Conception of Semantic Complex Event Pattern Mining methods on Event Streams

Focussing on predictive Episode Mining

Leon Bornemann

Department of Mathematics and Computer Science
Freie Universität Berlin

This thesis is submitted for the degree of
Master of Science

June 2016

Declaration

Does the FU have a declaration text in which I declare that I worked on this alone, up to scientific standard, did not copy anything without citing etc? If yes I will put this here.

Leon Bornemann
June 2016

Acknowledgements

And I would like to acknowledge (TODO) ...

Abstract

It is a little early for an abstract, I guess I could already write a preliminary Abstract...

Table of contents

List of figures

List of tables

Chapter 1

Introduction

This chapter serves as a rather broad introduction to the topic of this thesis and provides motivation for the work.

1.1 General Introduction and Motivation

Almost any application domain of information systems has some data that is being generated. Data is available in many different forms. One of these forms are data streams. Data streams are not limited to the recent rise in popularity of video and audio streams. On the contrary the application domains that produce data in the form of streams are very diverse. They include for example constantly running business applications that log business activities and events, sensor networks that report usage data or devices that take measurements of physical quantities (such as temperature, pressure, humidity, etc...) at certain points of time.
The fact that streams generate a constant stream of data and thus lead to a constantly growing database is a significant difference to classic applications of data mining in which there is a static (training) database. Despite that significant difference in the data representation, many fields of interest in the context of static databases remain the same for data streams. Common areas of interest are frequent patterns, predictive patterns, association rules, clustering and classification of the data entities. Approaches and algorithms that solve these problems for static databases, while by no means fully researched, are rather well known and evaluated. Applying these methods to data streams can present challenges and may demand many modifications due to the large and possibly infinite amounts of data produced by streams. Naturally, data mining methods for stream data must be especially fast, scalable and memory efficient.
Apart from the additional, algorithmic constraints on memory and computation time, data streams also present conceptual challenges. In contrast to static databases streams may

evolve over time, which can make it very difficult for algorithms to assess which past data of the stream should be considered when analyzing the currently incoming data. Recognizing these so called concept drifts is one challenge among many when processing or mining data streams.
A suitable way to look at most data streaming scenarios is that of event streams. An event can be anything that happens in the real world, which can be represented as an element of the stream. These events are commonly referred to as basic or simple events. A frequent area of interest when processing event streams is to mine complex events that consist of multiple basic events with different relationships between each other. Discovering interesting complex event patterns can be tough, especially since there may be a lot of potential candidates. Often we are only interested in specific event combinations. One possible approach to improve the mining process is to use domain knowledge. If the domain knowledge about the underlying event stream contains semantic information about the different event types it is possible to use that knowledge directly in the mining process. Semantic knowledge can take many different forms, a common one being an RDF-graph that can be examined using queries.
Figure 1.1 presents the basic idea of semantic complex event mining algorithms. On the lowest level of abstraction we have the low-level event stream, which is the unrefined data coming directly from the sources (for example sensor data). The low-level stream needs to be transformed in some way to an annotated event stream, which then in turn gets mined to discover complex events. The mining algorithm's basic input is the stream of annotated events, but it can also use the previously mentioned semantic knowledge, an ontology, which contains additional information about the event types.

In terms of the very broad term of complex events this thesis will focus on the subtopic of episode pattern mining from stream data or very large log files or databases. Episodes are a specific kind of complex events and are formally defined in chapter 2. For now it is sufficient to know that episodes are essentially complex events in which single events or entire episodes can be combined using two operators: the conjunction and the sequence operator. Figure 1.2 visualizes a simple episode pattern and example occurrences in a data stream.

So why is the mining of episodes of interest? There are many real-life use cases in which episode discovery can help predict or prevent problems. The discovery of frequent episodes can for example be related to the discovery of underlying models for data generation, which can help to better understand the data generation process. Another application is to find predictive episodes, meaning episodes which can help to predict events occurring in the future. This has already been applied to predict outages in Finnish power grids (TODO: find a citable source for this). The discovery of predictive episodes is relevant for many domains, for example sensor networks (if a certain chain of events leads to failure, predictive

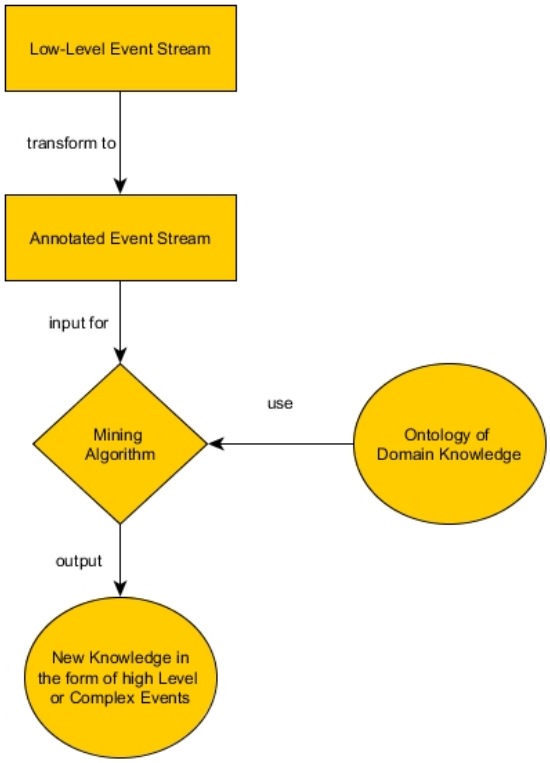

Fig. 1.1 General structure of a semantic mining process of complex events

episodes can be used to preemptively expect failures and react accordingly). The domain that will be used in the evaluation of this thesis is stock market prediction. Predicting the overall direction of the stock market or whether individual stocks will rise or fall is a difficult problem that has the obvious application of generating investment strategies. Apart from this, predictive episodes also have use cases in the enforcement of regulations. Predictive episodes could for example be used to detect illegal price arrangements of certain companies, which have been known to happen between oil and gas stations of major companies (TODO: find and cite a source for this).

In contrast to other regression and forecasting methods, such as artificial neural networks, predictive episodes have the advantage that once they are discovered they can make ad-hoc predictions in a fast moving data-stream, whereas neural networks usually forecast the closing values of stock markets of the next day based on the values of previous days. This gives predictive episodes a niche: fast moving (real-time) data-streams that need quick predictions of future events.

The rest of the thesis is outlined as follows: Chapter 2 gives an overview over relevant related work areas. Chapter 3 explains the basic terminology and gives formal definitions for important terms that are used in this thesis. TODO: the rest of the chapters.

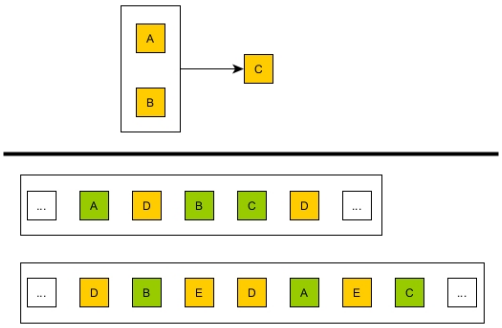

Fig. 1.2 The top half visualizes an example episode pattern, which consist of a conjunction (A and B must both occur, but the order does not matter) and a sequence (C must occur after A and B). The bottom half shows two windows of an example stream, in which occurrences of the episode are shown in green.

1.2 Contributions and Exact Research Question

This thesis aims to develop a semantic mining algorithm for predictive episodes. Since the algorithm will be used on data streams, the algorithm must have the following properties:

- The Algorithm must have an online learning mode
- The Algorithm must be able to adapt to a changing context (as the stream progresses, the underlying model may change completely)
- Fast prediction. Since streams, especially in the domain of stock markets, can have a high velocity it is important to be able to quickly predict events. If the prediction takes too long, the event which we want to predict may have already occurred before the algorithm outputs its prediction, thus robbing the user of the opportunity to take precautions.
- The Algorithm must not require to store the entire stream of events seen so far, since that is not feasible for most streams.

The algorithm will be evaluated on both synthetically generated data and real-life datasets. The latter ones will be from the domain of stock market prediction, in which the developed algorithm will be empirically compared to other approaches in terms of accuracy measures and execution time.

In summary the research question in particular is:

How can event streams be mined for episodes that effectively predict certain event

types and how can domain knowledge be used to improve the results or speed of the mining algorithm?

Chapter 2

Related Work

2.1 Related Work Overview

Since there are many different related work areas that all have their relevance to this thesis, this chapter is divided into different sections dealing with each of them in turn. Section 2.1.1 gives an overview on classical pattern mining and frequent itemset mining approaches. Section 2.1.2 continues with an outline of the state of the art classification and regression approaches after which section 2.1.3 will take a closer look on time series prediction with a focus on financial data and stock market forecasting. Section 2.1.4 then introduces related work in the very broad research area of processing and mining data streams. Afterwards section 2.1.5 presents the research area of general event processing with a focus on the mining of complex events and episodes. Finally section 2.1.6 finishes with an overview over the most relevant work (with respect to this thesis) in the research area of the semantic web.

2.1.1 Pattern Mining

Of all the fields related to this thesis basic pattern mining is arguably the most well researched and widely used related research area. First to mention are of course the standard pattern mining algorithms for frequent itemsets [15]. The authors of this paper nicely summarize different approaches, which either use candidate generation and the apriori-principle or pattern growth strategies. Problematic with the original Apriori-Algorithm is of course that while the apriori-principle helps to prune the candidate generation, its runtime can still be exponential. On top of that it requires multiple passes over the database, which is something that can be impossible given large data streams. These general pattern mining algorithms have of course already been modified or improved for more specific purposes. For example event data logs have been used to build predictive models that search for patterns that predict

critical events in the log file [8].
A rather large subfield of pattern mining is the field of (business) process mining. This particular domain usually evaluates log-files of (business) applications that log sequences of events. These logs can be mined to satisfy many different information needs, for example clustering [38] or process models [35]. Another approach in the field of business logs and process mining is outlier and anomaly detection [37]. The authors use the length of the longest common subsequence between event sequences as a distance measure to find unusual patterns of events. This field of interest is especially interesting since there are a lot of business applications, some of which use legacy software, whose behaviour is at times unclear or error-prone. Mining underlying business models or unusual behavior can help identify errors.
A subfield of pattern mining that is especially relevant for this thesis is sequential pattern mining. Comprehensive overviews are given by Slimani et al. [36] as well as Zhao at al. [41].

2.1.2 Classification and Regression

Classification and regression are the main applications of supervised learning. They are very similar and closely interlinked to each other. Classification aims to assign new, unseen objects to a class based on a model that was created from training data. Class labels are categorical. Regression is similar, but instead of assigning categorical class labels to new data points it is now the goal to generate a numerical (real) value that is close to the actual value. A comprehensive overview over most classification approaches was comprised by Kotsiantis et. al. [19]. The authors distinguish between different types of classifiers:

- logical classifiers such as Decision Trees and rule based algorithms
- preceptron based classifiers, such as the single layered perceptron or artificial neural networks
- statistical classifiers, such as naive bayes or bayesian networks
- instance based classifiers, such as k nearest neighbor
- maximum margin classifiers, such as support vector machines

It is notable that the authors do not mention ensemble learners [10], which combine several classifiers and classify new examples by letting each classifier vote. Arguably the most notable ensemble learning technique is the random forest [24]. It is important to keep in

mind that most classification approaches can also be tweaked to do regression, a random forest for instance can either be used to estimate a (binary) class label (classification) or a numerical value (regression). Classification and regression have also been attempted by using pattern mining and association rule generation [27]. Instead of mining all association rules the authors focus on finding so called class association rules. An especially relevant application of rule based regression was the usage of classification rules using minimal rule generation for the prediction of equity returns [1]. The authors modify a classification rule generation technique called R-MINI to be able to do regression. They evaluate their approach on historical stock market data from the S&P 500 data-set (data is aggregated to monthly values).

2.1.3 Prediction and Financial Time Series Forecasting

Similar to regression and classification, but not quite the same is prediction. Prediction (sometimes also called forecasting) introduces a new dimension, which is time. The task in prediction is to build a model that if given data points on a timeline will predict which data point will show up next. Due to its dependency on time, prediction plays a large role when mining time series data. Arguably the most popular models for time series prediction are artificial neural networks which are used by multiple authors with differnt variations and application domains [9] [32] [12]. Of particular interest for this thesis are of course prediction approaches that were used in the same domain, namely stock market or financial time series prediction. Naturally this has been looked into. Gestel at. al. used support vector machines to predict the closing values of stock market indices [39].Another, more recent approach used an artificial neural network with an improved learning algorithm (by integrating improved bacterial chemotaxis optimization into the back propagation of the neural network) to successfully predict the Standard's & Poor's 500 index (changes were aggregated to daily values). Another different approach is to use grey system models to predict time series [18], in this specific case the authors predict the daily foreign currency exchange rate of euro to dollar. A comprehensive comparison of different models for predicting the S&P CNX NIFTY index was carried out by kumar et. al. [20]. The best performing model in their study is the support vector machine, closely followed by a random forest. This is interesting, since random forests have not received as much attention for time series prediction as for example artificial neural networks. Of course no general conclusions can be drawn from the performance of these models for one index. An very detailed overview over a lot of work in the area can be found in the literature study done by Atsalakis et al. [3] although the focus seems to be on papers that use neural networks or support vector machines. The attentive reader will have noticed that most of the previously named examples focus on

predicting stock indices, a.k.a the general direction of the market. Predicting individual stock values has been looked at less, but there is of source some previous work to be considered. For example mahfoud et. al. compare genetic algorithms to neural networks for the prediction of individual stocks [28].

2.1.4 Data Stream Processing

Many research areas in classical data mining are also of interest when processing streams. As already mentioned in the introduction however, data streams impose severe restrictions on the algorithms, such them having to use only one pass over the data and that they must be incremental. Almost no data mining algorithms for the classical scenario in which the underlying data is a static database or even a data-set that fits into main memory have these properties. Thus the algorithms need to be modified and often approximations have to be made. A comprehensive basic overview over the application of different data mining tasks and how they can be applied to streams was comprised by Gaber et. al. [13].
As already mentioned many data mining algorithms for data streams make compromises of some sort or employ appoximations. Especially in the area of pattern mining there are quite a few approaches that use approximations or focus on the most recent data. Gianella et. al. have for example developed an algorithm to incrementally maintain approximately frequent itemsets for the most recent time windows using tilted time window frames [14]. Approaches to mine frequent items without tilting the time windows exist as well, such as the sticky sampling or the lossy counting algorithm [29]. These algorithms can be generalized to mine frequent itemsets, too. If this is done however pruning the candidates may become an issue if the main memory is small (the number of candidates to maintain may be too large).
A paper also related to regression and time series forecasting was published by Papadimitriou et. al. [34]. The authors tackle the problem of cyclic time series mining, as well as time series prediction of numerical values in potentially infinite data streams.
The rise in popularity of mining data streams also pushes research areas like distributed data mining, in which parallelized versions of data mining algorithms are researched. Large data streams and distributed systems often go hand in hand, which creates the need for distributed algorithms. Such solutions have been researched for example for frequent pattern mining [25], association rule mining [2], clustering [17] and classification (in this case a distributed boosting algorithm) [23].
A slightly different direction to data stream processing is querying data streams. Some contributing data scientists have thus approached streams in a similar manner as relational databases [33]. The authors present their progress at building a general purpose Data Stream Management System (DSMS) prototype, as a pendant to the traditional Database

Management Systems (DBMS). They suggest the usage of an extension of SQL as a stream query language to allow for continuous queries and discuss approximation techniques.

2.1.5 Complex Event Processing and Episode Mining

Before talking about the state of the art in complex event processing it is first important to clarify the term *"event"* which is a is very broad term that is used in many different areas of science. Even when restricted to computer science there may be different definitions with subtle differences. As already explained in the introduction this thesis will focus on events as defined by the glossary of the event processing society [26]. When talking about processing complex events there is an important difference between two cases:

- The patterns of interest are known before looking at the data. This is called complex event detection.
- There is no prior knowledge about which patterns might be interesting, they need to be discovered while looking at the data. This is called complex event discovery or complex event mining (which basically means that data mining methods need to be employed)

Complex event detection usually revolves around specification and query languages for complex events [11]. Quite a few different specification and query languages have been developed, such as SNOOP [7] or the SASE event processing language [40].
Discovering interesting complex events of arbitrary structure in data streams is a very challenging task, thus most work focuses on specific types of complex events. A popular example is mining frequent sequences of events (basically complex events that only use the sequence operator) [6] [16].
This thesis deals with a more expressive type of complex events: Episodes. Originally episodes were researched without any relation to event processing [30]. Episodes have roughly been categorized as serial, parrallel or composite and there are different mining methods proposed for each of these [30] [42]. The connection between episodes and hidden markov models was also explored in a PHD thesis by Laxman [21]. Evaluating episode mining algorithms on real-life datasets is often difficult due to a lack of knowledge about the ground truth. Thus, generation of realistic, synthetic datasets has been looked into as well [43].

2.1.6 Semantic Web

TODO

2.2 Basic Definitions and Terminology

This section introduces relevant definitions and terminology that was introduced in previous work and will be used in this thesis.

2.2.1 Event Processing Terminology

Most of the basic event terminology in this subsection is taken from the event processing glossary created by the Event Processing Technical Society [26]. Note that some of the definitions may be slightly altered or simplified. This is due to the fact that the event processing technical society uses these terms for a very general description of event processing and event processing architectures and thus some original definitions are more complex than what is needed in this thesis. The definitions given here aim to establish a clear terminology for this thesis.

Definition 1 ***Event*** *An event is either something that is happening in the real world or in the context of computer science an object that represents a real world event and records its properties. The latter can also be referred to as an event object or an event tuple. Note that the term is overloaded, but the context usually gives a clear indication of what is meant.*

Definition 2 ***Simple Event*** *A simple event is an event that is not viewed as summarizing, representing, or denoting a set of other events. Sometimes also referred to as a basic events.*

These two definitions can sometimes cause confusion. It is important to note that the term event is the most general term, since it can refer to any kind of event, be it simple, derived or complex (TODO: better, see below for definitions, or move this note). A simple event however is the most basic form of an event and often the ingredient for the creation of more complex events: Given a simple events it is possible to derive events from those or the absence of those. For example the absence measurement events of sensor A could be used to derive the event of the sensor A becoming defect. These events are called derived events:

Definition 3 ***Derived Event*** *A derived Event or synthesized event is an event that is generated according to some method or based on some reasoning process.*

It is also possible to combine multiple simple events to form what we refer to as complex events:

Definition 4 ***Complex Event*** *A complex event is a derived event that is created by combining other events. The events can be combined by using certain operators, for example disjunction, conjunction or sequence. An example would be* $(A \wedge B) \rightarrow C$ *(event A and B in any order followed by event C).*

This is a very broad definition of complex events. The choice of allowed operators strongly impacts the expressiveness of complex events. A specific kind of complex events are episodes (see TODO: reference), which will be the main focus of this thesis. The next notion that needs to be considered is that each individual event normally belongs to a certain class of events, which we refer to as the event type:

Definition 5 ***Event type*** *The event type, sometimes also referred to as event class, event definition, or event schema is a label that identifies events as members of an event class.*

Another important term that was not defined in the event processing glossary, but is very relevant to the topic at hand is the notion of a type alphabet:

Definition 6 ***Type Alphabet*** *The event alphabet is the set of all possible event types that can occur in the observed system.*

Event alphabets are often implicitly defined when mining frequent itemsets, patterns or episodes (see definition below). So far we have looked at events without considering the scenario we are most interested in which are event streams. To do so we need the notion of timestamps:

Definition 7 ***Timestamp*** *A time value of an event indicating its creation or arrival time.*

Given that we can define an event stream:

Definition 8 ***Event Stream*** *An event stream is an ordered sequence of events, usually ordered by the event timings.*

Note that this rather broad definition of an event stream does not assume anything about the kind of event that is contained in it. A stream can contain very basic forms of events (simple events) but can also be made up out of derived events or even complex events.

2.3 Event Episodes

2.3.1 Episode patterns and Occurances

As already mentioned, episodes are complex events whose basic building blocks are simple events. Note that in order to make use of episodes we require that all simple events have a type and we have a finite, previously known event alphabet, that contains these types, which we refer to as Σ. We follow up with a formal definition:

Definition 9 ***Episode*** *An event episode (also sometimes called episode pattern or elementary episode) E of length m (also called m-episode) is defined as a triple:* $E = (N_E, \leq_E, g_E)$ *where* $N_E = \{n_1, ..., n_m\}$ *is a set of nodes,* $\leq_E$ *is a partial order over* N_E *and* $g_E : V_E \rightarrow \Sigma$ *is a mapping that maps each node of* N_E *to an event type. TODO: add source*

There are a two special types of episodes that deserve special mention since they have received the most attention in the available literature. These are called serial and parallel episodes:

Definition 10 ***Serial Episode*** *An episode* $E = (N_E, \leq_E, g_E)$ *is called a serial episode if* $\leq_E$ *is a total order.*

Definition 11 ***Parallel Episode*** *An episode* $E = (N_E, \leq_E, g_E)$ *is called a parallel episode if* $\leq_E = \emptyset$*, in other words if there is no ordering imposed on* N_E *at all.*

Essentially, serial episodes are sequences, while parallel episodes are multisets. The original paper by Manilla et.al. (TODO cite) also introduces the notion of composite episodes. We repeat the definition here:

Definition 12 ***Composite Episode*** *An episode* $E = (N_E, \leq_E, g_E)$ *is called a composite episode if* $g_E : V_E \rightarrow \Sigma \cup C^*$*, where* C^* *is the set of all composite episodes.*

This recursive definition of composite episodes may be confusing at first but it has the advantage that any elementary episode can be represented as a composite episode which is exclusively a serial or parallel composition of serial, parallel or composite subepisodes (TODO: define subepsiodes).
Interestingly, there are other parts of the related work that use the term *composite episodes* but deviate from definition 12. For example Baathorn et. al. propose a method for finding composite episodes [4]. However they define composite episodes as a sequence of parallel episodes, which is more restrictive. Not all elementary episodes can be represented as sequences of parallel episodes (TODO: example for that). Also Baumgarten et. al. use this definition [5] when they present an approach to mine descriptive composite episodes. In this thesis we will stick to the definition 12.

TODO: representable as DAC

If we want to denote quick, simple episodes formally, we will use $\rightarrow$ as the sequence (ordering) operator. To show that there is no order specified between two nodes we use $\|$ as the parallel operator. For example $(A \,\|\, B) \rightarrow C$ denotes a composite episode of length 3, which specifies that it does not matter in which order A or B occur, but C must occur

after both A and B. If we want to discuss more complex episodes we will visualize them graphically like in figure TODO
It is helpful to think of episodes as a template or pattern for concrete occurrences. In oder to define what we mean by an episode occurance, we first need to formally introduce the notion of an event sequence.

TODO: include the more simple Definition.

Definition 13 ***Event sequence*** *An event sequence is defined as an ordered list of tuples* $S = [(T_1,t_1),...,(T_n,t_n)]$ *where* $T_i \in \Sigma$ *is the event type of the i-th event and* $t_i \in \mathbb{N}^+$ *is the timestamp of the i-th event. The sequence is ordered according to the timestamps, which means that* $\forall i,j \in 1,...,n\ i < j \implies t_i \leq t_j$.

Definition 14 ***Episode Occurrence*** *An event episode* $E = (N_E, \leq_E, g_E)$ *is said to occur in a sequence S if events of the types that the nodes in* N_E *are mapped to by* g_E*, occur in S in the same order that they occur in the episode. More formally if we are given a sequence of events* $S = [(T_1,t_1),...,(T_n,t_n)]$ *we can define an occurrence of E as an injective Map* $h : N_E \to \{1,...,n\}$*, where* $g_E(n_i) = T_{h(n_i)}$ *and* $\forall v,w \in V_E : v \leq_E w \implies t_{h(v)} \leq t_{h(w)}$ *holds.*

2.3.2 Episode Detection and general Mining Algorithm

When discovering episodes in a sequence one is usually interested in those episodes that occur frequently, meaning more often than a user defined threshold. This is similar to all the different kinds of pattern mining algorithms, such as the Apriori algorithm for finding frequent itemsets (TODO: cite). A general algorithm for mining the episodes occuring frequently in a sequence is given in algorithm 1. The algorithm is very alike the basic Apriori algorithms, since it uses a level-wise, breadth first search by first identifying all frequent episodes of a certain length i and then uses these frequent episodes to generate candidates (possibly frequent episodes) of length $i+1$. In order for this to be correct episode frequency must follow the Apriori principle (TODO cite). The Apriori principle states that if a pattern is frequent all of its subpatterns (TODO: define subepisode) must be frequent, too. Intuitively one would assume that this is true for episodes but strictly speaking this depends on the definition of episode frequency. However any frequency definition of episodes that does not satisfy the apriori principle would be highly questionable to say the least. To the best of our knowledge all frequency definitions proposed in the literature satisfy the apriori principle.

In summary, the general mining algorithm for frequent episodes requires:

Algorithm 1 General mining algorithm for frequent episodes

```
1: function EPISODEMINING
2:     C_i ← Episodes of Size 1
3:     freq ← ∅
4:     i ← 1
5:     while C_i ≠ ∅ do
6:         Count frequencies of each Episode E ∈ C_i
7:         L_i ← {E | E ∈ C_i ∧ C_i is frequent}
8:         freq ← freq ∪ L_i
9:         C_{i+1} ← Generate Episode Candidates of length i+1 from L_i
10:        i ← i+1
11:    return freq
```

- A definition of episode frequency, that does not violate the apriori principle
- An algorithm for counting episode frequency (of concrete candidates) according to this definition
- An algorithm to generate episode candidates

It may be a bit confusing that we need a definition of episode frequency for such a mining algorithm. Since we have already defined what an occurrence of an episode looks like it would seem that counting all occurrences of an episode would yield its frequency. While this is a possible definition of frequency, it is important to note that finding all occurrences of an episode within a sequence is neither practical nor useful. An example will demonstrate the problem with this. Consider the simple serial episode $A \rightarrow B$ and a sequence of length $2 \cdot n$ which repeats the subsequence (A,B) n times. One quickly realizes that the number of episode occurrences is very large due to the possibility of overlapping episode occurrences. In this particular case there are already $\frac{n \cdot (n+1)}{2}$ possible occurrences. This number swiftly increases with the size of the episode pattern, since it introduces more degrees of freedom (TODO: better word). In fact the number of possible occurrences of a serial episode in a fixed length sequence increases exponentially with the size of the episode (TODO: If I claim this I need to prove it, right?). Naturally the number of possible parallel and composite episode occurrences is even larger, since they are less restrictive in the order of the events.
This leads to various frequency definitions of episodes in the literature, which will be dealt with in the following sections (TODO: mention the sections). Each frequency definition comes with its own frequency counting algorithm. The algorithm for generating candidates is independent of the frequency definition and is present in (TODO: refer to section for that)

2.3.3 Window based frequency

To the best of our knowledge the window based frequency was the first frequency definition for episodes to gain general popularity. It was conceived by Mannila et. al. [30], although the frequency counting algorithms were only mentioned in text form. The same authors specified the algorithms in a later paper [31], which acts as the primary source for the overview given in this subsection. In order to define the window based frequency we first need the notion of a time window:

Definition 15 ***Time Window*** *Given a sequence of events S we define the Time Window* $W(S,q,r)$ *with* $q,r \in \mathbb{N}^+$ *and* $q < r$ *as the ordered subsequence of S that includes all events of the annotated event stream S that have a timestamp t where* $q \leq t \leq r$*. We call* $w = r-q+1$ *the size of Window* W*.*

TODO: nice picture visualizing the different windows

Definition 16 ***Episode Frequency - Window based Definition*** *Given a sequence of events S, a fixed window size of w and an Episode E, we define the window based frequency* $w\_freq(E)$ *as the number of windows W with size w of S in which E occurs:* $w\_freq(E) = |\{W(S,q,r) \mid r-q+1 = w \wedge E \text{ occurs in } W\}|$*. TODO: incorporate sequence bounds and number of windows*

For example given a sequence $S = [(12,A),(14,B),(19,C),(22,A),(34,D)]$ the Episode $E = B \rightarrow A$ occurs in window $W(S,14,22)$.

This definition can be confusing at first since it is intended that episode occurrences that are comprised of the exact same events count just as many times as there are windows in which the events appear. If we have a window size of $w = 11$ for the previously mentioned example, we can find the Episode *B after A* in the consecutive windows $W(S,12,22)$, $W(S,13,23)$ and $W(S,14,24)$, which means we will get a frequency of 3 just for the two events $(14,B)$ and $(22,A)$. This effect obviously increases with the window size. Note that for each episode E we only count one occurrence per window W, no matter how many occurrences of E there are in W.
When determining the window based frequency the naive approach would be to check each window of the sequence separately. Since the windows are adjacent though, it is possible to only iterate over the sequence once and determine the window based frequency for each candidate episode. Most papers focus purely on parallel and serial episodes and do not give an algorithm for composite episodes. The algorithms to determine the window based

frequency of serial and parallel episodes are given in algorithm 2 and algorithm TODO respectively.

TODO: window based algorithm for parallel is absolutely awful to read and may even be wrong

Mannila et. al. claim that each composite episode can be broken down into partial episodes, which are serial and/or parallel (TODO: cite). However they do not specify and algorithm for doing this.

TODO: add window based frequency counting algorithm, restructure chapter

2.3.4 Minimal Occurance based frequency

The second definition does not require a fixed window size to be specified but instead uses the concept of minimal occurrences:

Definition 17 ***Minimal Occurrence*** *An event episode E is said to occur minimally in a window $W(S,q,r)$ if E occurs in W and there is no subwindow of W in which E also occurs. In this context we also refer to the window W itself as a minimal occurrence of E.*

Definition 18 ***Episode Frequency - Minimal Occurrence based Definition*** *Given a high level event stream S and an Episode E, we define the minimal occurrence based frequency $mo\_freq(E)$ as the number of minimal occurrences of E in S.*

2.3.5 Non-Overlapping Occurances

The third definition introduces the concept of non-overlapping occurrences:

Definition 19 ***Non-Overlapping Occurrences*** *Given a m-Episode $E = (N_E, \leq_E, g_E)$ where $N_E = \{n_1, ..., n_m\}$, two occurrences h_1 and h_2 of E are non-overlapped if either*

- $\forall n_j \in N_E : h_2(n_1) > h_1(n_j)$ *or*
- $\forall n_j \in N_E : h_1(n_1) > h_2(n_j)$

A set of occurrences is non-overlapping if every pair of occurrences in it is non-overlapped.

This leads to the Definition:

Definition 20 ***Episode Frequency - Non-Overlapping Occurrences based Definition*** *Given a high level event stream S and an Episode E, we define the non-overlapping occurrence based frequency $noo\_freq(E)$ as cardinality of the largest set of non-overlapped occurrences of E in S [22].*

2.3.6 Summary

When looking at these definitions it is not clear whether any of these is always superior to or more useful than the other since they have different properties. We mention them briefly:

- As already mentioned in the above example. The window based frequency counts an episode occurrence that is comprised of the same events in multiple windows. This might especially distort the count if the window size is high and the events in the episode happen with minimal delay between them.
- The minimal occurrence based definition of frequency does not suffer from the problem of the previous point
- The window based definition has the advantage that it already incorporates a fixed size during which episodes may occur, meaning there can not be episodes that stretch over a time period larger than the fixed window size w. This might be beneficial for potential algorithms, since it reduces the search space for episodes. On top of that it is also closer to reality, since episodes normally happen within a small time window [**?**]. Of course also the minimal occurrence based definition can be extended to incorporate a maximal time span.
- Both definitions do not consider multiple occurrences of an episode in the same time window.

TODO: include more definitions, possibly my own.

These definitions are relevant when mining frequent episodes from databases or streams. We are however not specifically interested in all frequent episodes in thesis but instead focus on predictive episodes: TODO: define predictive Episodes.

Algorithm 2 Calculate Window based Frequency for parallel Episodes

Require: Let C be the set of candidate parallel episodes, and let $S = [(T_1, t_s), ..., (T_n, t_e)]$ be a sequence of events, let *win* be the window size and finally let *minS* be the minimum support. TODO: support vs frequency!

1: *// Initialization*
2: **for** each $\alpha \in C$ **do**
3: **for** each $A \in \alpha$ **do**
4: $A.count \leftarrow 0$
5: **for** $i \in \{1, ..., |\alpha|\}$ **do**
6: $contains(A, i) \leftarrow \emptyset$
7: **for** each $\alpha \in C$ **do**
8: **for** each $A \in \alpha$ **do**
9: $a \leftarrow$ number of events of type A in α
10: $contains(A, a) \leftarrow contains(A, a) \cup \{\alpha\}$
11: $\alpha.eventCount \leftarrow 0$
12: $\alpha.freq \leftarrow 0$
13: *// Recognition*
14: **for** $start \leftarrow t_s - win + 1$ to t_e **do**
15: *//Bring new events to the window*
16: **for** each $(t, A) \in S$ where $t = start + win - 1$ **do**
17: $A.count \leftarrow A.count + 1$
18: **for** each $\alpha \in contains(A, A.count)$ **do**
19: $\alpha.eventCount \leftarrow \alpha.eventCount + A.count$
20: **if** $\alpha.eventCount = |\alpha|$ **then**
21: $\alpha.inwindow \leftarrow start$
22: *// Drop old events out of the window*
23: **for** each $(t, A) \in S$ where $t = start - 1$ **do**
24: **for** each $\alpha \in contains(A, A.count)$ **do**
25: **if** $\alpha.eventCount = |\alpha|$ **then**
26: $\alpha.freq \leftarrow \alpha.freq + \alpha.inwindow - start$
27: $\alpha.eventCount \leftarrow \alpha.eventCount - A.count$
28: $A.count \leftarrow A.count - 1$
29: **return** $\{\alpha \mid \alpha \in C \wedge \frac{\alpha.freq}{t_e - t_s + win - 1} \geq minS\}$

Algorithm 3 Calculate Window based Frequency for serial Episodes

Require: Let C be the set of candidate serial episodes, and let $S = [(T_1,t_s),...,(T_n,t_e)]$ be a sequence of events, let *win* be the window size and finally let *minS* be the minimum support. TODO: support vs frequency!

// Initialization
for each $\alpha \in C$ **do**
  for $i \in \{1,...,|\alpha|\}$ **do**
    $\alpha.initialized[i] \leftarrow 0$
    $waits(\alpha[i]) \leftarrow \emptyset$
for each $\alpha \in C$ **do**
  $waits(\alpha[1]) \leftarrow waits(\alpha[1]) \cup \{(\alpha,1)\}$
  $\alpha.freq \leftarrow 0$
for $i \in \{t_s - win,...,t_s - 1\}$ **do**
  $beginsat(t) \leftarrow \emptyset$
// Recognition
for $start \leftarrow t_s - win + 1$ to t_e **do**
  //Bring new events to the window
  $beginsat(start + win - 1) \leftarrow \emptyset$
  $transitions \leftarrow \emptyset$
  for each $(t,A) \in S$ where $t = start + win - 1$ **do**
    for each $(\alpha, j) \in waits(A)$ **do**
      if $j = |\alpha| \wedge \alpha.initialized[j] = 0$ **then**
        $\alpha.inwindow \leftarrow start$
      if $j = 1$ **then**
        $transitions \leftarrow transitions \cup \{(\alpha,1,start + win - 1\}$
      else
        $transitions \leftarrow transitions \cup \{(\alpha,j,initialized[j-1])\}$
        Remove $(\alpha, j-1)$ from $beginsat(\alpha.initialized[j-1])$
        $\alpha.initialized[j-1] \leftarrow 0$
        Remove (α, j) from $waits(A)$
  for each $(\alpha, j, t) \in transitions$ **do**
    $\alpha.initialized[j] \leftarrow t$
    $beginsat(t) \leftarrow beginsat(t) \cup \{(\alpha, j)\}$
    if $j \leq |\alpha|$ **then**
      $waits(\alpha[j+1]) \leftarrow waits(\alpha[j+1]) \cup \{(\alpha, j+1)\}$
  // Drop old events out of the window
  for each $(\alpha, l) \in beginsat(start - 1)$ **do**
    if $l = |\alpha|$ **then**
      $\alpha.freq \leftarrow \alpha.freq + start - \alpha.inwindow$
    else
      Remove $(\alpha, l+1)$ from $waits(\alpha[l+1])$
      $\alpha.initialized[l] \leftarrow 0$
return $\{\alpha \mid \alpha \in C \wedge \frac{\alpha.freq}{t_e - t_s + win - 1} \geq minS\}$

Chapter 3

Basic Terminology and Definitions

This chapter establishes the terminology and basic formal definitions for this thesis and defines the problem of mining predictive episodes in a semi formal way (TODO: do that!).

3.1 Complex Pattern Mining: Basic Definitions

The following definitions are not taken from the previously mentioned event processing glossary, but will still be central for the agenda of this thesis.

Definition 21 ***Low Level Event*** *A Low Level Event l is defined as a tuple $l = (t, V)$, where $t \in \mathbb{N}^+$ is the timestamp of its occurance and $V = (v_1, ..., v_k)$ with $v_i \in S_i$ is a vector of values reported with the event. S_i is a set of all possible values for the position i which we will not narrow down further here since this obviously depends on the application domain. Common data types would of course be real numbers or categorical values.*

Low level events are essentially event objects (as defined above), we just demand that their properties must be expressed as a tuple.

Definition 22 ***Annotated Event*** *An annotated event e is defined as a tuple $e = (t, T)$, where $t \in \mathbb{N}^+$ is the timestamp and $T \in \Sigma$ is its derived event type. Σ is the set of all possible event types, which will sometimes also be referred to as the event type alphabet.*

There are a few things to note here. Annotated events are still basic event objects, however we are not interested in their specific properties except for their type label, which is what makes them annotated. Furthermore this definition as well as the definition of low level events assumes that events are instantaneous. This is not always the case, but we limit the focus of this thesis to the mining of instantaneous events. There are other variants of event

definitions that also allow for events to have a duration. Given these defintions we can start defining what streams of events look like:

Definition 23 ***Low Level Event Stream*** *A Low Level Event Stream S_L of length n is defined as a sequence $S_L = (l_1, ..., l_n)$, where all l_i are low level events and given two events $l_i = (t_i, V_i)$ and $e_j = (t_j, V_j)$ we know that $i \leq j \implies t_i \leq t_j$ (the stream is sorted according to the timestamps).*

The definition of an Annotated Event Stream is very similar:

Definition 24 ***Annotated Event Stream*** *An Annotated Event Stream S_A of length n is defined as a sequence $S_A = (e_1, ..., e_n)$, where all e_i are annotated events and given two events $e_i = (t_i, A)$ and $e_j = (t_j, B)$, where $A, B \in \Sigma$ and we know that $i \leq j \implies t_i \leq t_j$.*

These definition somewhat simplify reality, since they assume that the stream has a certain length, in other words the stream is finite. This is not problematic if we can assume the following:

- n always contains the number of all elements we have seen so far
- n gets updated whenever we read a new event

However we need to keep this restriction in mind, since the size of the stream will become relevant when determining frequency of complex events. Given these two definitions we can give a general definition of the first step of the basic problem setting: the transformation of a low level event stream to an annotated event stream as a function:

Definition 25 ***Transformation Function*** *A transformation function is a function that takes a low level event S_L as an input and maps it to a corresponding annotated event stream S_A.*

Obviously this is a very broad definition, since in most cases this function is completely dependent on the domain as well as target event type alphabet and thus by extension the complex events the user is interested in. The concrete transformation function though is very important for the whole mining process, since if the transformation is done incorrectly or not in an optimal way the results of the mining algorithms that look the annotated event stream are likely to be unsatisfying or misleading. When designing a transformation function one has to deal with a lot of issues, for example whether to use aggregation techniques and if yes how and at which granularity. These issues are discussed in detail in (TODO: discuss in a later chapter and refer to it).

There is one more general term that is relevant for the context of this thesis. Most of the time we are not interested in the whole stream, but instead only look at a small window:

In the following, whenever we speak of a stream without any further context we refer to an annotated event stream (likewise we will often only use S instead of S_A to denote an annotated event stream formally).

3.2 Event Episodes

Given an annotated stream there are multiple ways to define episodes of events. We will give two different definitions, first a very compact and formal definition and second a longer definition that is a bit more graphical and less formal.

Definition 26 ***Episode*** *An event episode (also sometimes called episode pattern) E of length m (also called m-Episode) is defined as a triple: $E = (N_E, \leq_E, g_E)$ where $N_E = \{n_1, ..., n_m\}$ is a set of nodes, $\leq_E$ is a partial order over N_E and $g_E : V_E \to \Sigma$ is a mapping that maps each node of N_E to an event type. If $\leq_E$ is a total order we call E a serial episode, if there is no ordering at all we call E a parallel episode. Otherwise we speak of composite episodes.*

If we want to denote quick, simple episodes formally, we will use $\to$ as the sequence (ordering) operator. To show that there is no order specified between two episodes we use $\|$ as the parallel operator. For example $(A \| B) \to C$ denotes a composite episode of length 3, which specifies that it does not matter in which order A or B occur, but C must occur after both A and B. If we want to discuss more complex episodes we will visualize them graphically like in figure TODO
It is helpful to think of episodes as a template or pattern for concrete occurrences, which we define next:

Definition 27 ***Episode Occurrence*** *An event episode $E = (N_E, \leq_E, g_E)$ is said to occur in a window W if events of the types that the nodes in N_E are mapped to occur in W in the same order that they occur in the episode. More formally if we are given a sequence of events $S = ((t_1, T_1), ..., (t_n, T_n))$ (which may be a Time Window, aka. a subsequence of the original stream) we can define an occurrence of E as an injective Map $h : N_E \to \{1, ..., n\}$, where $g_E(n_i) = T_{h(n_i)}$ and $\forall v, w \in V_E : v \leq_E w \implies t_{h(v)} \leq t_{h(w)}$ holds.*

For example given the high level event stream $S = [(12, A), (14, B), (19, C), (22, A), (34, D)]$ the Episode $E = B \to A$ occurs in window $W(S, 14, 22)$.

TODO: include the more simple Definition.

3.3 Frequency of Episodes

Defining frequency of episodes is surprisingly complex, since different definitions have a considerable impact on potential algorithms. Thus many different definitions exist, which will be briefly presented in this section. The first two definitions, which we will refer to as window based frequency and minimal occurance based frequency, are mentioned by Zimmermann, when he presents his method for synthetic episode generation [43], but were originally conceived in (TODO: include original sources). We refer to the third definition as the non-overlapping occurrence based Frequency which was suggested by Laxman et al. [22].

Definition 28 ***Episode Frequency - Window based Definition*** *Given a high level event stream S, a fixed window size of w and an Episode E, we define the window based frequency* $w\_freq(E)$ *as the number of windows W with size w of S in which E occurs:* $w\_freq(E) = |\{W(S,q,r) \mid r-q+1 = w \wedge E \text{ occurs in } W\}|$.

This definition can be confusing at first since it is intended that episode occurrences that are comprised of the exact same events count just as many times as there are windows in which the events appear. If we have a window size of $w = 11$ for the previously mentioned example, we can find the Episode *B after A* in the consecutive windows $W(S,12,22)$, $W(S,13,23)$ and $W(S,14,24)$, which means we will get a frequency of 3 just for the two events $(14,B)$ and $(22,A)$. This effect obviously increases with the window size.
The second definition does not require a fixed window size to be specified but instead uses the concept of minimal occurrences:

Definition 29 ***Minimal Occurrence*** *An event episode E is said to occur minimally in a window* $W(S,q,r)$ *if E occurs in W and there is no subwindow of W in which E also occurs. In this context we also refer to the window W itself as a minimal occurrence of E.*

Definition 30 ***Episode Frequency - Minimal Occurrence based Definition*** *Given a high level event stream S and an Episode E, we define the minimal occurrence based frequency* $mo\_freq(E)$ *as the number of minimal occurrences of E in S.*

The third definition introduces the concept of non-overlapping occurrences:

Definition 31 ***Non-Overlapping Occurrences*** *Given a m-Episode $E = (N_E, \leq_E, g_E)$ where $N_E = \{n_1, ..., n_m\}$, two occurrences h_1 and h_2 of E are non-overlapped if either*

- $\forall n_j \in N_E : h_2(n_1) > h_1(n_j)$ *or*
- $\forall n_j \in N_E : h_1(n_1) > h_2(n_j)$

A set of occurrences is non-overlapping if every pair of occurrences in it is non-overlapped.

This leads to the Definition:

Definition 32 ***Episode Frequency - Non-Overlapping Occurrences based Definition*** *Given a high level event stream S and an Episode E, we define the non-overlapping occurrence based frequency $noo\_freq(E)$ as cardinality of the largest set of non-overlapped occurrences of E in S [22].*

When looking at these definitions it is not clear whether any of these is always superior to or more useful than the other since they have different properties. We mention them briefly:

- As already mentioned in the above example. The window based frequency counts an episode occurrence that is comprised of the same events in multiple windows. This might especially distort the count if the window size is high and the events in the episode happen with minimal delay between them.
- The minimal occurrence based definition of frequency does not suffer from the problem of the previous point
- The window based definition has the advantage that it already incorporates a fixed size during which episodes may occur, meaning there can not be episodes that stretch over a time period larger than the fixed window size w. This might be beneficial for potential algorithms, since it reduces the search space for episodes. On top of that it is also closer to reality, since episodes normally happen within a small time window [**?**]. Of course also the minimal occurrence based definition can be extended to incorporate a maximal time span.
- Both definitions do not consider multiple occurrences of an episode in the same time window.

TODO: include more definitions, possibly my own.

These definitions are relevant when mining frequent episodes from databases or streams. We are however not specifically interested in all frequent episodes in thesis but instead focus on predictive episodes: TODO: define predictive Episodes.

References

[1] Apte, C. and Hong, S. J. (1994). Predicting equity returns from securities data with minimal rule generation. In *KDD Workshop*, pages 407–418.

[2] Ashrafi, M. Z., Taniar, D., and Smith, K. (2004). Odam: An optimized distributed association rule mining algorithm. *IEEE distributed systems online*, (3):1.

[3] Atsalakis, G. S. and Valavanis, K. P. (2009). Surveying stock market forecasting techniques–part ii: Soft computing methods. *Expert Systems with Applications*, 36(3):5932–5941.

[4] Bathoorn, R. and Siebes, A. (2007). Finding composite episodes. In *International Workshop on Mining Complex Data*, pages 157–168. Springer.

[5] Baumgarten, M., Büchner, A. G., and Hughes, J. G. (2003). Tree growth based episode mining without candidate generation. In *IC-AI*, pages 108–114.

[6] Bettini, C., Wang, X. S., and Jajodia, S. (1998). Mining temporal relationships with multiple granularities in time sequences. *IEEE Data Eng. Bull.*, 21(1):32–38.

[7] Chakravarthy, S. and Mishra, D. (1994). Snoop: An expressive event specification language for active databases. *Data & Knowledge Engineering*, 14(1):1–26.

[8] Chen, J. and Kumar, R. (2014). Pattern mining for predicting critical events from sequential event data log. In *WODES*, pages 1–6.

[9] Connor, J. T., Martin, R. D., and Atlas, L. E. (1994). Recurrent neural networks and robust time series prediction. *Neural Networks, IEEE Transactions on*, 5(2):240–254.

[10] Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Multiple classifier systems*, pages 1–15. Springer.

[11] Eckert, M. and Bry, F. (2009). Complex event processing (cep). *Informatik-Spektrum*, 32(2):163–167.

[12] Frank, R. J., Davey, N., and Hunt, S. P. (2001). Time series prediction and neural networks. *Journal of intelligent and robotic systems*, 31(1-3):91–103.

[13] Gaber, M. M., Zaslavsky, A., and Krishnaswamy, S. (2005). Mining data streams: a review. *ACM Sigmod Record*, 34(2):18–26.

[14] Giannella, C., Han, J., Pei, J., Yan, X., and Yu, P. S. (2003). Mining frequent patterns in data streams at multiple time granularities. *Next generation data mining*, 212:191–212.

[15] Han, J., Cheng, H., Xin, D., and Yan, X. (2007). Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86.

[16] Hasan, A., Teymourian, K., and Paschke, A. (2015). Probabilistic event pattern discovery. In *Rule Technologies: Foundations, Tools, and Applications*, pages 241–257. Springer.

[17] Januzaj, E., Kriegel, H.-P., and Pfeifle, M. (2004). Dbdc: Density based distributed clustering. In *Advances in Database Technology-EDBT 2004*, pages 88–105. Springer.

[18] Kayacan, E., Ulutas, B., and Kaynak, O. (2010). Grey system theory-based models in time series prediction. *Expert systems with applications*, 37(2):1784–1789.

[19] Kotsiantis, S. B., Zaharakis, I., and Pintelas, P. (2007). Supervised machine learning: A review of classification techniques.

[20] Kumar, M. and Thenmozhi, M. (2006). Forecasting stock index movement: A comparison of support vector machines and random forest. In *Indian Institute of Capital Markets 9th Capital Markets Conference Paper*.

[21] Laxman, S. (2006). *Discovering frequent episodes: fast algorithms, connections with HMMs and generalizations*. PhD thesis, Indian Institute of Science Bangalore.

[22] Laxman, S., Sastry, P., and Unnikrishnan, K. (2007). A fast algorithm for finding frequent episodes in event streams. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 410–419. ACM.

[23] Lazarevic, A. and Obradovic, Z. (2001). The distributed boosting algorithm. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 311–316. ACM.

[24] Liaw, A. and Wiener, M. (2002). Classification and regression by randomforest. *R news*, 2(3):18–22.

[25] Lin, K. W., Chung, S.-H., and Lin, C.-C. (2015). A fast and distributed algorithm for mining frequent patterns in congested networks. *Computing*, pages 1–22.

[26] Luckham, D. and Schulte, R. (2011). Epts event processing glossary v2. 0. *Event Processing Technical Society*.

[27] Ma, B. L. W. H. Y. (1998). Integrating classification and association rule mining. In *Proceedings of the fourth international conference on knowledge discovery and data mining*.

[28] Mahfoud, S. and Mani, G. (1996). Financial forecasting using genetic algorithms. *Applied Artificial Intelligence*, 10(6):543–566.

[29] Manku, G. S. and Motwani, R. (2002). Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357. VLDB Endowment.

[30] Mannila, H., Toivonen, H., and Verkamo, A. I. (1995). Discovering frequent episodes in sequences extended abstract. In *Proceedings the first Conference on Knowledge Discovery and Data Mining*, pages 210–215.

[31] Mannila, H., Toivonen, H., and Verkamo, A. I. (1997). Discovery of frequent episodes in event sequences. *Data mining and knowledge discovery*, 1(3):259–289.

[32] Martinetz, T. M., Berkovich, S. G., and Schulten, K. J. (1993). Neural-gas' network for vector quantization and its application to time-series prediction. *Neural Networks, IEEE Transactions on*, 4(4):558–569.

[33] Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G., Olston, C., Rosenstein, J., and Varma, R. (2003). Query processing, resource management, and approximation in a data stream management system. CIDR.

[34] Papadimitriou, S., Brockwell, A., and Faloutsos, C. (2003). Adaptive, hands-off stream mining. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 560–571. VLDB Endowment.

[35] Priyadharshini, V. and Malathi, A. (2016). Analysis of process mining model for software reliability dataset using hmm. *Indian Journal of Science and Technology*, 9(4).

[36] Slimani, T. and Lazzez, A. (2013). Sequential mining: Patterns and algorithms analysis. *arXiv preprint arXiv:1311.0350*.

[37] Sureka, A. (2015). Kernel based sequential data anomaly detection in business process event logs. *arXiv preprint arXiv:1507.01168*.

[38] Vaarandi, R. and Pihelgas, M. (2015). Logcluster-a data clustering and pattern mining algorithm for event logs. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 1–7. IEEE.

[39] Van Gestel, T., Suykens, J. A., Baestaens, D.-E., Lambrechts, A., Lanckriet, G., Vandaele, B., De Moor, B., and Vandewalle, J. (2001). Financial time series prediction using least squares support vector machines within the evidence framework. *Neural Networks, IEEE Transactions on*, 12(4):809–821.

[40] Wu, E., Diao, Y., and Rizvi, S. (2006). High-performance complex event processing over streams. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 407–418. ACM.

[41] Zhao, Q. and Bhowmick, S. S. (2003). Sequential pattern mining: A survey. *ITechnical Report CAIS Nayang Technological University Singapore*, pages 1–26.

[42] Zhou, W., Liu, H., and Cheng, H. (2010). Mining closed episodes from event sequences efficiently. In *Advances in Knowledge Discovery and Data Mining*, pages 310–318. Springer.

[43] Zimmermann, A. (2012). Generating diverse realistic data sets for episode mining. In *Data Mining Workshops (ICDMW), 2012 IEEE 12th International Conference on*, pages 611–618. IEEE.