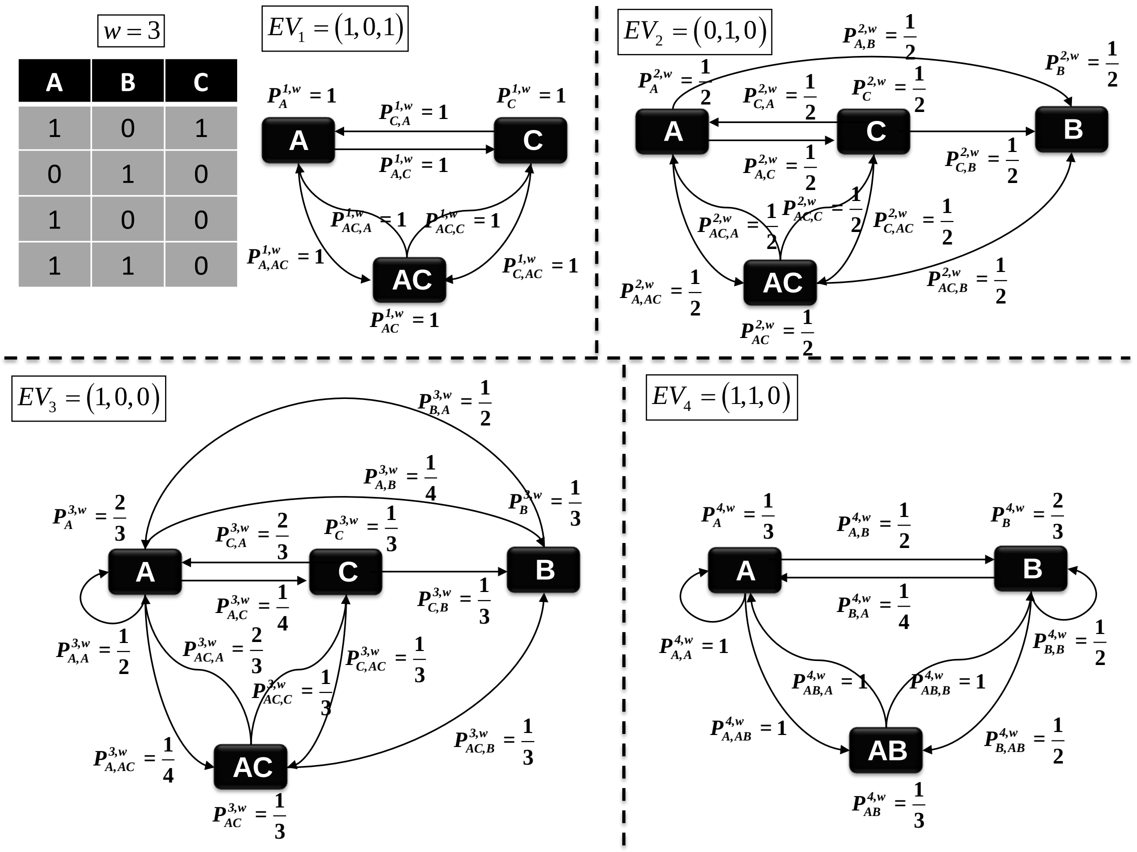
Figure 3: Example of stepwise correlation algorithm for multivariate event data

The Python routine we implemented imports a dataset as an event vector and using the first few data to train our model. Training, in this case, means that we do not make predictions but append these events to create a directed graph. This step could be omitted, but with minimal loss, we increase the precision and recall by a significant percentage.

Once the training period is over, our model is ready to start making predictions. After detecting the next event, an examination could indicate that perhaps it is already located inside the graph. Thus, we have already encountered it during the training phase and are now capable of making a prediction.

2.2.2 Sliding-window Algorithm

Although the Stepwise Correlation algorithm connects events that take place consecutively, it disregards all the cases where an event B takes place several time steps after event A, despite it having been triggered by A. In order to be able to correlate those types of connections, we introduce a time window of size *w*. At each step, all probabilities within this “sliding” window are recalculated, considering the newly reported event vector. In the Sliding Window algorithm [3], we consider each event vector stream to be related to any of the others or none at all. We present a set that contains every occurring event inside the window. Therefore, for every event vector stream, we calculate its powerset and a set of conditional probabilities within the given window. For a large number of event streams, these calculations can become quite complex. To avoid that, we can consider the combinations of up to K simultaneous event streams at most.



$w = 3$

A	B	C
1	0	1
0	1	0
1	0	0
1	1	0

$EV_1 = (1,0,1)$
$P_A^{1,w} = 1$
$P_C^{1,w} = 1$
$P_{C,A}^{1,w} = 1$
$P_{A,C}^{1,w} = 1$
$P_{AC,A}^{1,w} = 1$
$P_{AC,C}^{1,w} = 1$
$P_{A,AC}^{1,w} = 1$
$P_{C,AC}^{1,w} = 1$
$P_{AC}^{1,w} = 1$
A
C
AC
$EV_2 = (0,1,0)$
$P_{A,B}^{2,w} = \frac{1}{2}$
$P_B^{2,w} = \frac{1}{2}$
$P_A^{2,w} = \frac{1}{2}$
$P_{C,A}^{2,w} = \frac{1}{2}$
$P_C^{2,w} = \frac{1}{2}$
$P_{A,C}^{2,w} = \frac{1}{2}$
$P_{C,B}^{2,w} = \frac{1}{2}$
$P_{AC,A}^{2,w} = \frac{1}{2}$
$P_{AC,C}^{2,w} = \frac{1}{2}$
$P_{C,AC}^{2,w} = \frac{1}{2}$
$P_{A,AC}^{2,w} = \frac{1}{2}$
$P_{AC,B}^{2,w} = \frac{1}{2}$
$P_{AC}^{2,w} = \frac{1}{2}$
A
C
B
AC
$EV_3 = (1,0,0)$
$P_{B,A}^{3,w} = \frac{1}{2}$
$P_{A,B}^{3,w} = \frac{1}{4}$
$P_B^{3,w} = \frac{1}{3}$
$P_A^{3,w} = \frac{2}{3}$
$P_{C,A}^{3,w} = \frac{2}{3}$
$P_C^{3,w} = \frac{1}{3}$
$P_{A,C}^{3,w} = \frac{1}{4}$
$P_{C,B}^{3,w} = \frac{1}{3}$
$P_{A,A}^{3,w} = \frac{1}{2}$
$P_{AC,A}^{3,w} = \frac{2}{3}$
$P_{C,AC}^{3,w} = \frac{1}{3}$
$P_{AC,C}^{3,w} = \frac{1}{3}$
$P_{A,AC}^{3,w} = \frac{1}{4}$
$P_{AC,B}^{3,w} = \frac{1}{3}$
$P_{AC}^{3,w} = \frac{1}{3}$
A
C
B
AC
$EV_4 = (1,1,0)$
$P_A^{4,w} = \frac{1}{3}$
$P_{A,B}^{4,w} = \frac{1}{2}$
$P_B^{4,w} = \frac{2}{3}$
$P_{B,A}^{4,w} = \frac{1}{4}$
$P_{A,A}^{4,w} = 1$
$P_{B,B}^{4,w} = \frac{1}{2}$
$P_{AB,A}^{4,w} = 1$
$P_{AB,B}^{4,w} = 1$
$P_{A,AB}^{4,w} = 1$
$P_{B,AB}^{4,w} = \frac{1}{2}$
$P_{AB}^{4,w} = \frac{1}{3}$
A
B
AB

Figure 4: Sliding Window Correlation algorithm example [3], where w=3

3. IMPLEMENTATION DETAILS

3.1 General

For the implementation of the algorithms outlined above, we decided to use Python, given that the program would need to be able to handle large datasets, as well as that, from all other options (Matlab, Java), we were most comfortable working in Python. To work with the dataset, we used the NumPy and Pandas libraries, and we also utilized multi-threading to speed up the execution time, which was imperative for the experimentation stage. We based our implementation around a maritime domain dataset that contains around 22.000 logs from 29 different sensor streams. First, both CUSUM and Shewhart algorithms were applied to our data to get the event vectors, while noting which streams contained occurring events at a given time step. These vectors function as input in the implementation of the Stepwise and Sliding Window correlation algorithms, and after a short training period, produce the event prediction for the next time step and its probability. Furthermore, for the experimental stage, we created several scripts that tested the programs with different parameters and, therefore, have created several files. Their format explanation is as follows: shewhartRandomKevents*.data and cusumRandomKevents*.data are different event vector files, stemming from the same two original files shewartEventVector.data and cusumEventVector.data. Those files, each, have K random bits active if there are more than K events detected at any given time step, to reduce time complexity. The results files are all in the folder “results” separated according to algorithms and other parameters.

3.2 Stepwise Correlation

In the Stepwise correlation algorithm, the detection of an event in one data stream is indissolubly connected with the detection, or lack thereof, of an event in all other data streams at this given time, and, therefore, cannot be considered separately. Using the first thousand data vectors as training data, all appearance frequencies of events populate a graph G, implemented as a dictionary of dictionaries. The frequency of an event B following an event A increments each time event B occurs immediately after event A.

Predictions begin right after the training set, and for every step thereafter, each current prediction is compared to the actual event vector and a new prediction is made for the next time step. In case the current event vector is not previously found in G, a possible outcome in the next time step cannot be determined.

Once the selected dataset is parsed and all predictions are accounted for, the precision and recall rates are calculated as such:

$$precision = \frac{exact\ correct\ predictions}{number\ of\ predictions}$$

$$recall = \frac{exact\ correct\ predictions}{dataset\ size - train\ set\ size}$$

The algorithm that we developed and implemented is presented below:

ALGORITHM 3: Stepwise Correlation Algorithm

Input: train event vectors $\boldsymbol{tr_t}$, test event vectors $\boldsymbol{ts_t}$
Output: predictions $\boldsymbol{pred_k}$

```
1.  $graph \leftarrow \{\}$;
2.  $event \leftarrow tr_0$;
3.  $graph[event] \leftarrow \{\}$;
4.  $k \leftarrow 1$;
5.  $\boldsymbol{while}\ (\ true\ )$
6.      $\mathrm{prevState} \leftarrow \mathrm{event}$;
7.      $\mathrm{event} \leftarrow \mathrm{tr}_k$;
8.      $\mathrm{updateGraph}(\ \mathrm{prevSate},\ \mathrm{event},\ \mathrm{graph}\ )$;
9.      $\mathrm{k} \leftarrow \mathrm{k} + 1$;
10. $\mathbf{end}$
11. $\mathrm{t} \leftarrow 0$;
12. $\boldsymbol{if}\ graph[event] \neq 0\ \boldsymbol{then}$
13.     $pred_t \leftarrow \max(\ graph[event]\ )$;
14. $\boldsymbol{else}$
15.     $pred_t \leftarrow \emptyset$;
16. $\boldsymbol{end}$
17. $\boldsymbol{while}\ (\ true\ )$
18.     $prevState \leftarrow event$;
19.     $event \leftarrow ts_t$;
20.     $updateGraph(\ prevState,\ event,\ graph\ )$;
21.     $\boldsymbol{if}\ graph[event] \neq \{\}\ \boldsymbol{then}$
22.         $pred_{t+1} \leftarrow \max(\ graph[event]\ )$;
23.     $\boldsymbol{else}$
24.         $pred_{t+1} \leftarrow \emptyset$;
25.     $\boldsymbol{end}$
26.     $t \leftarrow t + 1$;
27. $\boldsymbol{end}$
```

3.3 Sliding Window

The Sliding Window algorithm treats event streams both as separate and as simultaneous events. To elaborate, from a 3-column dataset A, B, C, an event AC is an event A, an event C, and an event AC all at the same time. In addition to this, it is more realistic to use aging functions to maintain events that are likely to happen in a few time steps with diminishing probability. For that purpose, we implemented two(2) aging functions, a linear, and an exponential one.

The routine uses an event vector to begin implementing the algorithm. Every w steps, where w is the time window inside which we operate, the graph is renewed, and new probabilities are calculated. Thus, the first w steps are considered the training data for this graph. Furthermore, two(2) lists are used, one containing lists of the powersets mentioned above (prevPSets), and one containing every item of the first list autonomously (powerList).

Our next goal is to predict what will happen in the next time step. To do that, we define a combined probability between every event inside powerList and the current event. That probability is computed as such:

$$p = \frac{true\ occurences\ of\ a\ powerList\ event\ given\ the\ current}{ideal\ number\ occurences\ given\ the\ current}$$

Let's illustrate how this probability is calculated with an example. Suppose the dataset used, consists of 3 streams A, B and C. Presented below is the window of size 3:

Table 1: Sliding Window: Probability Calculation Example

	WINDOW		
	A	B	C
i	1	0	1
ii	0	1	0
iii	1	0	1

For the calculations, we add up all occurrences for every given. This means that:

$$p_{B|A} = \frac{1}{3+1} = \frac{1}{4}$$

This becomes even more interesting when we compute $p_{A|A}$:

$$p_{A|A} = \frac{2+1}{3+1} = \frac{3}{4}$$

When calculating that probability, one must realize that even if the event predicted does not occur in the exact next time step it is quite likely to happen in a span of a few moments. Hence an aging function is of importance, while these probabilities are stored for every w.

When an aging function is implemented, an array of predictions, along with their probabilities and their active time, are stored for a specific amount of time steps. For every prediction thereafter, one must decide between the maximum calculated probability from within the time window and those stored, choosing the maximum value in each case.

One should note, that to produce results in a viable way, an implementation based on threads was considered the best approach. For the duration of every prediction, a mutex was acquired and then released to speed up every process.

The algorithm for the Sliding-Window Correlation is presented below:

ALGORITHM 4: Sliding-Window Algorithm

Input: event vectors V_t, window size w
Output: predictions $pred_k$

```
1.  $prevPSets \leftarrow [\,];$
2.  $t \leftarrow 0;$
3.  $\boldsymbol{while}\ (t \leq w-1)$
4.      $event \leftarrow V_t;$
5.      $pSet \leftarrow powerset(event);$
6.      $prevPSets.append(pSet);$
7.      $t \leftarrow t+1;$
8.  $\boldsymbol{end}$
9.  $k \leftarrow 0;$
10. $\boldsymbol{while}\ (true)$
11.     $predictions_k \leftarrow (None, 0);$
12.     $currentPS \leftarrow powerset(V_t);$
13.     $prevPSets.append(currentPS);$
14.     $powerList \leftarrow all\ items\ in\ prevPSets;$
15.     $\boldsymbol{for}\ all\ combinations\ of\ sets\ \boldsymbol{in}\ powerList$
16.         $idealCount \leftarrow all\ possible\ appearences\ of\ setA\ in\ w;$
17.         $itemCount \leftarrow all\ appearences\ of\ setB\ after\ setA\ in\ w;$
18.         $prob \leftarrow itemCount/idealCount;$
19.         $\boldsymbol{if}\ prob > prediction_k[1]\ \boldsymbol{then}$
20.             $predictions_k \leftarrow (setB, prob);$
21.         $\boldsymbol{end}$
22.     $\boldsymbol{end}$
23.     $prevPSets.remove(0);$
24.     $t \leftarrow t+1;$
25.     $k \leftarrow k+1;$
26. $\boldsymbol{end}$
```

4. EXPERIMENTAL EVALUATION

The experimental portion of this thesis consisted of testing the aforementioned code with real world scenarios and more specifically, a set of values extracted from an infrastructure of three (moving) ships, whose data was collected in real-time. This dataset contains 29 sensor streams, measuring 29 different environmental parameters, including ship acceleration, direction, humidity, water inclination, depth levels, etc. All results yielded and analyzed through our implementation were produced using this particular dataset, by differentiating multiple variables.

For both Stepwise Correlation and Sliding Window algorithms, to be able to produce comprehensible output sets, the need arose to set an upper limit to all possible outcomes. Hence, a k variable was defined to represent the varying number of considered events at the same timestep, ranging from $k = 1$ to $k = 7$. Additionally, it was quite interesting to observe the way these two algorithms behaved, when a threshold was set, that disallowed predictions of lesser probability to be accounted for. However, through experimentation, it became obvious that those percentages could skyrocket when the time horizon expanded, considering a prediction accurate even if it occurred up to 3 timesteps forward. For all these cases, one should take into account that both Shewhart and CUSUM were applied so that a larger amount of results could be retrieved.

4.1 Stepwise Correlation Algorithm

The 4 variables that diversified our result sets were:

1. $\boldsymbol{k}$, random simultaneous stream events,
2. $\boldsymbol{p}$, probability,
3. $\boldsymbol{h}$, time horizon,
4. detection algorithm, i.e. $Shewhart, CUSUM$.

To identify whether a prediction was accurate, a general rule for distinction needs to be set. What constitutes a valid prediction is an event predicted on this current timestep occurring within the next h timesteps. Hence, the precision for each case is calculated as such:

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

, where TP represents True Positives (i.e. every valid prediction) and FP False Positives (i.e. every unsatisfied prediction made). It becomes apparent, that the denominator is the sum of predictions made during runtime.

A full algorithm assessment is also influenced by its recall, calculated as:

$$Recall = \frac{TP}{TP + FN}$$

where, again, TP represents True Positives (i.e. every valid prediction) and now FN False Negatives (i.e. every prediction that should have been made but was not). For this case, the denominator is the size of the dataset, as, theoretically, a prediction should exist for every event present in a dataset.

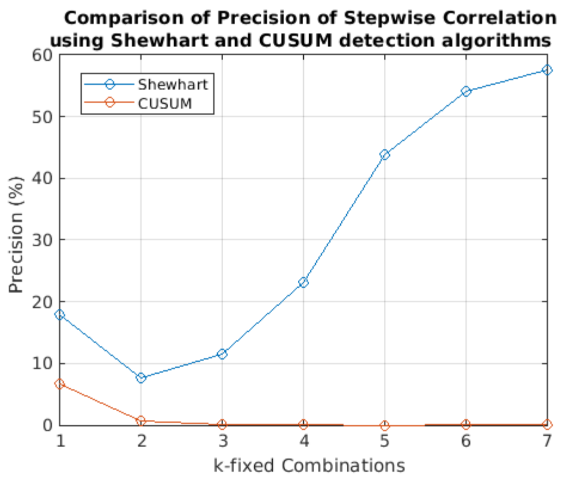
At each step, the prediction recommended by the algorithm was the one with the highest probability value as it was indicated by the event correlation scheme. Based on these results, the precision and recall values were updated accordingly in each step and the average values for each experiment are presented in the figures below.

During implementation, major obstacles concerning the amount of results were encountered. In order to be able to analyze the output data, few event streams in each case were activated. These k random activated bits of event streams ranged from 1 to 7 as mentioned above.

For a variety of different k values, different results were observed. Specifically, both lower and higher values led to better precision and recall, whereas intermediate k's seemed to yield relatively worse outcomes. Regarding the better performing values, a smaller k alludes to isolated events, unrelated to the rest of the data streams, which, due to occurring rarely, conclude to a higher amount of correct predictions, thus increasing both precision and recall. Respectively, a higher k allowed for predictions for a large set of simultaneous events. Those sparse system states have a clear correlation between the monitored variables and therefore, predictions were mostly accurate, resulting in better precision and recall. On the other hand, k values fluctuating in the middle, most of the times appear less successful, since they are neither low enough to appear as individual nor high enough to capture all simultaneously occurring events. This triggers a random selection of which streams are activated in each case, which causes a drop in performance.

A different task would be to observe how these algorithms behaved when restricted to probabilities with a threshold. Specifically, for Stepwise Correlation algorithm, the interest focused on predicting both without a lower limit but also with a probability set to 1.

Finally, the last parameter taken into account was the event detection algorithm. Considering the fact that Shewhart and CUSUM translate abrupt changes in continuous streams of data differently, it is safe to assume that they would produce different results. In most cases, Shewhart performed better due to our data not following a normal distribution. Graphs that present these algorithms' performance are depicted below.



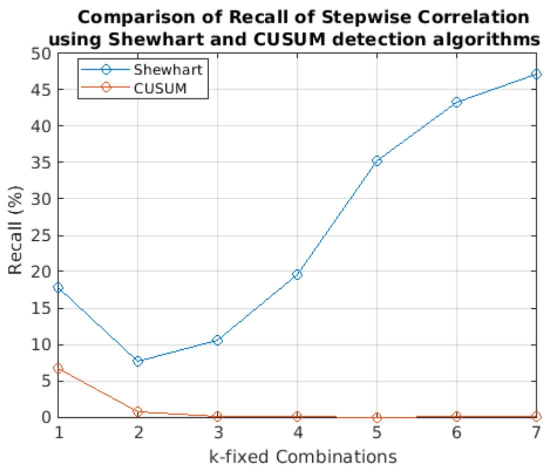


Figure 5:Comparison of Shewhart and CUSUM algorithms on Stepwise Correlation

As mentioned above, even though Shewhart exhibits a drop of middle k values, it still performs better than CUSUM for all results.

An extension of the Stepwise Correlation algorithm for this thesis concerned the lifetime of a prediction – thus changing the meaning of stepwise. It was proven during the experimentation phase that expanding the time horizon inside which a prediction is considered valid without any aging penalty, increases both precision and recall values for CUSUM and Shewhart detection algorithms by an approximate maximum of 5%. This is normal, considering that a prediction has more chances to be added to the True Positives set, which would in turn increase the algorithm’s accuracy.

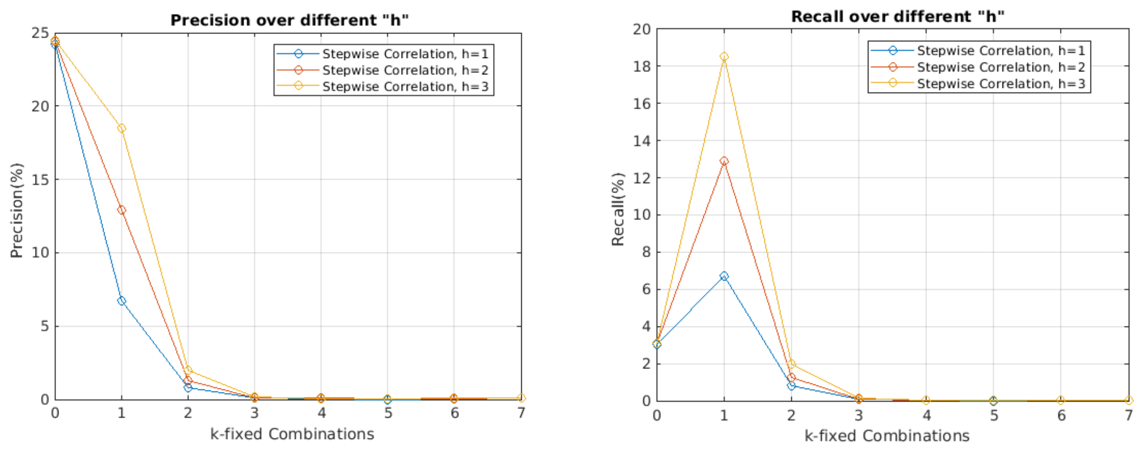


Figure 6: Precision and Recall Percentages on Stepwise Correlation using CUSUM

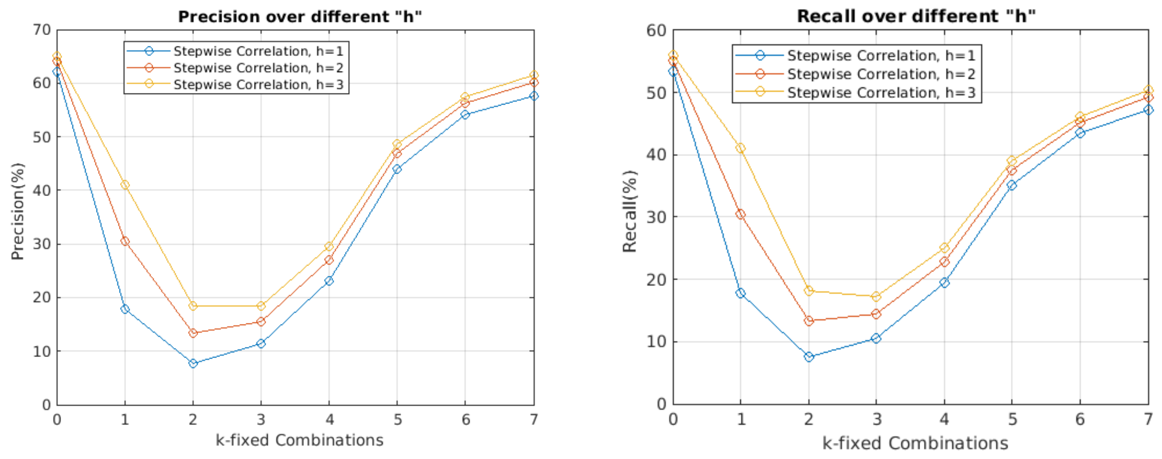


Figure 7: Precision and Recall Percentages on Stepwise Correlation using Shewhart

4.2 Sliding-window Algorithm

The case for Sliding Window algorithm consists of 6 different parameters:

1. $\boldsymbol{k}$, random simultaneous stream events,
2. $\boldsymbol{w}$, time window,
3. $\boldsymbol{p}$, probability,
4. $\boldsymbol{h}$, time horizon,
5. aging function, i.e. $Linear, Exponential$,
6. detection algorithm, i.e. $Shewhart, CUSUM$.

Precision and recall are calculated in the same way as Stepwise. In this case, though, True Positives are determined differently. Provided that change has occurred in m sensor streams in a particular timestep, a prediction is considered precise even if the algorithm only predicts n changes, where $n \subseteq m$. This applies both to precision and recall, differentiating only their respective denominators.

The time window $\boldsymbol{w}$ represents a fixed graph size, one that contains any events that have occurred up to $\boldsymbol{w}$ time steps ago.

Since this implementation follows an approach of retaining probabilities inside the spectrum of a time window, it makes sense to store an array of extra values $\boldsymbol{a}$, representing the highest probability of past events for a specific time period, greater than $\boldsymbol{w}$. This probability decreases over time by an aging function that may be either linear or exponential.

In Sliding Window, it is of greater significance to acknowledge what happens when a lower limit is set on accepted predictions' probability. Therefore, $\boldsymbol{p}$ is differentiated by receiving more, intermediate values, incremented by 0.1 in each loop, starting at 0.4 up to 0.9.

For the remaining variables, each definition applies as in Stepwise Correlation algorithm documentation. To illustrate this further, below is a presentation of how CUSUM and Shewhart algorithms behave for Sliding Window algorithm:

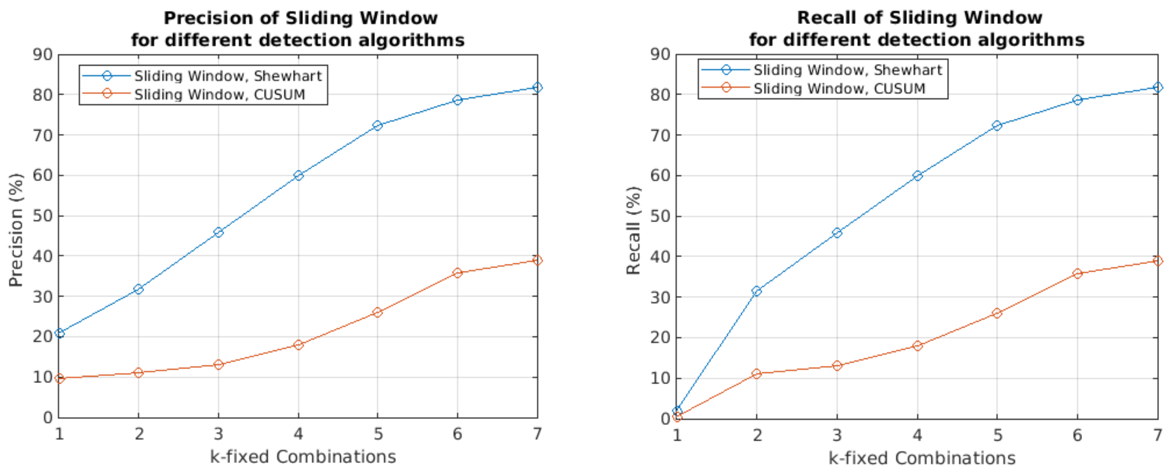


Figure 8: Comparison of Shewhart and CUSUM algorithms on Sliding Window

Again, due to the data not following a normal distribution, Shewhart's algorithm renders better results, in this case almost twice as accurate as CUSUM.

Moving on, testing probability acceptance over a set threshold yields an interesting inference. Precision and recall seem to stabilize for all different probability thresholds, when, for the k randomly selected bits, $k \geq 2 \; or \; k \geq 3$. As mentioned in the Stepwise Correlation's experimentation segment, a higher k alludes to more specific events that represent real occurrences, hence the probability of these actually happening increases and surpasses all monitored thresholds.

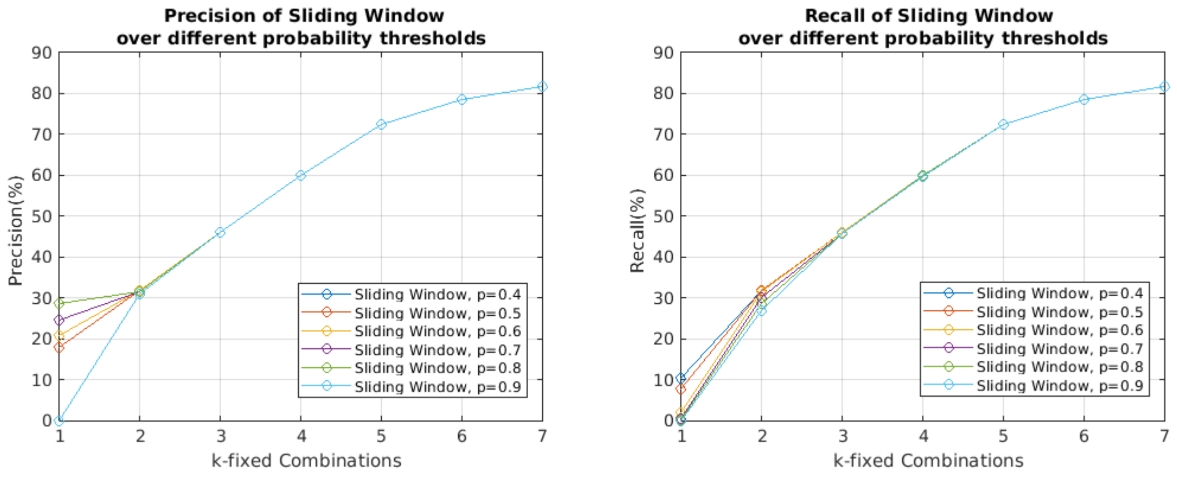


Figure 9: Precision and Recall Percentages over different probability thresholds

As expected, a differentiation of probability thresholds produces the same results (overlapping plots) over a higher k choice.

In the same manner as with Stepwise, a larger horizon of maintaining probabilities increases the algorithm's accuracy. However, for bigger k values, the gap is now much smaller (~3%) and does not constitute a good tradeoff.

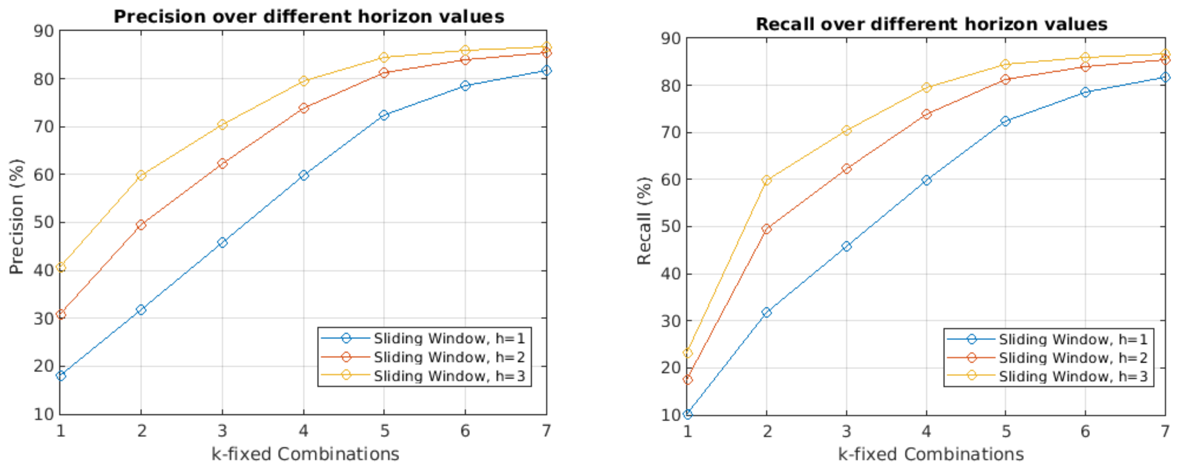


Figure 10: Precision and Recall Percentages over different horizon values

In this case, using an horizon of $h = 3$, for smaller k values, procures an increase in precision and recall of 30%, which would be considered a very smart tradeoff. We conclude that only for lower k values, an extension of the algorithm that would allow probability maintenance over a set horizon is advised.

When developing the Sliding Window algorithm, we were presented with the opportunity to use aging functions. The utility of an aging function is the ability to keep the best probability for each set of predicted events for a limited time window. After calculating the highest probability from inside the sliding window, if a past prediction has better chances at forecasting the next event correctly, then that prediction is chosen. The way this is achieved includes storing a constant number of events with their respective probability, along with how many time steps have passed since their last occurrence. That probability has been decreasing in relation to the chosen aging function and the time elapsed.

To further elaborate how this aging matrix functions, we present the following example. Let us suppose that in a 3-stream event dataset, A, B and C the aging matrix of size 3 contains the following:

Table 2: Sliding Window: Aging Matrix Example (a)

Aging Matrix		
Event	Probability	Steps Passed
AC	0.7	2
B	0.2	1
BC	0.3	0

Now, assuming that the next event is again AC, with a probability of 0.2 and every probability available inside the window is less than 0.7, the choice is obviously AC from inside the aging matrix. The aging matrix is now:

Table 3: Sliding Window: Aging Matrix Example (b)

Aging Matrix		
Event	Probability	Steps Passed
B	0.2	2
BC	0.3	1
AC	0.7	0

If none of those probabilities are utilized during their lifetime, they are popped from the matrix and replaced by new sets. During experimentation, the aging matrix capacity was set to $n = 20$ after testing various other sizes.

Aging could appear in 2 ways: (a) linear, reducing every probability by a constant percentage on each step (5% in this case), (b) exponential [4], by a predefined factor (set to $k = 0.3$), and function $\lambda = \exp(-ki)$. Those factors were also set after tests to decide on the ones that procure the best accuracy metrics.

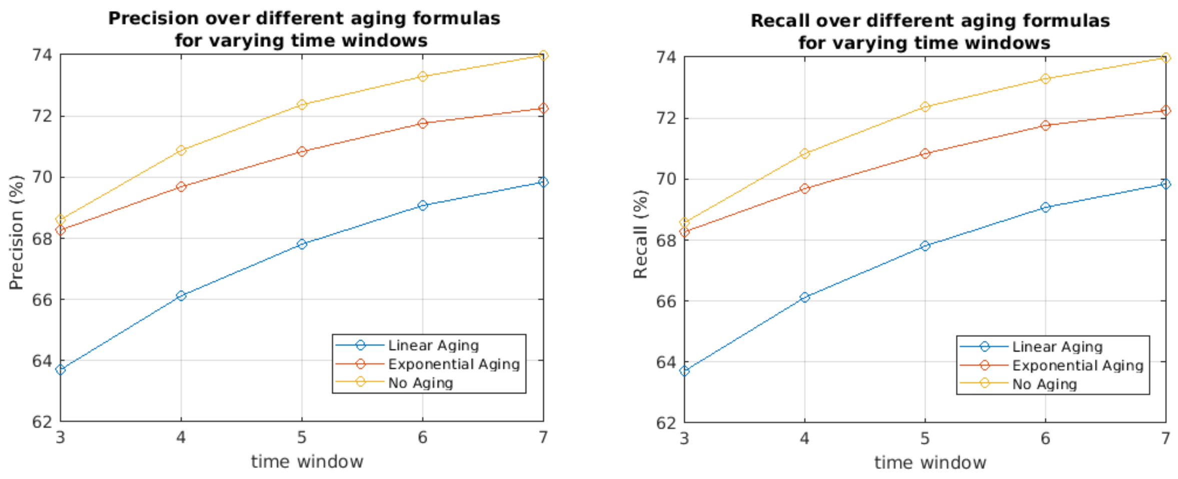


Figure 11: Precision and Recall Percentages for different Aging Formulas

Even though the use of aging functions was implemented to increase the algorithm's precision and recall, these graphs show that the acquired results are actually worse. Even though aging provides with a wider variety of probabilities for future events to choose from, it also causes more predictions, which, when not accurate, trigger a drop of these metrics.

For all figures regarding Sliding window, where fixed values for the parameters are neededd, the following are applied: window size is set to 5, as a value that renders high precision scores but does not require high time complexity and an horizon of 1 step lookahead is utilized as the default algorithm. A good tradeoff so as to include most predictions would be to consider a probability threshold of 0.6, and excluding those that are less likely to happen. As detection algorithm, Shewhart performs better than CUSUM and no aging function was applied as default parameter of the algorithm.

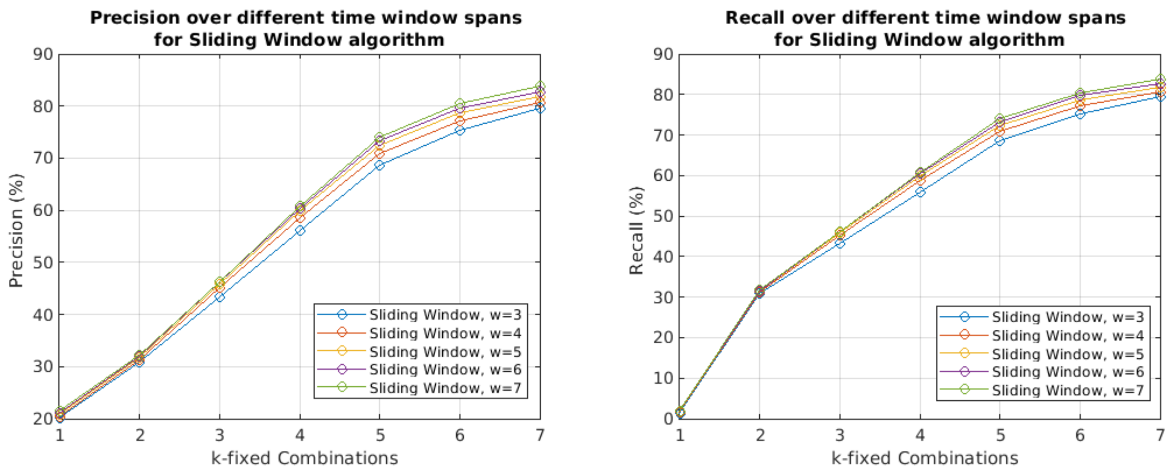


Figure 12:Precision and Recall Percentages for different Time Window Spans

It makes sense that a larger window would create more combinations and assign more chances to an event of occurring again. That would in turn increase its probability and assign its value to next steps' predictions, hence the highest accuracy scores.

Final experimentation stage should be a comparison between the two developed implementations of Stepwise Correlation and Sliding Window algorithms.

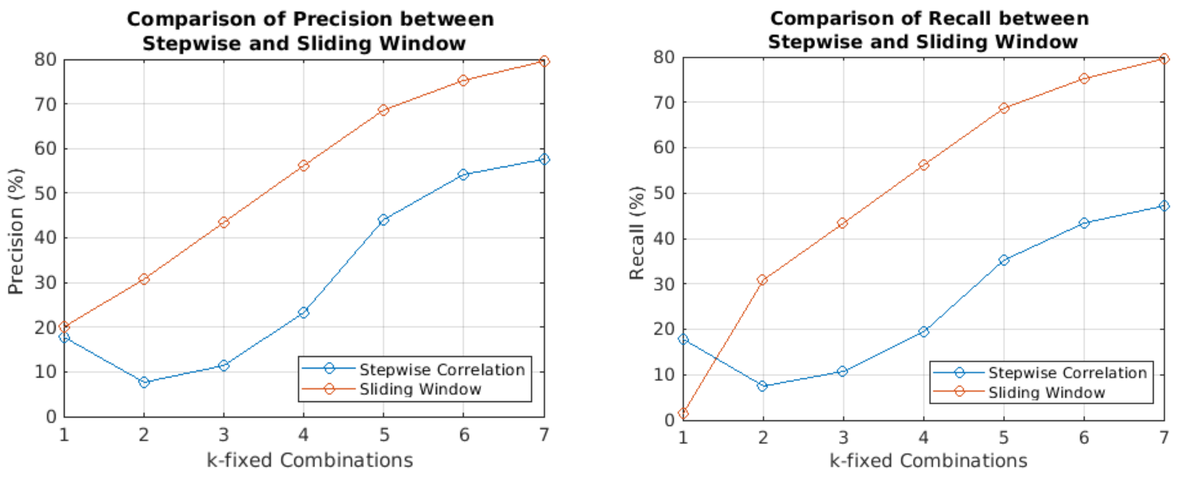


Figure 13: Comparison of Precision and Recall for Stepwise Correlation and Sliding Window algorithm

Apparently, Sliding Window performs better than Stepwise. This could be attributed to a variety of reasons. Most importantly, the algorithms differ on what they consider an accurate prediction, with Sliding accepting all possible combinations of the potentially isolated, simultaneously occurring events. The fixed probability case for Sliding also only allows predictions over a certain threshold, which reduces the number of predictions, thus increasing precision. Note that for Sliding Window the drop in middle k values does not exist. This happens due to the fact that a subset prediction is also counted as a correct prediction.

However, Stepwise provides us with the ability to skip the k parameter and use all streams, with a high accuracy (~62%), that presents a more realistic approach by including the true dataset value. That could not be implemented for Sliding, since a dataset containing 29 event streams would require up to 2^{29} probable events for every time step. Obviously, space and time complexity forbid this test case from this experimentation segment.

Finally, even though as mentioned, a broader horizon renders better results, it is outside the context of the developed algorithms, as is the use of aging functions. In order to achieve more realistic results, when fixing values, the defaults were used (1 event lookahead h=1, no use of aging function).

5. CONCLUSION

In this thesis, we discussed the flow of actions concerning event forecasting over sensor stream data. As a first measure, it is crucial to interpret what constitutes an event and create algorithms that implement this interpretation efficiently. Depending on whether or not the incoming data follow a normal distribution, the most suitable algorithm was chosen to create the event vectors. As we already noted, an established interdependence of the provided data is advised, since it produces better and more realistic results.

To produce as up-to-date results as possible, the general approach followed, assigns more predictive power to more recent events by using aging functions and removing obsolete events from the predictions' graph. Furthermore, by adding memory to the prediction algorithm (e.g. Sliding Window), old predictions concerning a larger amount of subsequent time steps may now "win" over newer, more minimal predictions that exist inside the predefined time window.

Through thorough experimentation, we managed to develop an implementation that produces results of high precision and recall. That implementation utilizes all available resources and for sets of differentiated variables, reaches up to ~80% exact matches.

With this thesis, we aim to contribute to the vast sector that is Sensor Network Data Processing and create algorithms that enable devices to act according to their indicated behavior, defined for the following time quantum, with high probability for success. In the future, we aim to extend the algorithms' applicability to unobservable system states that affect observable variables and thus, lead to incorrect predictions throughout the system. Our ultimate goal is to create a universal model that identifies all hidden states and assigns them to already known system states.

REFERENCES

[1] E. S. Page, "Continuous Inspection Scheme," *Biometrika,* no. 41, pp. 110-115, 1954.

[2] D. Montgomery, Introduction to Statistical Quality Control, Hoboken, New Jersey: John Wiley & Sons, Inc, 2005.

[3] V. Papataxiarhis and S. Hadjiefthymiades, "Event correlation and forecasting over high-dimensional streaming sensor data," *WiMob,* pp. 1-8, 10 2018.

[4] R. Klinkenberg, "Learning drifting concepts: Example selection vs. example weighting," *Intelligent Data Analysis,* no. 8, pp. 281-300, 2004.