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

September 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
September 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 is relevant. 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 episodes can be used to

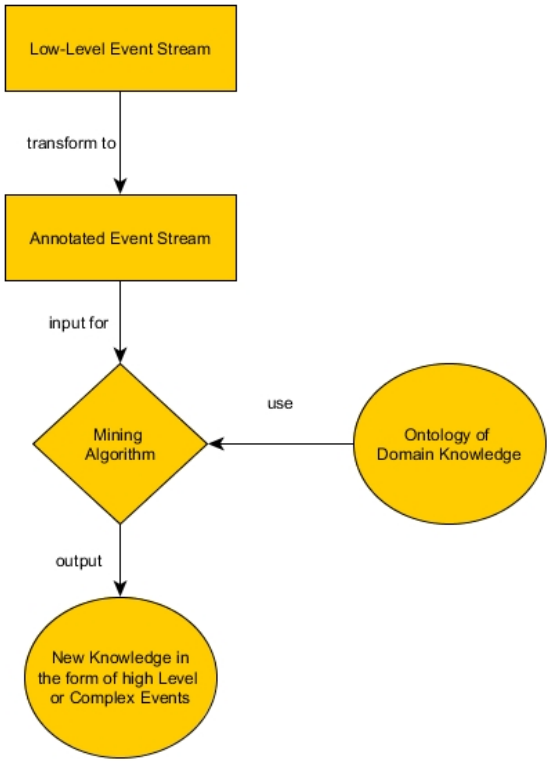

Fig. 1.1 General structure of a semantic mining process of complex events

preemptively expect failures and react accordingly). The domain that will be used in the evaluation of this thesis is stock market prediction. Both predicting the overall direction of the stock market and predicting whether individual stocks will rise or fall are a difficult problems that have 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 introduces the basic terminology, reviews the related work, and gives a detailed, formal introduction to episode mining. Chapter **??** presents the suggested algorithm to mine predictive episodes from event streams in detail

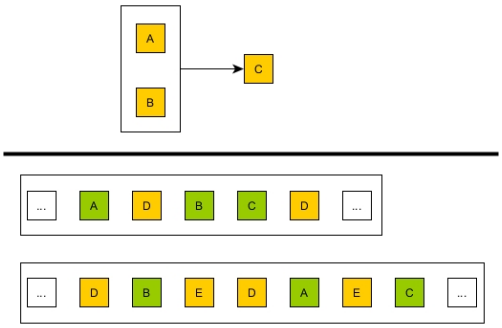

Fig. 1.2 The top half visualizes an example episode pattern, which consist of a conjunction (A and B must both occur, but the order does not matter) and a sequence (C must occur after A and B). The bottom half shows two windows of an example stream, in which occurrences of the episode are shown in green.

and analyzes its complexity. Subsequently chapter 5 presents an evaluation using both synthetically generated data, as well as real stock market data. Finally chapter 6 concludes the paper and mentions possible future work.

1.2 Problem Definition and Exact Research Question

Even though many terms have not yet been precisely defined (which will be done in chapter 2) this section presents a loose, intuitive definition of the problem to be tackled by this thesis. Given a stream of basic events and a special event type P one is interested to predict occurrences of that special event type P in the stream, so that actions can be taken before P actually occurs. For this purpose we are interested to discover episodes in the stream of which occurrences are likely to be followed by P. A window of an example stream is visualized in figure 1.3.

Fig. 1.3 The figure visualizes an event stream in which events of type P are to be predicted. The occurrences of the possible predictive episode A followed by E is colored in teal, whereas events of types P are colored in green and all other events are yellow

Note that it is very important to the mining process how much time is allowed to pass between the occurrence of the predicting episode and the event that is to predict. If this time

span is too small a mining algorithm might not find reliable predictors. If the timespan is too large a mining algorithm might be overwhelmed by the number of candidates that need to be considered, since the more time may pass the larger are the sequences before each event P that must be considered.
To solve the problem outlined above a semantic mining algorithm will be suggested in this thesis. Since the underlying data structure to be analyzed are data streams, the algorithm must have the following properties:

- The algorithm must be able to adapt to a changing context (as the stream progresses, the underlying model may change completely)
- The recognition of predicting episodes must be quick, since streams, especially in the domain of stock markets, can have a high velocity. This makes it 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 action.
- 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 data-sets. 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

This chapter reviews the related work relevant for this thesis. 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. First section 2.1 introduces and defines basic the basic terminology that will be used in this thesis. Subsequently section 2.2 gives a broad overview over the general area of mining data streams but lays a focus on the mining patterns as well as prediction algorithms. Section 2.3 then provides more details about prediction in the domain of financial data. Finally section 2.4 introduces the concept of episodes and summarizes different episode mining approaches in static databases. TODO: semantic web

2.1 Basic Definitions and Terminology

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

2.1.1 Event Processing Terminology

The basic event terminology in this subsection is paraphrased from the event processing glossary created by the Event Processing Technical Society [23]. 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.*

The event processing society claims that the context usually solves the ambiguity in the above definition. Since this may not always be the case we will only use the term *event* in this thesis to refer to event objects unless it is otherwise specified.

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 (see definitions 3 and 4). A simple event however is the most basic form of an event and often the ingredient for the creation of more complex events: Given simple events it is possible to derive events from those or the absence of those. For example the absence measurement events of a sensor could be used to derive the event of that very same sensor 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 section 2.4), 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 explicitly 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 type alphabet, often simply called 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. 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. Also other properties for example that the stream is constantly updating (new events coming in) are not considered yet.

2.2 Data Stream Mining

As already mentioned in the introduction data streams present a challenge to data miners in many ways. This section aims to give both a general overview over the broad research area of mining data stream as well as specifically cover the related topics of forecasting and stock market prediction.

2.2.1 Data Stream Mining Taxonomy

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 should be incremental. Normally 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. [16]. Note that the paper by Gaber et. al. was published in 2005, so quite a lot of work has been done since then, which means that these papers are obviously not present in their literature review. However the overview provided by the authors is still very useful, since it brings structure to the large field of data stream mining. A taxonomy that follows the basic structure of the paper by Gaber et. al. is visualized in figure 2.1.

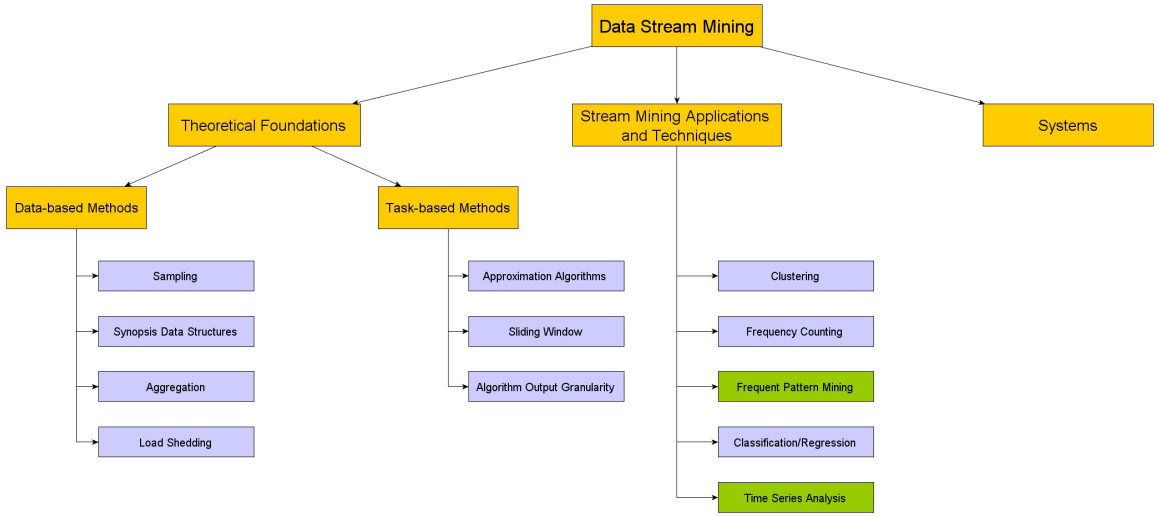

Fig. 2.1 A taxonomy of the research areas of data stream mining. The subcategories that are marked green are the categories to which this thesis will make contributions.

As can be seen in the figure the state of the art in data stream mining can be roughly divided into three basic parts:

1. Theoretical Foundations
2. Mining Techniques and Applications
3. Systems

The theoretical foundations contain general approaches on how to deal with the issues of data streams. There is a distinction between data-based techniques and task-based techniques. Data-based techniques aim to reduce the data being processed or apply transformations on the data stream, while task-based techniques are modifications of existing techniques to meet the requirements for time and space. The subcategories of data-based techniques are:

- **Sampling:** The idea of sampling is basically the same as in statistics. Instead of processing the whole stream, only a subset of the stream is looked at. Due to the

unknown size of the stream, sampling methods are more complex than in static systems like databases [26].

- **Load Shedding:** Load shedding is similar to sampling in that it simply drops certain incoming data without processing it, however load shedding is a dynamic approach that is only applied if needed, for example if there are volume spikes (large amounts of data coming in suddenly). Thus in contrast to sampling, which is applied over the whole lifetime of the stream, load shedding is a more reactionary strategy [8].
- **Synopsis Data Structures:** Synopsis data structures summarize the stream in data structures that use less memory than the stream itsself. These data structures are then used to approximately answer queries. A specific subcategory are sketeches, which use very little memory when compared to the whole stream. Examples of this are the so-called frequency moments [7]
- **Aggregation:** Aggregation is mainly useful when statistical measures are to be computed over a stream [38].

The task-based techniques mentioned by the authors are:

- **Approximation Algorithms:** Approximation algorithms are common for hard problems (such as NP-complete problems) but also many classic data-mining problems can be solved on data streams using approximate variants of the original algorithms. A commonly cited example are approxiamtion algorithms for the mining frequent items or itemsets, such as the sticky sampling or the lossy counting algorithm [25].
- **Sliding Window:** The usage of sliding windows over the data streams is common when the user is only interested in the most recent events as opposed to the complete history. The main challenge here is to construct algorithms that work with the incremental updates that happen whenever new data arrives (the window slides forward). Windows can be defined by either the number of observations in it (sequence-based window definition) or by the duration (timestamp-based window definition) [17].
- **Algorithm Output Granularity:** The term algorithm output granularity refers to the strategy of dynamically reacting to the available memory and fluctuating data rates. The basic idea is to mine the incoming data stream as long as possible (normally until the device runs out of memory). If this happens the generated knowledge structures are merged and summarized in order to free up memory and continue the mining.

The second category contains the actual mining techniques and applications, which are well known from classical data mining:

- **Clustering** There are many different approaches that try to apply clustering algorithms to the data stream scenario. Some authors introduce novel methods [2] [3], while others modify existing clustering algorithms [18].
- **Classification and Regression** In Addition to time and memory constraints classification and regression in evolving data streams presents the extra challenges of concept drift, which means that the underlying class distribution may change, which will make the originally built model invalid over time. A commonly used approach to the time and memory constraints are the so-called very fast decision trees [12] which can be built incrementally and require constant time and memory per example.
- **Frequency Counting** Frequency counting is mainly used in conjunction with pattern mining. It is a little unclear why Gaber et. al. decided to name the category frequency counting instead of pattern mining, since all examples that they mention in the frequency counting section of their review count frequencies of items or itemsets, which fit into the category of pattern mining. A reason could be that there exist approaches to counting frequencies that could be generalized to many different pattern mining problems.
- **Pattern Mining** As already mentioned, Gaber et. al. do not explicitly mention pattern mining as a category, however since it is a large field of work and is relevant to this thesis it definately needs to be mentioned. In fact pattern mining in data streams will be looked upon in subsection 2.2.2 in detail.
- **Time Series Analysis** Just like pattern mining, time series analysis is especially relevant to this thesis, since predicting (forecasting) future developments based on past data is a large research area in time series analysis. Thus time series analysis in data streams is discussed in detail in subsection 2.2.3.

The last category in the taxonomy is less focused on research issues and conceptual or algorithmic problems but instead lists the existing data stream processing systems (to that date). Since that is less relevant to this thesis we do not mention or describe existing systems but instead refer the interested reader to the original paper by Gaber et. al. [16].
As visualized in figure 2.1 this thesis mainly deals with the subtopics of time series data and pattern mining, which is why these two areas of work will now be inspected more closely in the next subsections

2.2.2 Pattern Mining in Data Streams

TODO: is it actually necessary to look at this - the really relevant stuff is done in the episode mining section

2.2.3 Time Series Analysis in Data Streams

Before diving deeper into the topic of time series analysis, it is important to distinguish time series from data streams. Both concepts are similar and also have overlapping research areas. In this thesis we speak of a time series if we refer to a temporally ordered sequence of data points, whose time values are sampled at a fixed time interval (QUESTION: is that correct?). Usually time series contain numerical values, examples of such data are:

- values of a stock market index over a trading day
- measurement values sampled from continous readings of a temperature sensor
- electrocardiography readings of a human heart

Data streams are also ordered sequences of data. The important distinctions between time series and data streams are:

- Data streams continuously have new data points coming in, thus are continuously growing. This does not have to be the case for time series data. A database that records electrocardiography readings of different patients still contains time series data, however these are not data streams, since the readings are finished and no longer updating.
- Data Streams do not have to be sampled at the same time interval, in fact varying time delay between data points is very common here.
- In contrast to time series it is common in data streams to have streams of categorical values or events, whereas values of time series are usually numeric.

The combination of both, a time series data stream, is a time series that is constantly updating, for example an electrocardiography reading that is currently taking place, or stock values that are being recorded over a day. In these cases the time series must be processed online.
At the first glance this thesis does not seem to be located in the area of time series analysis, since it deals with the mining of complex events. However, since this thesis aims to create a

novel method in order to build predictive models using complex events. Predictive models however are commonly build for time series data, since predicting time series (also called forecasting) is of interest in many different domains. Since any novel method needs to be compared to the state of the art it is important to at least give a brief overview, which is what is done here.
Most other research areas in data mining such as clustering or classification are rather focussed in their goals (find an appropriate clustering or build an acurate classifier). In contrast to this the research area of time series analysis is more broad, meaning there are a variety of different objectives that can be of interest when analyzing time series. Thus it is helpful to first get an overview over the most common objectives and techniques in a similar way as the taxonomy for data stream mining visualized in figure 2.1. The main source for this subsection is the book by J. Gama [17]. Figure 2.2 visualizes the different areas of interest in time series analysis in data streams mentioned by the author.

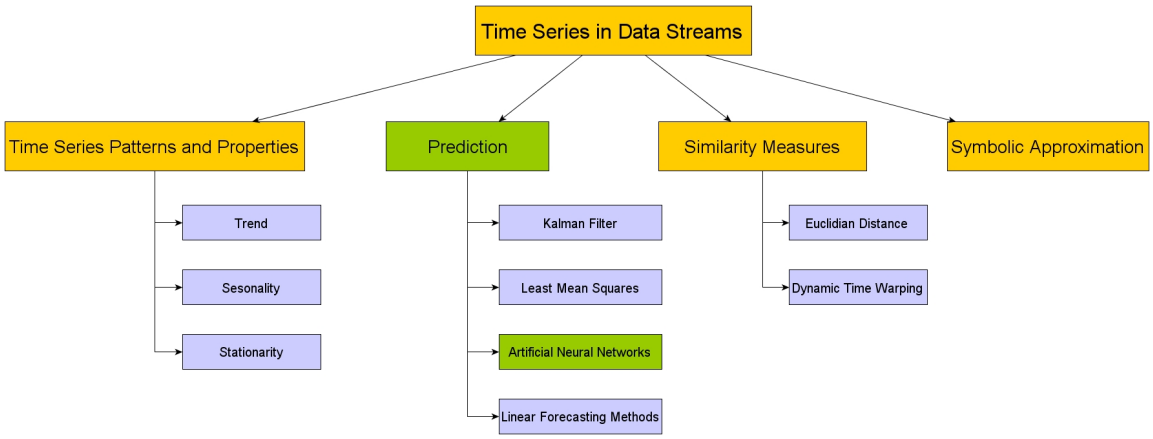

Fig. 2.2 An overview over the different research areas in time series analysis in data streams. The subcategories that are marked green are the categories, which are of interest in thesis and thus will be looked at more closely.

Note that the research area of time series analysis is large and therefore there are important subareas that are not mentioned by the author. Examples of such research areas are curve fitting, function approximation and time series segmentation (QUESTION: Do I need citations for these?). We briefly go over the different areas that are mentioned by J. Game but are not particularly relevant to this thesis:

- **Time Series Patterns and Properties:** A lot of time series can be categorized according to their behavior and can be said to have certain properties. Among these are for example long term trends, seasonality (cyclic behavior) and stationarity (mean, variance and autocorrelation are constant).

- **Time Series Similarity Measures:** Similarity of time series has many applications and can for example be used in k-NN classification. There are different similarity measures, which each have their advantages and disadvantages. Two well known examples are the classic euclidean distance and the dynamic time warping algorithm.
- **Symbolic Approximation:** Symbolic Approximation is a technique that aims to discretize time series into a string of arbitrary length. It can be of interest if one wants to apply algorithms that work on strings or categorical data to time series.

As already mentioned time series prediction (also called forecasting) is relevant for this thesis due to the empirical evaluation which uses financial time series data. Thus, instead of giving a detailed overview over the general topic of time series prediction, we remain rather broad in this subsection and instead we devote the next subsection to exclusively report the state of the art on the prediction of financial time series.
Very broadly forecasting methods can be divided into linear and non-linear methods. Linear models are usually simpler, but are at a disadvantage when the underlying model is non-linear [39]. Non-linear methods, such as neural networks are more powerful, in fact it has been shown that neural networks can in theory model any non-linear function [1] [15]. However, building and training an actual neural network is a difficult task, since multiple design choices (such as the number of hidden neurons, the activation function and the initial weights) need to be made, which usually requires expert knowledge of both the underlying domain as well as neural networks in general in order to train an appropriate network [1]. Researchers have also tried to combine both approaches in order to form hybrid methods [39].
Neural networks were originally conceived as batch methods, meaning there were used in an offline scenario with no new data coming in constantly. However adapting them to the stream environment is surprisingly simple in most cases and has been done on multiple occasions [11] [14]. In fact, the streaming environment can be beneficial to neural network training, since training a neural network in a static environment usually means making multiple passes over the training data, due to the lack of training data. If done incorrectly this can result in overlearning the training data and thus poor generalization. In the streaming environment however, there is an abundance of data, which means that each example has to be processed only once [17].
Predicting or forecasting time series values is normally very domain specific which results in different domains having their own specialized forecasting methods or specific modifications of popular general predictive models. Some of the domains in which time series forecasting is relevant are:

- Forecasting price developments in stock markets (see section 2.3 for a detailed review of the state of the art)
- Forecasting the electricity demands of households [34]
- Forecasting the power output of solar energy plants [19]

TODO: a few more sentences about forecasting in different domains and why...
QUESTION: maybe explain neural networks ?

2.3 Stock Market Forecasting

When reviewing the related work for the forecasting of stock markets, it is important to note that in a lot of cases, financial time series are not analyzed in a streaming environment. Authors commonly attempt to forecast daily closing values of stock markets, which means that in this case data velocity is very low (one new data point per day). However many techniques applied in these scenarios can also be applied in more rapidly moving streams. Examples of these are autoregressive models [32] or artificial neural networks [17].
A good starting point into the prediction of stock market movements is provided by a literature study by Atsalakis et. al. [6]. The authors review more than 100 papers that attempt to predict stocks or stock indices. Figure 2.3 visualizes some of the different properties and experimental settings of the approaches covered by Atsalakis et. al.

Most of the visualized information in figure 2.3 was taken from the previously mentioned literature study by Atsalakis et. al. [6]. However some pieces of work were not covered in that study, for example a comparative study of different models including the otherwise rarely used random forest [20]. We briefly review the different properties and categories visualized in figure 2.3:

- **Target:** The target that is to predict can in theory be any financial index, indicator or price of interst. Many researchers tried to forecast the movement of stock indices [40],[33], [20] . However, also individual stocks have received attention [24].
- **Performance Measures:** When building models it is crucial to measure their performance to enable comparison to other models. The number of different performance measures in this case is surprisngly large and diverse. The employed measures range from economic measures, such as the annual rate of return or the Hit-Rate to more technical measures such as the average percentage error or the mean squared error to only name a few. It is impractical to visualize or enumerate all measures that are in

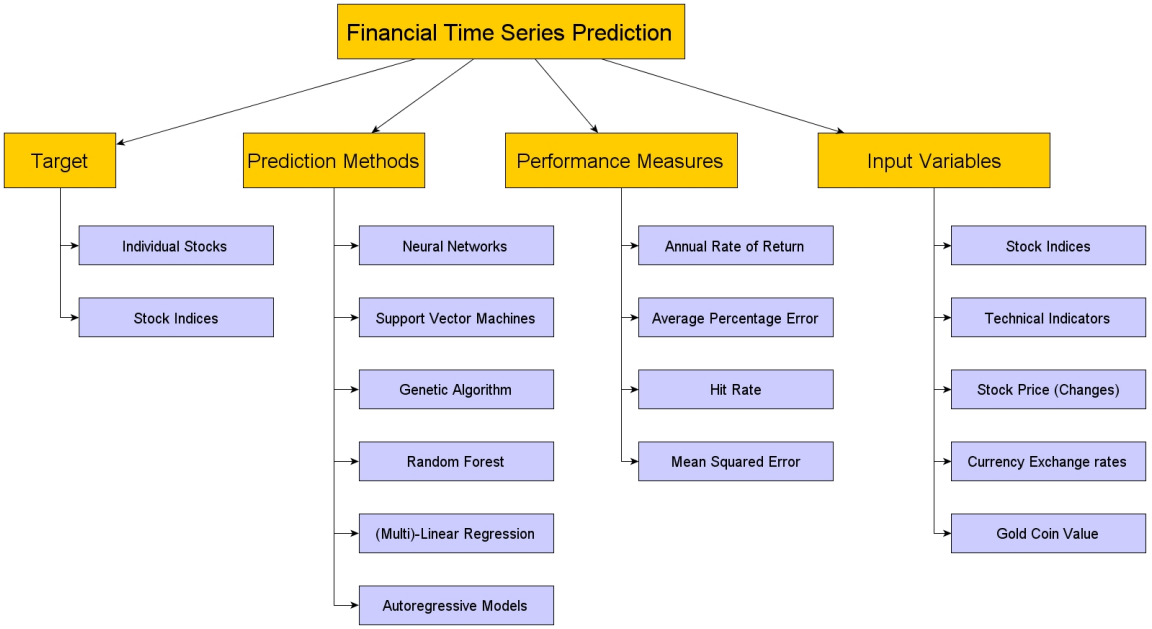

Fig. 2.3 An overview over different properties, experimental settings and approaches to the forecasting of of stock markets

use, thus we limit ourselves to the previous examples. For a comprehensive list we refer the reader to the literature study by Atsalakis et. al. [6].

- **Input Variables:** In real world scenarios this is probably the most important category. No matter which kind of elaborate model is built, if the choice of the input variables is poor, meaning they contain little information, then the model built from those simply can not perform well. This is especially true for financial time series, since those are known to be chaotic and noisy [40]. In fact there are researchers that argue that financial time series follow the principle of random walks which would imply that accurately, meaning better than random, forecasting financial time series based on historical data is impossible. [13]. However there is a considerable amount of papers that suggest otherwise, since accurate prediction results have been achieved by multiple authors based on multiple different forecasting methods (see the literature study by Atsalakis et. al. for examples [6]). Input variables for stock market prediction are almost always past stock data. Sometimes other financial indicators are used as well. Examples include different stock indices, gold price or currency exchange rates.

- **Prediction Methods:** While figure 2.3 mentions many different approaches to the forecasting of stock markets, a large majority of the published papers in this area uses

some form of neural network. In fact Atsalakis et. al. note that 60% of the papers they surveyed use feed forward Neural Networks and recurrent networks [6].

TODO: formulate a nice conclusion

2.4 Episodes

As already mentioned in the introduction, this thesis deals with a specific type of complex events called episodes. Despite being a rather specialized area of research there exists quite a bit of related work that deals with episode mining. What is notable is that there are some discrepancies in terminology. Different authors sometimes use different terms to refer to the same concept or use the same term but with a different meaning. These discrepancies will be mentioned here and the exact definitions for this paper will be mentioned in chapter 3.
Before diving into the previous work on episodes it is important to have a rough idea of what kind of patterns episodes are. Thus we provide a short and informal explanation here, whereas section 3.1 will give a much more detailed and formal definition of all concepts revolving around episodes.
Most simply put, episode patterns are partially ordered sequences of events. They can be visualized as directed acyclic graphs like the example episode shown in figure 2.4.

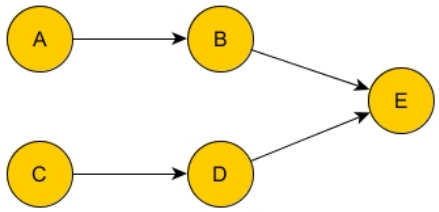

Fig. 2.4 An example episode pattern visualized as a directed acyclic graph. This example pattern specifies that event A and event C may occur in any order, however A must come before B and C must come before D, E must occur last.

Episodes are usually mined from a very large sequence. This distinguishes the concept of episode mining from the concept of sequential pattern mining, which takes place on a sequential database, in which there are many records and each record is a sequence of events [37]. The research concerning episodes can be organized in basic categories as visualized in figure 2.5.

The different types of episodes that have been looked at in the literature are mostly special cases of general episodes. Mannila et. al. introduced the three concepts of serial, parallel and composite episodes [27]. According to their definitions serial episodes are episodes

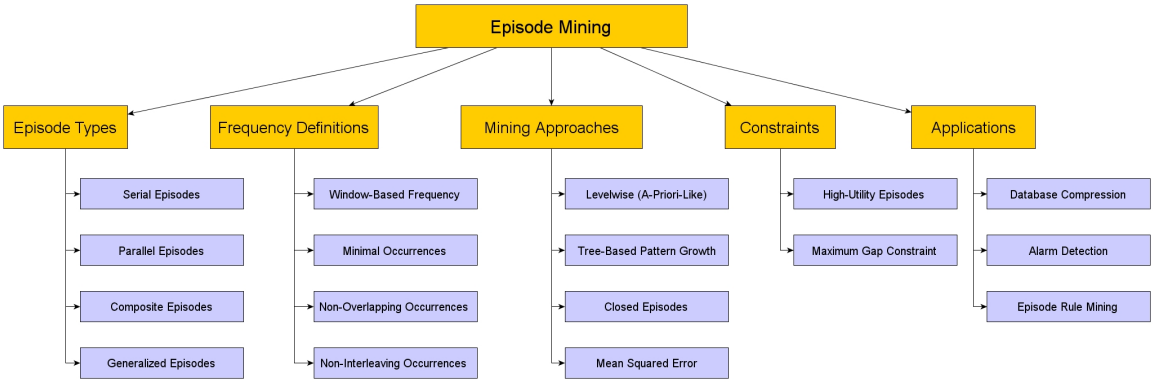

Fig. 2.5 A rough categorization of the existing research in episode mining

that have a total ordering (essentially sequences), parallel episodes are episodes without any order (essentially multisets) and composite episodes are episodes that have some order imposed on the events, but do not need to have a total ordering (like serial episodes do). There are however altering definitions of composite episodes in the literature, for example both Bathoorn et. al. [9] and Baumgarten et. al. [10] deviate from the original definition and redefine composite episodes as sequences of sets. in their works. Section 3.1.1 explains the difference between the two definitions in more detail. Extended episodes have been investigated by S. Laxman in his PHD thesis [21]. The main difference between classic episodes and extended episodes is that classic episodes assume that the basic events are instantaneous, whereas extended episodes can be mined from events that have a duration. It is notable that most work focuses on serial and parallel episodes [27] [28] [21] [22]. Authors have already identified this gap in the research and come up with two different reasons:

1. The problem of frequent pattern explosion is already significant for serial and parallel episodes, but still much worse for composite episodes [9].

2. Detection of composite episodes (checking whether a given composite episode occurs in a sequence) is NP-complete, since 3-SAT can be reduced to this problem [31].

When mining frequent episodes from sequences, frequency of episodes can be defined in multiple ways. Since the different variants of episode frequency need to be carefully considered in this thesis we do not give a brief overview here but instead devote subsections of chapter 3 to discuss the different frequency definitions. Specifically subsection 3.1.3 gives a detailed explanation of the window based frequency, whereas subsection 3.1.4 provides an overview over the other frequency definitions that have been used in the literature.
The different mining approaches for episodes are similar to well known approaches for frequent itemset mining. Since most episode frequency definitions follow the apriori principle,

a level-wise approach like in frequent itemset mining [4] is suggested by many authors [27] [21]. As an alternative, tree growth methods have been proposed, in which candidate episodes are represented in a tree data structure that gets grown as the algorithm progresses [10]. Since frequent pattern explosion is an issue when mining frequent episodes it is unsurprising that the concept of closed patterns from classical pattern mining [36] has been adapted to episodes [41] [31].
When mining episodes, authors have considered several constraints or specific scenarios. Usually one is interested to mine episodes that are local, meaning the events of the episode are supposed to happen close to each other (time-wise). The maximum duration of an episode can be restrictd in different ways, for example by specifying a window size [27], when employing the window-based frequency or by a maximum gap constraint, which restricts the maximum time difference between two events of an episode [30]. Another special scenario that considers external constraints is the mining of high-utility episodes, in which one is not interested in frequent episodes but instead into those episodes that cover events that have a high utility score (which is given externally) [37].
The applications for episode mining are diverse and some of them are mentioned in the following.

- Mannila et. al. (arguably the researchers to make episode mining a popular task) were motivated to mine episodes in order to analyze alarms in telecommunication systems [28].
- Several authors attempt to describe, summarize or compress databases or large sequences by mining appropriate episodes. Examples of research in this application area include the work by Bathoorn et. al. [9] as well as the work by Vreeken et. al. [35].
- Episode mining has been used to extract rules from sequences in several different domains, such as geophysics (finding dependencies between earthquakes) [30] or health care (analysis of temoral dependencies between risk factors for atherosclerosis) [29].

2.5 Semantic Web

TODO

Chapter 3

Background and Problem Definition

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 Episode Mining Background

While episodes were already intuitively defined and talked about in the previous chapters it is important to formally define episodes, since formal definitions are clear and unambiguous. This is important in order to define and explain mining objectives and techniques as well as algorithms that are used in this thesis. The rest of this section is divided as follows:

3.1.1 Basic Definitions

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 episode (also sometimes called episode pattern or elementary episode)* α *of length m (also called m-episode) is defined as a triple:* $\alpha = (V_\alpha, \leq_\alpha, g_\alpha)$ *where* $V_\alpha = \{v_1, ..., v_m\}$ *is a set of nodes,* $\leq_\alpha$ *is a partial order over* V_α *and* $g_\alpha : V_\alpha \rightarrow \Sigma$ *is a mapping that maps each node of* V_α *to an event type. [27]*

Put more simply an episode is a multiset of event types, whose elements can be, but do not have to be ordered by a relation ($\leq_\alpha$). Another way of putting it is that an episode is essentially a partially ordered sequence of events. Before we look at examples there are a

two special types of episodes that need to be mentioned since they have received the most attention in the available literature. These are called serial and parallel episodes:

Definition 10 ***Serial Episode*** *An episode $\alpha = (V_\alpha, \leq_\alpha, g_\alpha)$ is called a serial episode if $\leq_\alpha$ is a total order. [27]*

Definition 11 ***Parallel Episode*** *An episode $\alpha = (V_\alpha, \leq_\alpha, g_\alpha)$ is called a parallel episode if $\leq_\alpha = \emptyset$, in other words if there is no ordering imposed on V_α at all. [27]*

Essentially, serial episodes are sequences, while parallel episodes are multisets. Figure 3.1 visualizes example episodes in the same way that we have visualized episodes in earlier chapters, namely as directed acyclic graphs (DAG).

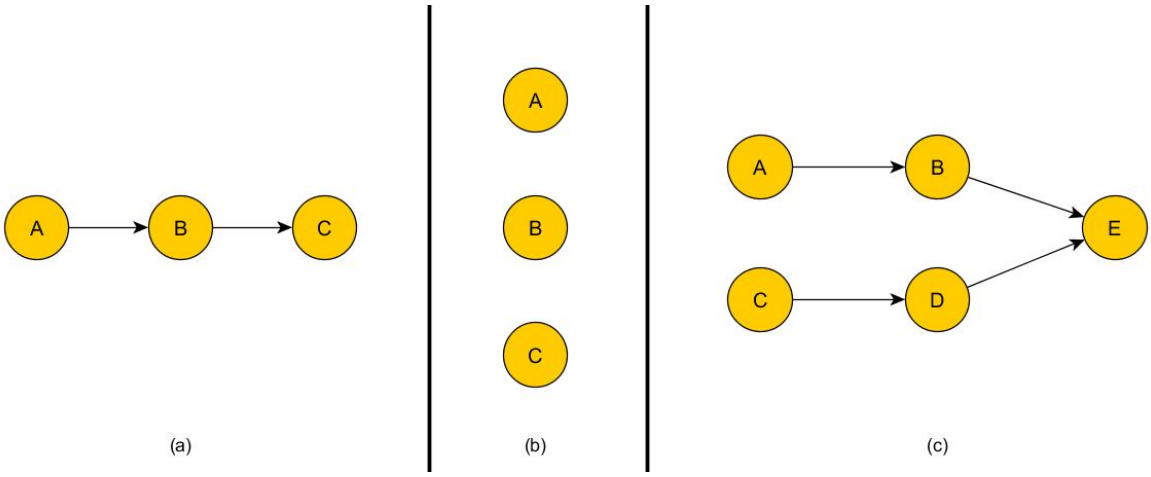

Fig. 3.1 An example of different episodes: (a) - a serial episode, (b) - a parallel episode, (c) - an elementary (composite) episode

In fact each episode can be formally transformed to a DAG using the following simple procedure: Given an Episode $\alpha = (V_\alpha, \leq_\alpha, g_\alpha)$, create the corresponding DAG $G = (V, E)$ by executing the following:

1. For each $v \in V_\alpha$ add v to V and label v with $g_\alpha(v)$
2. For each pair $v, w \in V_\alpha$ where $v \leq_\alpha w$ add edge (v, w) to E

The original paper by Manilla et.al. [27] also introduces the notion of composite episodes. We repeat the definition here:

Definition 12 ***Composite Episode*** *An episode $\alpha = (V_\alpha, \leq_\alpha, g_\alpha)$ is called a composite episode if $g_\alpha : V_\alpha \to \Sigma \cup C^*$, where C^* is the set of all composite episodes. [27]*

This recursive definition of composite episodes may be confusing at first, since it is formulated very compactly. A slightly more clear way to look at it is the following:
A composite episode is either:

- A single event (An elementary episode of size 1).
- A serial composition of two composite episodes
- A parallel composition of two composite episodes

This definition 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 (see definition 13). Note that composite episodes are not more expressive than elementary episodes as defined in definition 9, it is just a recursive way of defining episodes.
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 [9]. However they define composite episodes as a sequence of parallel episodes, which is more restrictive than the original definition. Also Baumgarten et. al. use this definition [10] when they present an approach to mine descriptive composite episodes. Note that not all elementary episodes can be represented as sequences of parallel episode. A simple example shown in figure 3.2 illustrates this. If the presented episode were to be represented as a sequence of parallel episodes obviously A and B would have to be in different parallel episodes in order to fulfill the requirement that B must be after C. After that the problem is that it is impossible to assign C to any of those sets. If it gets assigned to the same parallel episode set as A then this would prohibit A and B occurring first followed by C, which is allowed in the original definition. Likewise if C gets assigned to the same parallel episode as B then that eliminates the possibility of C occurring before A and B, which once again was allowed in the elementary episode. In this thesis we will stick to the original definition as presented in definition 12.

If we want to quickly denote simple episodes formally without drawing a DAG, 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 the previous figures
The notion of sub- and superpatterns, which is very important for most pattern mining applications also applies to episodes, as shown in the next definition.

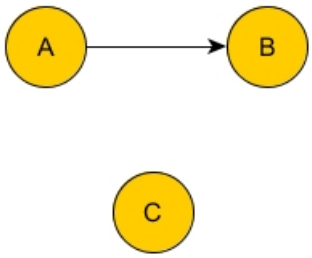

Fig. 3.2 An elementary episode that can not be represented as a sequence of parallel episodes

Definition 13 ***Subepisode*** *An episode $\beta = (V_\beta, \leq_\beta, g_\beta)$ is said to be a subepisode of episode $\alpha = (V_\alpha, \leq_\alpha, g_\alpha)$ if all of the following conditions hold:*

1. *The nodes of β are a subset of the nodes of α:*
   $V_\beta \subseteq V_\alpha$
2. *All nodes of β are assigned to the same event type as their corresponding nodes in α:*
   $\forall v \in V_\beta \ : \ g_\beta(v) = g_\alpha(v)$
3. *The ordering in α is at least as strict as the ordering in β:*
   $\forall v, w \in V_\beta, v \leq_\beta w \ : \ v \leq_\alpha w$

In this context we also refer to α as the superepisode of β. [27, 22].

It is important to note that in the original definition of sub- and superepisodes by Mannila et. al. [27], the first property of the above definition was actually defined as $V_\beta \subset V_\alpha$, which implies that subepisodes would always have to consist of at least one less node than their superepisodes. This definition was changed by Laxman et. al. [22] to allow set equality as well. The implications of this change are that parallel episodes like $A \| B$ are now subepisodes of their serial counterparts of the same length $A \rightarrow B$. In this thesis we stick to the definition that allows set equality between super- and subepisodes. In both cases however subepisodes may relax the ordering specified in their superepisodes. For example if given the serial episode $\alpha = (A \rightarrow B \rightarrow C)$, not only $A \rightarrow B$ is a subepisode of α but also $A \| B$.
So far we have talked a lot about episode patterns but we have not yet talked about the detection such episodes in data. It is helpful to think of the episode patterns we have defined above as templates for concrete occurrences. In order to define what we mean by an episode occurrence, we first need to formally introduce the notion of an event sequence.

Definition 14 ***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 $\forall i, j \in 1,...,n\ i < j \implies t_i \leq t_j$. *[28]*

Note that it is allowed for two or more consecutive elements in a sequence to have the same timestamp. This is necessary in order to allow for multiple events to occur at the same time. The ordering of events with the same timestamp is not further specified and irrelevant in most cases. Another way to define such sequences is to define them as sequences of event sets in which consecutive event sets must all have different timestamps and events that occur simultaneously are part of the same set [9]. Definitions from other authors explicitly prohibit two events happening at the same time [10]. Which definition is appropriate depends on the context, for this thesis we will stick with definition 14.
Given the notion of an event sequence we can now define episode occurrences:

Definition 15 ***Episode Occurrence*** *An event episode* $\alpha = (V_\alpha, \leq_\alpha, g_\alpha)$ *is said to occur in a sequence S if events of the types that the nodes in* V_α *are mapped to by* g_α*, 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* α *as an injective Map* $h : V_\alpha \to \{1,...,n\}$*, where* $g_\alpha(v) = T_{h(v)}$ *and* $\forall v,w \in V\alpha : v \leq_E w \implies t_{h(v)} \leq t_{h(w)}$ *holds. [27]*

A classic episode pattern as defined in (TODO) in itself does not specify any time span in which the events of the episode must occur, however if given an episode occurrence we can define the duration of that occurrence:

Definition 16 ***Episode Occurrence Duration*** *The duration of an occurrence is defined as the difference between the timestamps of the last and first event that the episode nodes are mapped to. TODO: formalize and sources*

An example of an episode pattern, a sequence and two occurrences is visualized in figure 3.3.

3.1.2 Episode Discovery 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 [5]. A general algorithm for mining the episodes occurring frequently in a sequence is given in algorithm 1. The algorithm is very alike the basic apriori algorithms, since it uses

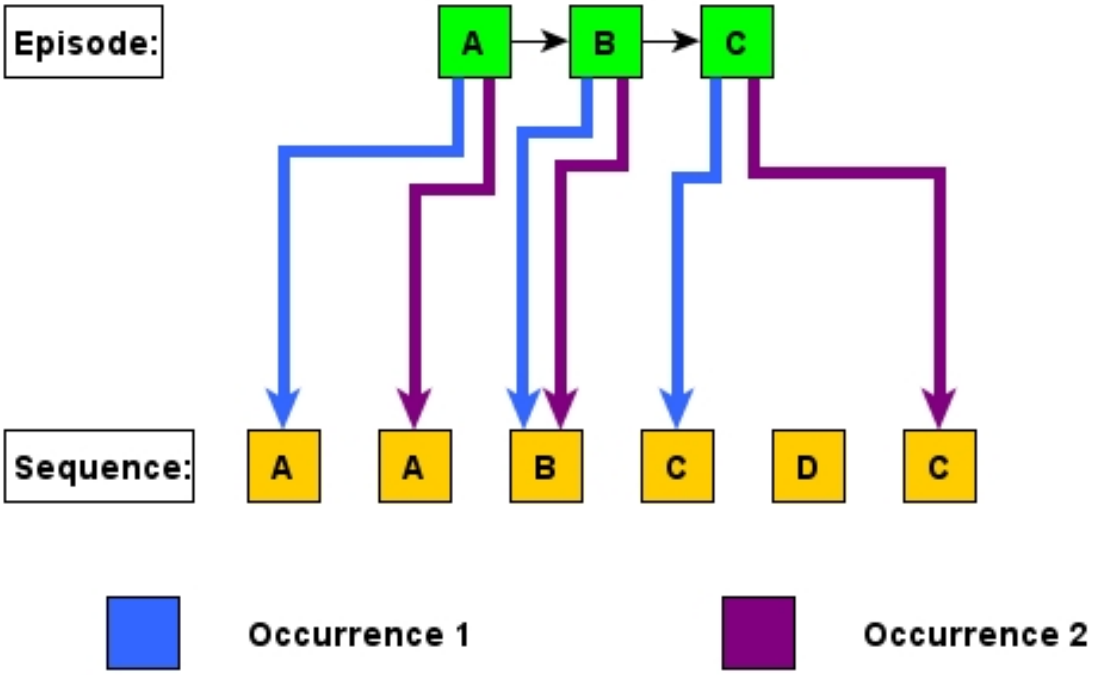

Fig. 3.3 A visualization of the concept of episode occurrences in an event sequence. The episode pattern is drawn in green, the sequence in yellow. An occurrence is defined as a mapping from the nodes of the pattern to events in the sequence (see definition 16). Thus the occurrences are visualized as arrows of different colors (blue and purple).

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 [5], which formally means that if an episode is frequent, all its subepisodes must be frequent as well. Intuitively one would assume that this is always 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, since this would eliminate the possibility of an efficient candidate generation that prunes those episodes which have infrequent subepisodes. To the best of our knowledge all frequency definitions proposed in the literature satisfy the apriori principle.

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
```

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

- 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 candidate episodes

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 potential overlappings. Naturally the number of possible parallel and composite episode occurrences is even larger, since they are less restrictive in the order of the events. Additionally, such a frequency definition would violate the apriori principle, since subepisodes can have less occurrences than their superepisodes. Consider the example sequence $[A,B,C,A,B,C]$ (timestamp values are left out). In this sequence there are 3 distinct occurences for the episode $A \rightarrow B$ and

four distinct occurrences for its superepisode $A \rightarrow B \rightarrow C$. These detrimental effects of this naive frequency definition are nicely summarized by Laxman et. al. in a paper presenting the non-overlapped frequency definition [22].
This leads to various frequency definitions of episodes in the literature, which will be dealt with in subsections 3.1.3 and 3.1.4. Each frequency definition comes with its own frequency counting algorithm. The procedure for generating candidates is independent of the frequency definition and is presented in 3.1.5.

3.1.3 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. [27], although the frequency counting algorithms were only mentioned in text form. The same authors specified the algorithms in a later paper [28], 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 17 ***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.*

An example of how windows of a fixed size are located in a sequence of events is presented in figure 3.4.

Definition 18 ***Episode Frequency - Window based Definition*** *Given a sequence of events S, a fixed window size of w and an episode* α*, we define the window based frequency* $w\_freq(\alpha)$ *as the number of windows W with size w of S in which* α *occurs:* $w\_freq(\alpha) = |\{W(S,q,r) \mid r-q+1 = w \wedge \alpha \text{ occurs in } W\}|$.

For example given the sequence visualized in figure 3.4 $A \rightarrow B \rightarrow C$ occurs in windows $W(S,1,5)$ and $W(S,3,7)$.
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. In the previously mentioned example, we can find the episode $C \rightarrow D$ in the consecutive windows $W(S,1,5)$, $W(S,2,6)$ and $W(S,3,7)$, which means we will get a frequency of 3 just for the two events $(3,C)$ and $(4,D)$. This effect obviously increases with

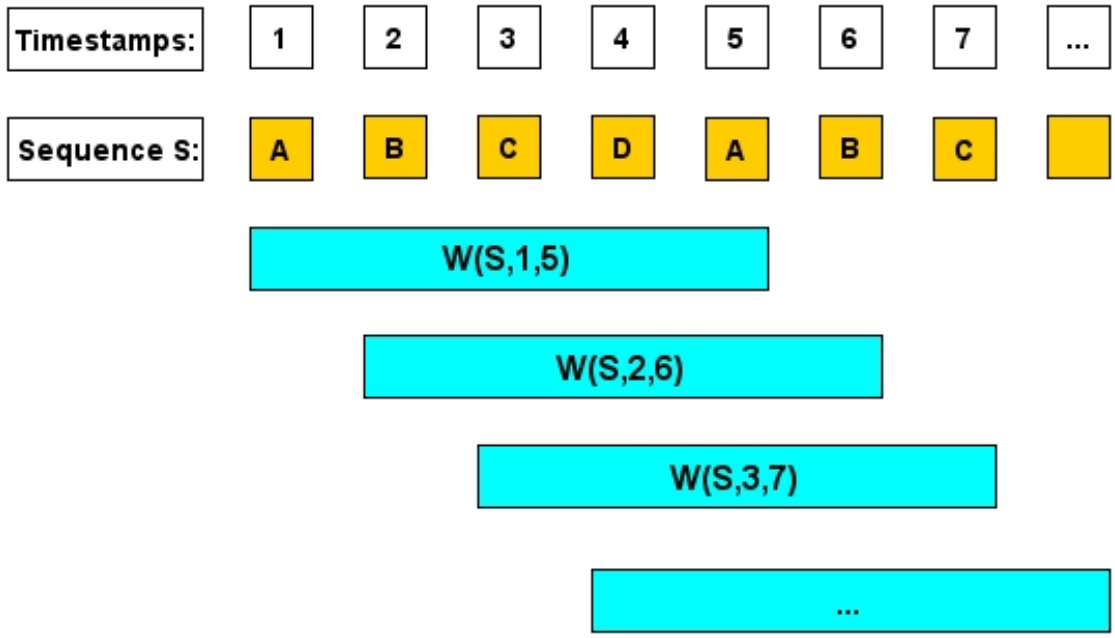

Fig. 3.4 Different time windows of size 5 in an event sequence.

the window size. Note that for each episode α we only count one occurrence per window W, no matter how many occurrences of α 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 there is a better approach, which makes it 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 can be looked up in the appendix A.1. There is a notable absence of frequency counting algorithms for elementary (composite) episodes in literature. Mannila et. al. claim that each composite episode can be broken down into partial episodes, which are serial and/or parallel [28]. However they neither specify an algorithm for breaking down composite episodes into purely serial and parallel parts, nor do they specify a frequency counting algorithm for composite episodes. Subsequent research, such as alternate frequency definitions and counting algorithms has also mainly focused on parallel and serial episodes. If composite episodes have been studied they were usually studied in the above mentioned, more restrictive form of sequences of parallel episodes.

3.1.4 Other frequency definitions

In this subsection we briefly present alternative frequency definitions without specifying counting algorithms. For the exact algorithms we refer the reader to the respective papers. Most alternative definitions tried to move away from the ideas of fixed windows and tried to improve the performance of the counting algorithms.

Minimal Occurrence Based Frequency Definition
The first alternate definition does uses the concept of minimal occurrences:

Definition 19 ***Minimal Occurrence*** *An event episode α is said to occur minimally in a window $W(S,q,r)$ if α occurs in W and there is no subwindow of W in which α also occurs.*

Definition 20 ***Episode Frequency - Minimal Occurrence based Definition*** *Given a sequence of events S and an Episode α, we define the minimal occurrence based frequency $mo\_freq(\alpha)$ as the number of minimal occurrences of α in S. TODO: find and cite the original source*

The second alternative definition introduces the concept of non-overlapping occurrences:

Definition 21 ***Non-Overlapping Occurrences*** *Given a m-Episode $\alpha = (V_\alpha, \leq_\alpha, g_\alpha)$ where $V_\alpha = \{v_1, ..., v_m\}$, two occurrences if all timestamps of one of the occurrences are bigger than all the timestamps of the other occurrence. Formally two occurrences h_1 and h_2 of α are non-overlapped if either:*

- $\forall v_j \in V_\alpha : h_2(v_1) > h_1(v_j)$ *or*
- $\forall v_j \in V_\alpha : h_1(v_1) > h_2(v_j)$

A set of occurrences is non-overlapping if every pair of occurrences in it is non-overlapped [22].

An example scenario visualizing overlapping and non-overlapping occurrences is visualized in figure 3.5.

This leads to the Definition:

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

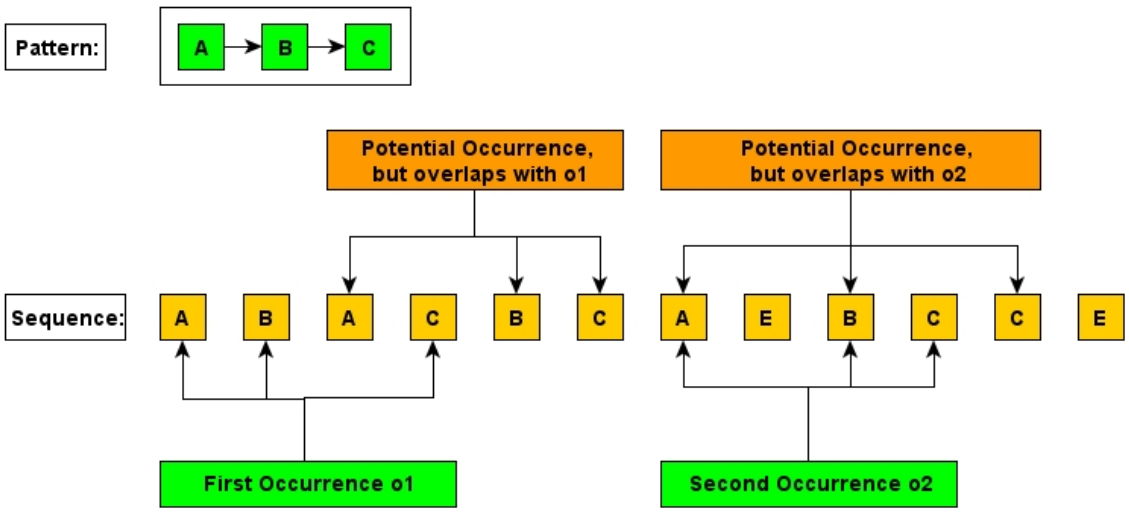

Fig. 3.5 An example sequence that shows how occurrences of episode patterns can overlap. Two non-overlapping occurrences are colored in green, whereas the orange occurrences would overlap with one of the green occurrences.

When looking at these definitions in comparison to the window based frequency definition 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 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 in many domains episodes happen within a small time window [**?**].
- The non-overlapping occurrence based frequency offers the fastest counting algorithm of all three definitions. However when incorporating expiry times for serial episodes it looses this advantage. Additionally previous literature has not yet identified an efficient algorithm to count non-overlapped occurrences of parallel episodes with expiry times.

3.1.5 Candidate Generation

Most previous work generates candidates for serial and parallel episodes separately, using a levelwise approach for both cases. The candidate generation procedure presented below was originally specified by Manila et. al. [28]. A more detailed explanation can be found in Laxman's PHD thesis [21].
We first consider the case for parallel episodes. Given F_k as the set of all frequent parallel episodes of length k we can generate the candidate parallel episodes C_{k+1} of length $k+1$ by doing the following:

1. Represent each candidate $\alpha \in F_k$ as a lexicographically sorted array of length k.
2. For each unordered pair (α, β), $\alpha, \beta \in F_k$ where α and β share the same first $k-1$ nodes, generate candidate γ by copying α and appending $\beta[k]$.

For example the two frequent parallel episodes $A\|A\|B$ and $A\|A\|C$ will generate the candidate $A\|A\|C\|D$.
The same procedure can be applied to serial episodes, except that

- we do not order lexicographically, instead the serial episodes remain in the array in their natural order
- each pair (α, β) with the same properties as above now generates two candidates:
  - γ_1 by copying α and appending $\beta[k]$.
  - γ_2 by copying β and appending $\alpha[k]$.

Thus the two frequent serial episodes $A \to A \to B$ and $A \to A \to C$ will now generate the two candidates $A \to A \to B \to C$ and $A \to A \to C \to B$.
Since the mining of composite episodes has not received much attention, it is unsurprising that there little related work that mentions candidate generation strategies for these general types of episodes (TODO: refer to later chapter, since I will do that). For a more strict definition of composite episodes which only includes sequences of parallel episodes, Baumgarten et. al. use a tree growth strategy to generate candidates for composite episodes [10].

3.2 Problem Definition

As already mentioned before the problem to be tackled in this thesis is the prediction of future events in data streams using episodes. The approaches shall have the following properties:

- **Local and fast prediction:** We do not focus on long term trends or on predictions of special values at fixed times (for example daily closing values of stocks), but instead aim to give predictions about the near future given the current state of the stream.
- **Usage of an Annotated Event Stream** Most forecasting methods (as reviewed in TODO) focus on regression, meaning the predictions of numerical values. In contrast to this we work on annotated event streams, meaning we deal with a stream of events, which have predefined types (which is necessary to discover episodes) and we aim to predict the occurrences of events of certain types.

Since a lot of basic streams are numeric in nature we define:

Definition 23 ***Low Level Event Stream*** *A Low Level Event stream is defined as a (possibly infinite or constantly updating) sequence:* $LLES = [v_1, v_2, ...]$ *where each* v_i *has the form of a tuple:* $t_i = (t_i, v_{i1}, v_{i2}, ..., v_{in})$ *where* $t_i \in \mathbb{N}$ *is the timestamp of tuple i and* v_{ij} *is the j-th value of the i-th tuple. The datatypes of the values in the tuple depend on the concrete domain or stream, so they are not further specified. Common types are numerical, categorical or string values. The Low Level Event Stream is ordered by the timestamp values, but events are allowed to occur at the same time. Formally this means:* $T_i \leq T_j \implies i \leq j$.

We refer to the streams that we perform the suggested mining approaches on as Annotated Event Streams:

Definition 24 ***Annotated Event Stream*** *An Annotated Event Stream is defined as a (possibly infinite or constantly updating) sequence:* $AES = [(T_1, t_1), (T_2, t_2), ...]$ *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* $\forall i, j \in 1, ..., n \; i < j \implies t_i \leq t_j$.

Note that this definition is essentially the same as definiton 14, which defines event sequences. The only difference is that in an Annotated Event Stream we allow for new events to constantly come in, which makes the sequence possibly infinite. If the original underlying data source is a low level event stream it needs to be transformed to an annotated event stream, which can be done via a transformation procedure:

Definition 25 ***Tranformation Procedure*** *A transformation procedure is a mapping* f *that takes a low level event stream LLES as an input and outputs an annotated event stream AES as well as the corresponding event aphabet* Σ.

Note that the selection or development of the transformation procedure is extremely important for the success of the subsequent mining of the annotated event stream. If the annotated events that are generated are largely meaningless due to a suboptimal transformation it is unlikely that the mining process will discover episodes that are helpful to predict occurrences of the desired events. TODO: Aggregation, discretization If the low level annotated stream is processed in an online scenario, it is necessary to also do the transformation in an online way, which imposes additional restrictions on the transformation process (TODO: exmplain those).

Given an annotated event stream AES it is the goal of this thesis to develop algorithms that will build predictive models:

Definition 26 ***Predictive Model*** *A predictive model for $M(T)$, $T \in Sigma$ is a model that if given the current state of an annotated event stream will output a binary value in the following:*

- 1 *if the model expects an event of type T to occur in the stream in the near future*
- *0 otherwise.*

Definition 26 purposefully remains rather general and thus vague, since a predictive model can naturally function in many ways. Thus some of the key points, such as what is the state of a stream and what exactly does the term "near future" mean mathematically remain unspecified here and instead will be specified by the concrete algorithms. Nevertheless one should note that in most streaming scenarios we cannot store the whole stream, meaning it is impractical for a model to demand that the current state of the stream is defined by its entire history. Likewise a model may define that the "near future" means an extremely long time, thus most likely making the model correct if it simply outputs 1 every time, however such a model is obviously not very useful.

Chapter 4

Suggested Algorithms

This chapter presents the suggested learning algorithms for predictive models in detail. Section 4.1 explains predictive episode mining, an approach that aims to discover predictive episodes in the stream. Section 4.2 explains the idea of feature based stream window classification in order to predict event occurrences.

4.1 PERMS - Predictive Episode Rule Mining in Streams

The first suggested algorithm is called the PERMS algorithm, short for **P**redictive **E**pisode **R**ule **M**ining in **S**treams.

4.1.1 Basic Ideas and Definitions

In order to fully explain the algorithm we first need to define what we mean by predictive episode rules.

Definition 27 ***Predictive Episode Rule*** *An episode α is called a predictive episode rule for event $A \in \Sigma$ if α has the form $\beta \rightarrow A$ where β is an episode.*

In the context of a predictive episode rule $\alpha = \beta \rightarrow A$ we also refer to β as the prefix and to A as the suffix of α. The basic idea of the prediction algorithm using predictive episode rules is rather simple. If given a set P of predictive episode rules, we monitor the stream and whenever we detect the prefix of an episode in P we predict an occurrence of A. The idea is similar to the mining of sequential rules or rule-based classification (TODO: cite). Of course there is an infinite amount of predictive episode rules for an event type, finding those predictive episode rules that are actually useful for predicting the occurrence of an event is the main task when building (training) the model. The use case here is very similar

to the related work that deals with the constraint based mining of episode rules (TODO: cite), where episode rules are mined in order to discover dependencies between earthquakes. There are two main important differences that make the approach used by the authors (TODO: name them) impractical.

- We are interested in episode rules of one specific type (those predicting the desired event A), thus it is not necessary to mine all frequent episode rules in the stream.
- We are dealing with data in a streaming environment, meaning we can neither analyze the whole data, nor make multiple passes over the entire stream.

In order to describe how the set P is built by the suggested learning algorithm,we first need to define frequency, support and confidence for predictive episode rules, which are similar to classic association rules(TODO: cite). Before this can be done however we need to determine which frequency definition we will use in this algorithm.

4.1.2 Choice of Frequency Measure

Recall that there are three main frequency measures that were proposed in the literature:

- The Window-based Frequency (see defintion 18)
- The frequency based on minimal occurrences (see definition 20)
- The frequency based on non-overlapping occurrences (see definition 22)

Their properties were already discussed in subsection 3.1.4. While the frequency measure using the non-overlapping occurrences is the latest measure and offers the best theoretical runtime, when recognizing episodes in a sequence it has a few drawbacks that are detrimental to its use in the given scenario. The first one is that it does not limit the duration of the episode occurrences. However limiting the duration of episode rules, or in other words giving episode occurrences expiry times (TODO: cite) is necessary when predicting event occurrences in streams. Consider for example the simple predictive episode rule $\alpha = B \rightarrow A$ and the following example sequence:

$$S = [(B,1),(C,2),(A,3),(B,4),...,(A,2000)] \tag{4.1}$$

The non-overlapping frequency definition recognizes both $(B,1) \rightarrow (A,3)$ and $(B,4) \rightarrow (A,2000)$ as equally valid occurrences of α in S. However it is to be expected that when looking for episode rules for predictive purposes in a stream the events should happen close

to each other (time-wise). This means that in this case $(B,1) \rightarrow (A,3)$ is likely a correct occurrence and prediction, whereas the occurrence $(B,4) \rightarrow (A,2000)$ is simply owed to the fact that at some point of time event A will occur again in the stream and $(B,4)$ happened to occur a long time before that, without there necessarily being a causality. Additionally the non-overlapping frequency assumes that there is one long sequence, from which the episodes are to be mined. The window-based frequency however works on a set of individual windows of a sequence or stream. In the previous work this had not been an advantage or disadvantage, since it did not focus on the streaming scenario, which meant that it was possible to analyze the entire sequence. This lead to the use of sliding windows over the sequence when using the window-based frequency. In the streaming scenario it is impossible to use all of the data, instead a certain selection is necessary. This can be done easily when using the window-based frequency by simply storing the windows that are of interest. Selecting data for the non-overlapping frequency definition is more difficult. Thus we will use the window-based frequency in this approach

4.1.3 Basic Definitions

Since we decided on a frequency measure it is now possible to define frequency, support and confidence of predictive episode rules.

Definition 28 ***Frequency*** *If given a set of time windows WIN (see definition 17) of a sequence the frequency of a predictive episode α is defined as the number of windows in which α occurs: $freq(\alpha) = |\{W \,|W \in WIN \wedge \alpha \text{ occurs in } W\}|$*

Definition 29 ***Support*** *If given a set of time windows WIN of a sequence the support of a predictive episode α is defined as $s(\alpha) = \frac{freq(\alpha)}{|WIN|}$ TODO: source*

Definition 30 ***Confidence*** *The confidence of a predictive episode $\alpha = \beta \rightarrow A$ is defined as $c(\alpha) = \frac{freq(\alpha)}{freq(\beta)}$ TODO: source*

The intention is clear and similar to the mining of classic association rules (TODO: cite): If a rule has a high confidence, it means that the prefix of the rule rarely occurs without its suffix A, meaning there is a high chance that this is a true predictor for the event A. Thus the goal of PERMS is to find a set of predictive episode rules P that have a very high confidence and are above a certain support limit.

4.1.4 Choice of training data

Recall that in streaming applications it is impossible for to analyze the whole stream as one sequence using multiple passes. Thus, when presented with a stream any prediction or forecasting algorithm first needs to take some time to study the stream and extract training data to build the model. So if given an annotated event stream, how do we determine the training data? As it was described in section 3.1 the data basis for episode mining is one very long sequence. Thus the first simple approach would be to simply record the stream as a sequence until we have reached a number of desired elements or run out of memory. The recorded sequence would then be the training sequence from which the predictive episode rules can be mined. This approach is visualized in figure 4.3.

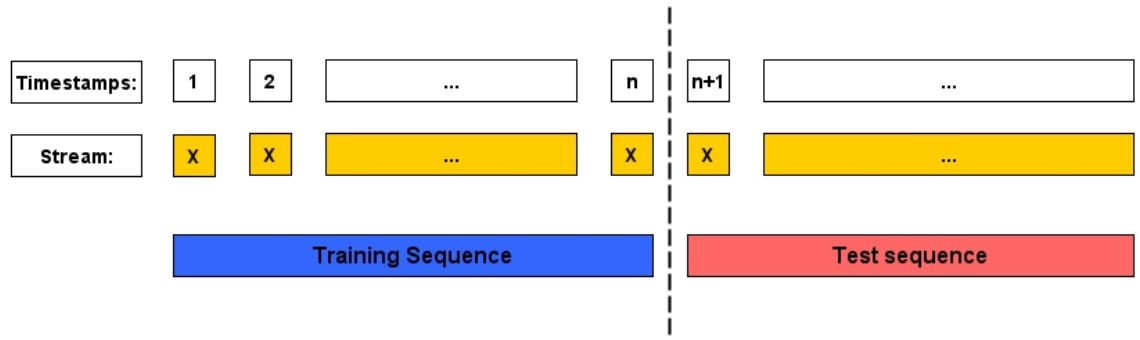

Fig. 4.1 Visualization of a simple split of the stream into a training segment at the beginning, followed by a (potentially endless) test phase

The disadvantage of this naive approach is that there is no guarantee about the number of occurrences of the event we aim to predict. Say we want to predict A and A is rather sparse in the beginning of the stream, then we will have a very small data basis to extract predictive episode rules for A and thus will likely not succeed. A better approach is to scan the stream for occurrences of the event A and whenever an event of type A enters the window we store the current window in a list until we have a sufficient number of windows. This approach is visualized in figure 4.2. This approach guarantees that we have a sufficient number of windows that include the target event at their end.

The obvious and very big disadvantage is that there are no negative examples in the training sample taken from the stream. Each window ends with the target event A, thus every episode mined from the windows can have A appended as a suffix and thus every predictive episode rule will have a confidence of 1.0. This means that selection via confidence is meaningless, since we have seen no negative examples, meaning windows that are not

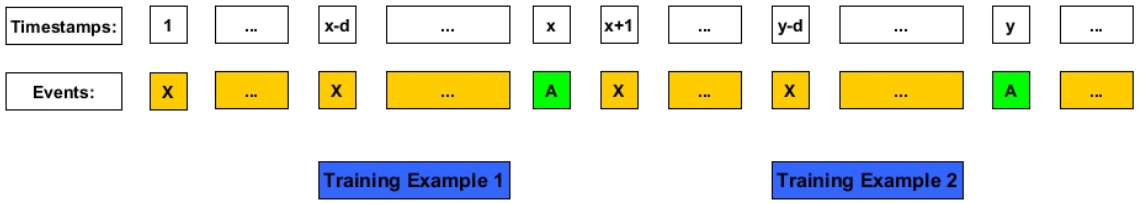

Fig. 4.2 Visualization of using fixed windows that precede the target event as training examples. Predictive episodes can be mined from the windows that are extracted from the stream as shown above.

followed by *A*. However negative examples can be extracted from the stream in a similar manner. This is the basic idea behind the training data selection in PERMS. It is visualized in figure **??**.

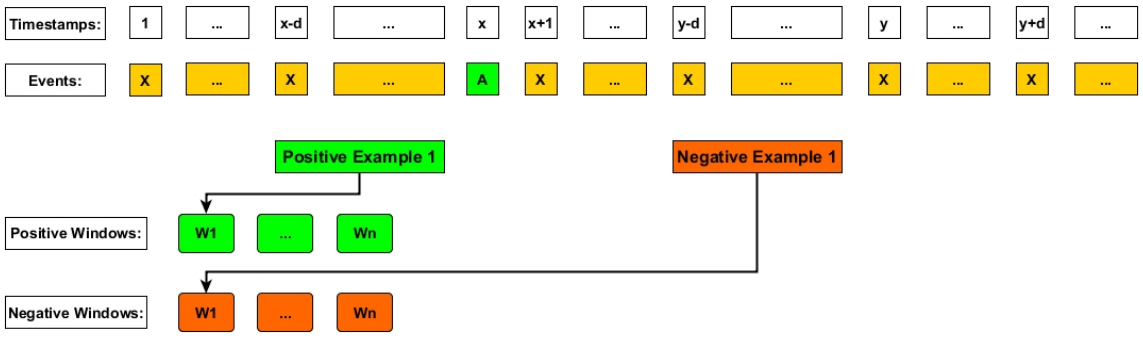

Fig. 4.3 Extracting positive and negative example windows from the stream.

There are still some issues with this approach that can be detrimental to the predictive performance of the model. One issue is that, in the simple standard scenario extract an equal amount of positive and negative windows from the stream with no respect to the original distribution. If *A* occurs very rarely, having an equal amount of positive and negative examples in the training data does not reflect the original event distribution in the stream. If that is the case, this can be fixed by including a number of negative examples that is proportionate to the original distribution (which is either known or learned while extracting the training data). However it is unclear if that would actually have a significantly positive effect on the performance of the resulting model.

TODO: define negative and positive windows!

4.1.5 PERMS Parameters and Pseudocode

The PERMS algorithm uses the following user-defined parameters:

- d - the (temporal) size of the sliding window. This also implies that all windows which the predictive episode rules will be mined from will exactly have duration d. TODO: talk about episode duration?
- m - the number of windows to mine the predictive episode rules from. Basically the sample size we take from the stream.
- s - the minimum support that predictive episode rules must have to be considered for the model (rules with support s or higher are frequent).
- $|P|$ - the desired size of the final set of predictive episodes.

The basic idea of the PERMS algorithm is the following: We start at the beginning of the stream and keep a sliding window of a user defined size d in memory. Whenever an event of type A enters the window we store the current window in a list until we have a sufficient number of windows (m). Additionally we also store m windows that do not contain A and were also not followed by A in the near future. Once we have enough windows, we can start to mine serial and parallel episodes from the windows preceding A that have support s or higher. Afterwards we rank the discovered rules by confidence, keep the $|P|$ episodes with the highest confidence and return them as the set of predictive episodes P. The subsequent application of the predictive model to the stream is very simple: If an episode $\alpha \in P$ occurs in the current sliding window output 1 and otherwise output 0 for the current sliding window. TODO: pseudocode plus images

4.2 Feature Based Stream Window Classification

4.3 Evolving the models with the stream

Chapter 5

Empirical Evaluation

Chapter 6

Conclusion and Future Work

References

[1] Abraham, A. (2005). Artificial neural networks. *handbook of measuring system design*.

[2] Aggarwal, C. C., Han, J., Wang, J., and Yu, P. S. (2003). A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 81–92. VLDB Endowment.

[3] Aggarwal, C. C., Han, J., Wang, J., and Yu, P. S. (2004). A framework for projected clustering of high dimensional data streams. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 852–863. VLDB Endowment.

[4] Agrawal, R., Imieliński, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM.

[5] Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499.

[6] 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.

[7] Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. (2002). Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM.

[8] Babcock, B., Datar, M., Motwani, R., et al. (2003). Load shedding techniques for data stream systems. In *Proc. Workshop on Management and Processing of Data Streams*. Citeseer.

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

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

[11] Chang, F., Chang, L.-C., Huang, H.-L., et al. (2002). Real-time recurrent learning neural network for stream-flow forecasting. *Hydrological Processes*, 16(13):2577–2588.

[12] Domingos, P. and Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80. ACM.

[13] Fama, E. F. (1965). The behavior of stock-market prices. *The journal of Business*, 38(1):34–105.

[14] 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.

[15] Funahashi, K.-I. (1989). On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192.

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

[17] Gama, J. (2010). *Knowledge discovery from data streams*. CRC Press.

[18] Guha, S., Mishra, N., Motwani, R., and O'Callaghan, L. (2000). Clustering data streams. In *Foundations of computer science, 2000. proceedings. 41st annual symposium on*, pages 359–366. IEEE.

[19] Inman, R. H., Pedro, H. T., and Coimbra, C. F. (2013). Solar forecasting methods for renewable energy integration. *Progress in energy and combustion science*, 39(6):535–576.

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

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

[25] 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.

[26] Manku, G. S., Rajagopalan, S., and Lindsay, B. G. (1999). Random sampling techniques for space efficient online computation of order statistics of large datasets. In *ACM SIGMOD Record*, volume 28, pages 251–262. ACM.

[27] 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.

[28] 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.

[29] Meger, N., Leschi, C., Lucas, N., and Rigotti, C. (2004). Mining episode rules in stulong dataset. In *In Proc. of ECML/PKDD'04 Discovery Challenge-A Collaborative Effort in Knowledge Discovery. Prague: Univ. of Economics*. Citeseer.

[30] Méger, N. and Rigotti, C. (2004). Constraint-based mining of episode rules and optimal window sizes. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 313–324. Springer.

[31] Tatti, N. and Cule, B. (2011). Mining closed episodes with simultaneous events. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1172–1180. ACM.

[32] Teräsvirta, T. (1994). Specification, estimation, and evaluation of smooth transition autoregressive models. *Journal of the american Statistical association*, 89(425):208–218.

[33] 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.

[34] Veit, A., Goebel, C., Tidke, R., Doblander, C., and Jacobsen, H.-A. (2014). Household electricity demand forecasting: benchmarking state-of-the-art methods. In *Proceedings of the 5th international conference on Future energy systems*, pages 233–234. ACM.

[35] Vreeken, J. and Tatti, N. (2012). Summarising event sequences with serial episodes. In *FIFTH WORKSHOP ON INFORMATION THEORETIC METHODS IN SCIENCE AND ENGINEERING*, page 82.

[36] Wang, J., Han, J., and Pei, J. (2003). Closet+: Searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 236–245. ACM.

[37] Wu, C.-W., Lin, Y.-F., Yu, P. S., and Tseng, V. S. (2013). Mining high utility episodes in complex event sequences. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 536–544. ACM.

[38] Zhang, D., Gunopulos, D., Tsotras, V. J., and Seeger, B. (2002). Temporal aggregation over data streams using multiple granularities. In *International Conference on Extending Database Technology*, pages 646–663. Springer.

[39] Zhang, G. P. (2003). Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175.

[40] Zhang, Y. and Wu, L. (2009). Stock market prediction of s&p 500 via combination of improved bco approach and bp neural network. *Expert systems with applications*, 36(5):8849–8854.

[41] Zhou, W., Liu, H., and Cheng, H. (2010). Mining closed episodes from event sequences efficiently. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 310–318. Springer.

Appendix A

Episode Mining

A.1 Window-Based Frequency Counting Algorithms

The algorithms to determine the window based frequency of serial and parallel episodes are given in algorithm 2 and 3 respectively. The basic ideas are described in the respective sections. The source for both algorithms is the paper published by Mannila et. al. [28].

A.1.1 Frequency Counting of Parallel Episodes

The algorithm for counting the frequency of parallel episodes uses the following data structures and variables:

- For each event type A store the number of occurrences of this event in the current window in $A.count$
- For each episode α store
  - $\alpha.freq$ - the number of windows in which alpha occurred so far
  - $\alpha.eventCount$ - the number of events of $alpha$ that are present in the current window
  - $\alpha.inwindow$ - this variable gets set to the current time whenever α becomes fully present ($alpha.eventCount = |\alpha|$).
- additionally maintain *contains* which is a set of sets. Each set in *contains* is identified by a tuple (T,i), where $T \in \Sigma$ is an event type and $i \in \mathbb{N}^+$. These sets are referred to as $contains(T,n)$. The set $contains(T,n)$ contains all candidate parallel episodes, which contain the event type T exactly n times. This is done to be able to efficiently access candidates, when certain events enter or leave the window.

The algorithm works as follows. First the above mentioned variables and index structures are initialized. Subsequently we perform the main loop which iterates through every point of time between the start and the end time of the sequence (TODO: they start earlier for some reason?). Essentially each iteration is sliding the window one step forwards. The only two things that need to be handled inside the loop are new events coming into the window and old events dropping out of the sliding window. If a new event comes in its count gets updated and each episode α that is affected by this will get its event count updated and, if it is completed, the current time will be saved in $\alpha.inwindow$.
If an event A drops out of the window, for all episodes α that occurred in the previous window(s) and now no longer have an occurrence in the current window (by losing A) the number of windows they were present in gets added to $\alpha.freq$.

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.

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** $\{\alpha \mid \alpha \in C \wedge \frac{\alpha.freq}{t_e - t_s + win - 1} \geq minS\}$

A.1.2 Frequency Counting of serial Episodes

The basic idea when determining the window based frequency of serial episodes is that a serial episodes can be recognized by using automaton that accepts the events of its corresponding serial episode in exactly the specified order and ignores all other input. For each serial episode α there can be several instances of the recognizing automaton (in different states) at the same time. This is necessary in order to replace old occurrences with newer ones whenever possible. The algorithm for counting the frequency of serial episodes uses the following data structures and variables:

- Each episode α is represented as an array in which the event types contained in α are stored in the correct order
- For each episode α the number of windows in which alpha occurred so far is stored in $\alpha.freq$
- An automaton is simply represented by a tuple (α, i), where α is the corresponding candidate serial episode and $i \in \{1, ..., |\alpha|\}$ is the state (position in the episode) in which the automaton currently is.
- The automata are grouped by the event type that will allow them to perform the next transition. These lists are referred to as $waits(T)$, where $T \in \Sigma$ is an event type.
- For each automata belonging to episode α that is currently in state i, the time at which it this automaton was initialized is stored in $\alpha.initialized[i]$.
- Additionally any automata that were initialized at point of time t will be contained in a list referred to as $beginsat(t)$.
- transitions to be made are stored in a list named transitions and are stored in the form (α, i, t), where α is the corresponding episode, i is the index of the state from which the automaton will transition to the next one and t is the time in which this automaton was initialized.

The structure of the algorithm is the same as the one of algorithm 2. First the variables are initialized, then the sequence is looped over and the sliding window moves by one time unit in each iteration. A new instance of an automaton is initialized, whenever an event A is the first event of an episode and enters the window. Additionally all automata that wait for A will be moved to the next state, while memorizing their initial starting time. If an automaton of episode α moves to a state which is already occupied by another automaton, the old

automaton is discarded (since the newer one has a later starting time and thus will be present in more windows). If an automaton reaches its final state the current time is memorized in $\alpha.inwindow$. If an automaton in its final state expires (its starting time drops out of the window) the number of windows it was present in is added to its corresponding sequence.

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: part of the algorithm is cut off

1: *// Initialization*
2: **for** each $\alpha \in C$ **do**
3: **for** $i \in \{1,...,|\alpha|\}$ **do**
4: $\alpha.initialized[i] \leftarrow 0$
5: $waits(\alpha[i]) \leftarrow \emptyset$
6: **for** each $\alpha \in C$ **do**
7: $waits(\alpha[1]) \leftarrow waits(\alpha[1]) \cup \{(\alpha,1)\}$
8: $\alpha.freq \leftarrow 0$
9: **for** $i \in \{t_s - win,...,t_s - 1\}$ **do**
10: $beginsat(t) \leftarrow \emptyset$
11: *// Recognition*
12: **for** $start \leftarrow t_s - win + 1$ to t_e **do**
13: *//Bring new events to the window*
14: $beginsat(start + win - 1) \leftarrow \emptyset$
15: $transitions \leftarrow \emptyset$
16: **for** each $(t,A) \in S$ where $t = start + win - 1$ **do**
17: **for** each $(\alpha, j) \in waits(A)$ **do**
18: **if** $j = |\alpha| \wedge \alpha.initialized[j] = 0$ **then**
19: $\alpha.inwindow \leftarrow start$
20: **if** $j = 1$ **then**
21: $transitions \leftarrow transitions \cup \{(\alpha, 1, start + win - 1\}$
22: **else**
23: $transitions \leftarrow transitions \cup \{(\alpha, j, initialized[j-1])\}$
24: Remove $(\alpha, j-1)$ from $beginsat(\alpha.initialized[j-1])$
25: $\alpha.initialized[j-1] \leftarrow 0$
26: Remove (α, j) from $waits(A)$
27: **for** each $(\alpha, j, t) \in transitions$ **do**
28: $\alpha.initialized[j] \leftarrow t$
29: $beginsat(t) \leftarrow beginsat(t) \cup \{(\alpha, j)\}$
30: **if** $j \leq |\alpha|$ **then**
31: $waits(\alpha[j+1]) \leftarrow waits(\alpha[j+1]) \cup \{(\alpha, j+1)\}$
32: *// Drop old events out of the window*
33: **for** each $(\alpha, l) \in beginsat(start - 1)$ **do**
34: **if** $l = |\alpha|$ **then**
35: $\alpha.freq \leftarrow \alpha.freq + start - \alpha.inwindow$
36: **else**
37: Remove $(\alpha, l+1)$ from $waits(\alpha[l+1])$

Naturally there are enhancements to these algorithms. For example both counting algorithms iterate over each point of time t between the start and end time of the sequence, regardless of the fact that there might not be anything to do at this point of time (no event drops out of the window and no new event comes in). Especially if the sequence of events is sparse, meaning that the time between events is usually large, this will become problematic. This issue can be fixed rather easily: instead of increasing *start* by one in each iteration, one can increase start by the amount of time that is needed until a change in the window occurs and update the data structures accordingly.