Evolving Systems (2013) 4:61–70
DOI 10.1007/s12530-012-9067-0

ORIGINAL PAPER

EVE: a framework for event detection

Iris Adä · Michael R. Berthold

Received: 13 February 2012 / Accepted: 25 September 2012 / Published online: 3 November 2012

Abstract In this paper, we introduce EVE, a generic framework for event detection where events can also include outliers, model changes and drifts. Various methods for event detection have been proposed for different types of events. However, many of them make the same or very similar prior assumptions but use different notations and formalizations. EVE provides a general framework for event detection, which allows existing algorithms to be represented using a common basis. The framework includes generic types of time slots, streaming progresses, and measures of similarity between those slots. We demonstrate how existing algorithms fit nicely into this framework by instantiating appropriate window combinations, progress mechanisms, and similarity functions.

Keywords Event detection · Data mining framework · Stream mining · Change detection

1 Introduction

Knowledge discovery is usually concerned with identifying the underlying general theme, i.e. the modeling of the underlying overall system. However, new knowledge is often also generated by detecting the "odd one out". There are many aspects of this type of knowledge discovery, the most obvious one is known as *outlier detection*. However being able to notice when the underlying models slowly (or abruptly) changes—often referred to as *model change* or *model drift*—is also of importance.

In recent years most of this type of work has been conducted under the umbrella of stream mining (Gama 2010, 2012). Here the focus lies on the processing of data online where it is usually not possible to analyze all the data or—in a more extreme case—even access past data points. Aside from these added difficulties of data access, a very prominent topic in data stream mining is *event detection* where an event denotes the departure from the normal, e.g. some irregular behavior observed in the data. Irregularity in stream mining indicates that current observations are not related to concepts derived from previous observations. As in normal data mining, events in data stream mining can be outliers, misclassified patterns, or an overall shift or change of the underlying model. In data stream mining literature the terms *novelties* and *anomalies* are also often used for these types of events.

A detailed overview of various novelty detection approaches in stream mining can be found in Markou and Singh (2003). Less isolated events, such as model changes and drifts, have also increasingly drawn attention in the last few years (Widmer and Kubat 1996; Hoens et al. 2012; Zliobaite 2009).

When these different kinds of algorithms are investigated in more detail, it quickly becomes apparent that many of them follow the same underlying principle. They all store some type of summary information about the past behavior of the stream and compare this representation to the present situation. Of course they use various different methods of comparison in order to detect different types of events but the underlying mechanism stays the same.

Based on earlier work (Adä and Berthold 2011), in this paper we propose EVE, a general framework for event detection encompassing different approaches but also different types of events. Providing a general framework not

I. Adä (✉) · M. R. Berthold
Nycomed Chair for Bioinformatics and Information Mining, Universität Konstanz, 78457 Konstanz, Germany
e-mail: iris.adae@uni-konstanz.de

M. R. Berthold
e-mail: michael.berthold@uni-konstanz.de

only allows easy comparison of different approaches, but EVE also enables different types of event detection approaches to be categorized.

The most straight forward approach for event detection would function in a similar manner to lazy machine learning and simply store all of the past data points. In reality this is not generally possible. Most existing approaches either store a certain window of data (either a sliding window or starting from some fixed data point in the past) or build/update a model summarizing all the data or a subset of the data seen so far. Depending on this choice, actual event detection subsequently compares either this past data or the model representing past data with the present data point or window. Depending on the focus of event detection (outlier, drift/change) the present data to be considered will range from one single instance to a larger window. The second ingredient of event detection algorithms is the type of comparison – some sort of similarity function needs to be used to determine whether or not the past data (or the model abstracting the past data) matches the present data. Various alternatives exist, comparing either two data windows, a model to a data window or two models.

At this, fairly abstract level, the main ingredients of every event detection are therefore:

- Window modeling: how does the method keep information about the past: as a (sliding) window and/or as an aggregating model? How much of the most recent data is used: just one data point for outlier detection or a larger window for (gradual) model change?
- Dissimilarity: how is past and present information compared? By matching data (window) against data (window), by matching a model to data, or by matching two models?
- Detection: when is an event reported to the analyst?

In the following we will first introduce the window modeling itself in more detail and then show the importance of the dissimilarity function and how it can be formalized. Afterwards we discuss related work and show how it can be embedded into the presented framework. A demonstration of the framework on exemplary data sets should further underline the general idea. We conclude by briefly discussing an implementation of the system, which allows different variations of this framework to be evaluated flexibly.

2 Terminology

Before detailing the EVE framework we first need to clarify the terminology we will be using in the remainder of this paper. As already mentioned the key idea of event detection is to detect irregular behavior in the data, which is unexpected or unknown with respect to previous observations. The term "event" is used here to define everything that is new and potentially useful. Hence, information about the event is in some sense (often statistically) different to previous parts of the data. This information is considered to be important as it differs from already known or learned underlying models.

A number of problem categorizations exist, particularly in drift detection. The most common one distinguishes if a change is either gradual/slow or abrupt/sudden based on the level or speed of change (e.g. Gama et al. 2004). Another categorization splits the gradual changes further, by considering whether the change has occurred in the attribute space or in the class label, and adds the possibility of a reoccurring change (Zliobaite 2009). Recently Minku et al. (Minku et al. 2010) proposed a more detailed categorization, splitting the possible drift into various categories such as isolated vs. sequenced drift on the top level, or speed, predictability, frequency, etc. at lower levels. This detailed categorization helps to clearly identify differences and similarities in future drifting methodologies.

For this work we will use the well-known but less detailed distinction between abrupt versus gradual events (e.g. discussed in Tsymbal 2004; Gama et al. 2004) to classify the level of difference and speed of change. In order to more specifically adjust to various event types another distinction is made by considering the duration of the event (short-term vs. long-term, a distinction we have not found in the literature). Figure 1 shows the resulting four main event types.

The first type of events are *short-term events*. These are mostly detectable by a single data point only. This data point either does not match the previous data at all or belongs to a subset of data for which there are no recent comparisons. This type of event is already well known by the term outlier or anomaly (Chandola et al. 2009). Short-term events are always abrupt events.

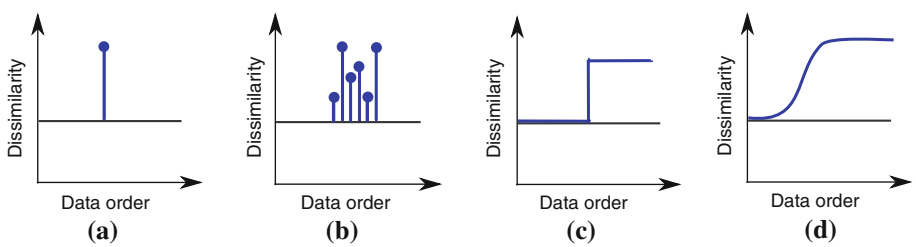


Fig. 1 A visualization of the four possible types of events, as mentioned in the terminology Sect. 2, is depicted above. The major distinction is based on the duration of the event, as short-term events (**a** abrupt and **b** collective) and long-term events (**c** abrupt and **d** gradual)

A short-term event can also be caused by more than one data point. In this case, the *collective short-term event* demonstrates high similarity between the data points and high dissimilarity to data points seen before and also to subsequent observations. The behavior of the data abruptly returns to the previous behavior after the event. In current literature (e.g. Markou and Singh 2003), this event type is also referred to as *novelty*.

Events that are induced by multiple data points are called *long-term events*. The main difference is that they are induced by multiple data points and the data configuration does not return to the one seen before. If there is a clearly identifiable change in the data configuration the event is referred to as an *abrupt* event.

Contrary to short term events, long term events can also be characterized by slow changes of the data. Instead of single data points not fitting to the previous scheme, subsequent data points slowly slide away from the previously built model. This change in the data is of considerable interest as it indicates a slow but continuous event in the data. Such *gradual long-term events* also bring a special demand to the detection process, as their exact start is hard to detect.

3 EVE: the event detection framework

This section explains the core of the unifying event detection framework. Detecting events fundamentally relies on comparing past to present situations, which are, in our case, represented by data points from different windows in time. Events are then characterized by the two windows being different in some way. Various methods can be used to measure this dissimilarity between the past (represented by the data points in the *past window*) and present (the *present window*). We can directly compare the data points in both windows or compare models abstracting the data points in the windows. Such models can be simple clusters or more complex abstractions such as decision trees or rules. (We can, of course, also compare a data set to an abstract model.) The dissimilarity measured between the two windows indicates the probability of an event. For example: A model, built with knowledge from an outlier can never be similar to a model built on normal behavior only.

We will discuss appropriate dissimilarity methods in Sect. 3.2, detection methods in 3.3 but start with providing a more formal discussion of the windows types and process.

3.1 Window modeling

As described above, the event detection framework described here is based on two windows. The first window, which is used for summarizing the known/pre-vious/old behavior of the data, is referred to as the *past*. It should start and end a certain number of steps before the currently regarded data point. The later window, which possibly contains/incorporates knowledge about the event, is referred to as the *present* window. This window is supposed to contain the most recently considered data points.

3.1.1 Window configuration

The first step in the framework is to formalize the different window types. This formalization will later help to clearly identify different types of window configurations.

A window type is defined by two terms. The first term refers to the behavior of the start position of the window and can either be *fixed*(F) or *sliding*(S). The width of the window is described by the second term. It can be of *constant*(C) or *growing*(G) size. Although the windows are described as subparts of the overall stream, only a summarizing model has to be saved, and not the whole information. This formalization results in four possible configurations of a window:

- FC: The fixed constant window does not change during the progress of the detection mechanism.
- FG: This window type always starts at the same position(F), but incorporates one new data point in every step(G). Although the theoretical window size is constantly increasing, this does not have to be the case for the model. The model of the previous time step can only be updated according to the newly achieved data point.
- SC: The sliding and constant window changes its start and end position simultaneously. Hence, the size of the window does not change, only the start position does.
- SG: See comments below.

3.1.2 Comments on sliding growing windows

Using the default idea of data stream mining, i.e. learning one new data point with each step of the algorithm, a window cannot change its start position and increase its size at the very same point in time.

Some approaches in change and drift detection already use the idea of variable-sized windows. The Drift Detection Method (Gama et al. 2004) extends the size of the window(FG) until a drift level is reached and then restarts learning. In the windowing case, restarting learning means setting the window size to 0. The FLORA2 algorithm (Widmer and Kubat 1996) uses a window adjustment heuristic to decide when the window should be shrunk, grown or kept at a constant size. The ADWIN(Adaptive sliding window) methodology (Bifet 2010) increases the

window size until two subwindows are found that are "distinct enough". In this case the elements from the start of the window are dropped until the statistics fulfill the null hypothesis.

Overall one can summarize that using a rule base to define sliding and growing behavior is the most promising direction for SG-windows. However, there are multiple possibilities for defining an overall concept of SG-windows.

3.1.3 Combining windows

Two window types are needed to define the combination type. For example, the past window is XY and the present window $\bar{X}\bar{Y}$, therefore the combination type is described as $XY \hookrightarrow \bar{X}\bar{Y}$, where $X, \bar{X} \in \{\mathrm{F}, \mathrm{S}\}$ and $Y, \bar{Y} \in \{\mathrm{C}, \mathrm{G}\}$

3.1.4 Window formalization

A data stream $D = (d_0, d_1, d_2, \ldots)$ is defined as an infinite sequence. The data point d_i arrives at time point i and the time lag between two data points d_i and d_j can be calculated as the distance $|i - j|$. The set of all i is defined as T. T is a subset of $\mathbb{N}$, T is a closed interval and $1 \in T$. Subsequences of the stream form the windows used for the learning process: $D_{[i,j]}$ is defined by data points arriving between and including time i and j.

A window combination, which refers to two subsequences of the data, is configured by a *window configuration function* **a:**

$$\mathbf{a} : T \mapsto T^3 : t \mapsto (a_0(t), a_1(t), a_2(t))$$

$$\exists t_0 \in T : \forall t > t_0 \in T : \{a_0(t), a_1(t), a_2(t)\} \subseteq T$$

The subsequences used for present and past windows can be derived from function $\vec{a}$. The past window at time point t is

$$D_{[a_0(t), a_1(t)]}$$

and the present

$$D_{[a_2(t), t]}.$$

3.1.5 Window filtering

We obtain four possible window types, which result in 16 different combinations types. As already mentioned the SG window type is not covered here, hence only nine combinations remain.

We start by filtering combinations which are either not usable at all, or are not used in current literature. The FC↪FC combination, which can be seen in Fig. 2 does not change at all and hence is not usable for any detection process.

As determined by the definition of present and past, the present window should always end after the past window. However, if the present window is modeled by a Fixed Constant window, the past will overtake the present. This can be seen in the two cases shown in Fig. 3.

In the final filtering step, we consider strong overlapping windows as depicted in Fig. 4. If the second window is realized by an FG window, it is not feasible for the first window to change its end position as well. Considering, e.g. the FG↪FG combination in conjunction with the progress of the algorithm, the overlap is going to be the main part of modeling. A dissimilarity function would therefore naturally decrease over time, as both slices mainly contain the same information. A similar explanation can be given for the SC↪FG combination. Figure 5 depicts the four remaining combinations.

3.1.6 Window combinations

1. FC↪SC: One of the pioneering works in the area of change detection (Kifer et al. 2004) presented a change detection framework based on the first data points and a second constant window ending at the most recent data point. This also refers to our first window combination. The past is modeled as a never changing model at the start (t_0) of the data and the present is a sliding window of fixed size s_1. The respective window configuration function is therefore defined by:

$$\vec{a}(t) = (t_0, t_0 + s_0, t - s_1)$$

When s_1 equals 1, this can be used to detect short term events. A long-term event can be detected with $s_1 \gg 1$.

2. FC↪FG: In the second combination the past window is a subsequence at the start of the data set of size s_0. With every step the present window incorporates the new data point.

$$\vec{a}(t) = (t_0, t_0 + s_0, t_0 + s_0 + 1)$$

3. SC↪SC: The third approach uses two synchronous sliding windows. Hence we achieve:

$$\vec{a}(t) = (t - s_0 - s_1, t - s_1 - 1, t - s_1)$$

(s_0 is the size of the first window; s_1 is the size of the second window) This approach has been used by Dries and Rückert (2009) for example.

4. FG↪SC: The probably most typical data stream algorithm type incorporates all of the previous information in the past and uses a single data point for the present model. This is a special case of the last modeling possibility. Only a sliding window of a fixed size s_1 is used for the present window and all of the previous data is used for the past. This can be formalized by the following window configuration function:

Fig. 2 A visualization of the FC↪FC combination over four consecutive time steps

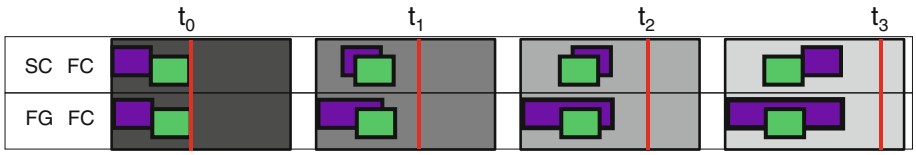


Fig. 3 Shows how SC↪FC and FG↪FC evolve over time. As the present window does not change, the past window overtakes it

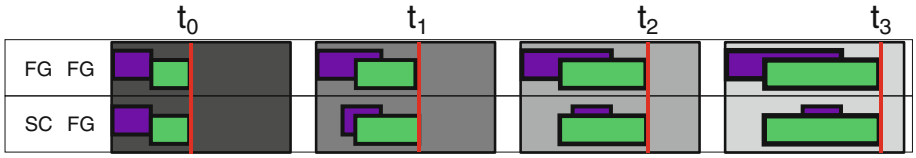


Fig. 4 Combinations ending on a fixed growing window suffer from a strong overlapping of the two windows

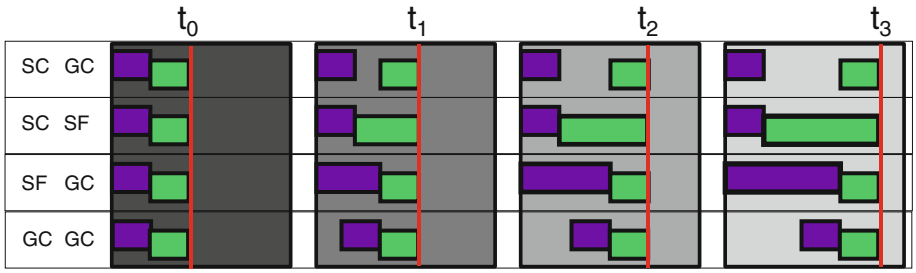


Fig. 5 The four combinations remaining after filtering are shown here. The figure visualizes how they start with the same initial configuration and proceed to evolve over time

$$\vec{a}(t) = (t_0, t - s_1 - 1, t - s_1)$$

Note that all of the combinations start with the same base line: two windows following each other directly. It is quite obvious that a constant gap can be introduced with an extra input parameter g added to $a_2(t)$.

Most of these approaches can be directly achieved via every data mining algorithm. Therefore the model has to be created on the window with every step of the algorithm. Subsequently the main difficulty is to find a dissimilarity function between two windows or their models. However, recreating the model for each step from scratch considerably increases the complexity.

3.2 Dissimilarity

The next step in configuring an event detection algorithm is the definition of the dissimilarity function. The main goal of the dissimilarity function lies in simplifying the problem into a one dimensional function. This can be evaluated much more easily by the detection method afterwards. The dissimilarity function $d_a : T \mapsto \mathbb{R}$ first maps the time point t to the respective window combination. It is configured by using the window configuration function $\vec{a}(t)$ and finally measures dissimilarity between the two windows.

There are two different ways of handling the previously described windows. One can work directly on the data and define a similarity metric in the corresponding space. Alternatively, one can use a summarizing model. This consideration creates four possibilities for the dissimilarity function.

First, data windows can be compared directly. This setup represents the most general method, of course, and achieves the best accuracy. However, in many cases the resulting high complexity makes it infeasible. In this case we have to select a dissimilarity function which directly calculates between the two windows. Obviously for numerical data one can use the Euclidean distance or one of the other L_p-norms. But, of course, the model-based abstractions below can also be put under this category.

To reduce the complexity of the dissimilarity calculation a more compact model can be learned from one of the windows (mostly the past) and used to test the data of the second part (mostly the present). The dissimilarity function then averages or sums up the achieved errors. The advantage of this second and third possibility of comparing windows is that we can use e.g. classification or regression algorithms together with their evaluation possibilities. After building a classifier, such as a decision tree, on one window, it is applied to the other window and the dissimilarity is derived from the missclassification rate. With a regression function as the model the dissimilarity can similarly be calculated as the mean difference between the predicted and the true values of the second window.

As a final possibility, the dissimilarity function can be calculated directly between two models. By using only this more abstract information we are able to measure a more general event in the data. For partition-based clustering (as produced by k-means) the dissimilarity can be calculated using the idea behind Ward's method. The Kullback–Leibler divergence (Kullback 1987) compares two probability distributions, which are in this case the summarizing model. Finally it is also possible to compare the learned coefficients between two models, e.g. the dissimilarity between the gradients of the regression function.

In the following, D_i represents windows that are subsequently analyzed. Hence, D_{i-1} is positioned before D_i, which again is located before D_{i+1}. We use the following convention for the dissimilarity function to simplify the definitions $d(D, \bar{D}) = d_a(t)$ with $D := D_{[a_0(t),a_1(t)]}$. and $\bar{D} := D_{[a_2(t),t]}$. To achieve a good dissimilarity function, the function has to fulfill the following requirements.

1. **No event:** $d(D_1, D_2) \approx d(D_2, D_3)$ If the model does not change, this has to be the case for the dissimilarity function as well. We believe that only a change in the dissimilarity should be directly related to the possibility of an event.
2. **Abrupt event:** If D_3 is the first slot to incorporate knowledge about an abrupt event, the dissimilarity should abruptly increase ($d(D_1, D_2) \ll d(D_2, D_3)$). Such an abrupt change of the dissimilarity function mostly indicates that an outlier or novelty has been found in the data.
3. **Gradual event:** $d(D_1, D_2) < d(D_2, D_3) < \cdots < d(D_{i-1}, D_i)$ (respectively $>$) A gradual event in the data results in a dissimilarity between the models, which increases (or decreases) continuously over time. The exact start of such a gradual event is of particular interest, but is very difficult to detect.

3.3 Detection

Taking our previous observations, we can create a diagram, as seen in Fig. 6. The next step is to determine—preferably automatically—when an actual event is happening. Most algorithms use a threshold in order to be able to output whether an event has happened or not. However, we can find out more information by taking a closer look at the plot.

Based on the hypothetical dissimilarity function shown in Fig. 6 the detection process is split into different possible stages. Stage 1 describes normal behavior. In this phase all new unseen data points coincide with the previous ones. An abrupt change was found in stage number 2. Once this abrupt change has passed, phase 1 with its constant

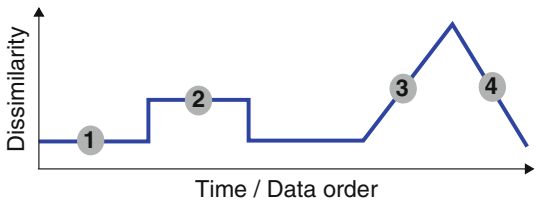


Fig. 6 A hypothetical dissimilarity function. The higher the peak, the more dissimilar the data becomes with respect to previous behavior. An event can be determined when values leave static behavior (marked by *1*)

value is seen in the error plot again. In contrast to the jump provoked by the abrupt change, the behavior of the function in Sect. 3 is different. The slow increase indicates that new data points influence the model causing it to slide away from the previous model. This rise describes a change in the data. In the reversal point between stage 3 and 4 a state is reached in which most of the previous model is forgotten and the system assimilates to a new changed underlying model.

One of the most common methods for the detection is, next to using a simple threshold, a control chart. A control chart (Shewhart 1980) uses an intelligent threshold by measuring the mean and standard deviation of the function over time. In the area of drift detection, many more algorithms are discussed, which can report events from a one dimensional function. Examples of these are: Drift Detection Method (Gama et al. 2004), Early Drift Detection Method (Baena-García et al. 2006) and EWMA (Ross et al. 2012).

3.4 Concluding remarks

We conclude this section with a short recommendation as to which type of window modeling is appropriate for different scenarios. Overall, it is difficult to give a recommendation based on the type of event, furthermore the system should be configured based on a detection goal or on assumptions that can be made.

- FC↪FG : The advantage of this combination is that a possible complicated but detailed model can be used for the past. As the present includes the following data set, it will also include possible events, thus increasing the dissimilarity for all the following comparisons.
- FC↪SC: For this combination, the data should only change for a certain time but always return to a base level. If the underlying model does not return to the previous one, the value of the dissimilarity function stays at a high level.
- SC↪SC: If this is not the case, i.e.the model changes without returning, the two sliding windows in this combination are advisable, as they accept the changed

model as normality over time and will even forget the event.
- FG↪SC: The last combination is the most flexible one. It includes all information about previous events in the past window. If similar events happen in the future they are accepted as normal.

4 Modeling existing approaches

The goal of this section is to demonstrate how a few well known approaches for detecting events, or strongly related approaches like change detection, anomaly detection, drift detection—to mention a few—fit into the presented framework.

First the system is applied to Shewharts 3-sigma control chart (Shewhart 1980). It is typically only used for one-dimensional data streams but can easily be extended to a multi-dimensional data stream by using a single Gaussian kernel to model the data.

Data: One-dimensional numerical data.
Window: FG $\hookrightarrow$ SC
Model: Mean μ and standard deviation σ.
Dissimilarity: $d(t) = x - (\mu + 3\sigma)$
Detection: Threshold: Warning if $d(t) > 0$

A recent approach uses Gaussian kernels (Bondu et al. 2010) to model the time windows. In this approach our framework provides clarity as it already uses a dissimilarity measure. The measure only depends on the updates in the model. Their approach can be formalized as follows:

Data: Multi-dimensional numerical data.
Window: FC $\hookrightarrow$ SC
Model: Gaussian Mixture Model (e.g. M_1,M_2)
Dissimilarity: $d(t) = KL(M_1,M_2)$ Kullback–Leibler-Distance (Kullback 1987) is used to measure the dissimilarity of two Gaussian Mixture Models.
Detection: Threshold: Warning if $d(t) > \alpha$

Another early approach (Japkowicz et al. 1995), which is concerned with the modeling of high-dimensional data, uses a neural network to model the data stream. The auto-associative network subsequently minimizes the difference between the input and the learned output for the training patterns.

Data: Multi-dimensional numerical data stream.
Window: FC $\hookrightarrow$ SC
Model: Neural network
Dissimilarity: $\sum (x_i - y_i)$, where x is the input vector and y the vector predicted by the neural network.
Detection: Threshold: Warning if $dist > \alpha$

5 Practical considerations

The window framework is realized in KNIME (Berthold et al. 2007). KNIME[1] is a workflow based data mining tool. A process for analyzing data is created by combining single units (nodes). The main advantage here is that it is very easy to change individual subroutines in the workflow. For the framework the type of window configuration, modeling, parameters or the dissimilarity function can be exchanged by substituting one of the nodes with a different one. Hence the modularity of KNIME naturally fits the EVE framework.

Figure 7 shows an example of a workflow. With this simple construct it is very easy to test different distance measures or window combinations.

The first node is called the "Window Loop Start", in which the type of window combination as well as the respective parameters for the window sizes and step width are configured. This node has one input port, into which the whole data set is entered. The loop start node together with the loop end node are best described as a "while"-loop. All nodes between the two of them are executed for all consequent window combinations until the end of the data set is reached. Hereby the past window is always passed to the first outport and the present to the second outport. In each iteration the loop node combination moves the windows with the defined number of data points.

The models which can be created for abstracting the data are built in a second step before a dissimilarity measure is applied. In Fig. 7 the model creation and the dissimilarity measure are visualized by place holder nodes. The loop end finally collects the measured dissimilarities of each loop iteration for further analysis, which could, for example, be done by a control chart or visualization.

6 Experimental demonstration

Having introducing EVE and categorized some of the existing approaches in this Event Detection Framework, this chapter illustrates how the events presented in terminology Sect. 2 can be detected by EVE. Or, put differently, we aim to demonstrate how the framework needs to be configured in order to detect different kinds of events.

In the following we will use three different data sets which have all been generated using the Modular Data Generators discussed in Adä and Berthold (2010).[2] Each of the data sets generated is one dimensional and is generated

[1] http://www.knime.org

[2] The corresponding data generation workflow can be downloaded from the KNIME Example Workflows server at http://www.knime.org/example-workflows

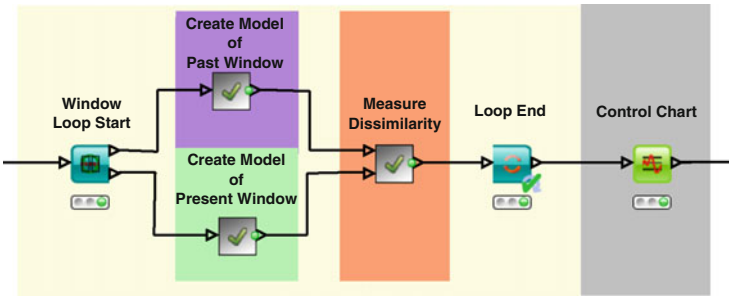


Fig. 7 This shows a KNIME workflow realizing EVE. The loop start provides the consequent window combinations. The dissimilarity between the two built models are finally collected by the loop end and evaluated by a control chart

over 500 consequent time steps. The various base lines are a Gaussian distribution.

6.1 Long-term event

To start with the most obvious event type, the first data set contains one abrupt event in the middle of the data. This can be seen in Fig. 8, where the mean of the value jumps from 0 to 5 after 250 time steps. A more gradual event can be seen in the second data set (Fig. 9). The value increases from time 130 to 185 remain constant for another 130 data points and finally reaches its original value at 370.

As we do not expect static data behavior, we use a SC window for the present, which is able to forget. Two possibilities (the SC and FC) are considered for the past. We use 5% of the data as the window size. A threshold of 1 is used for the final event detection. Hence we consider the following setup:

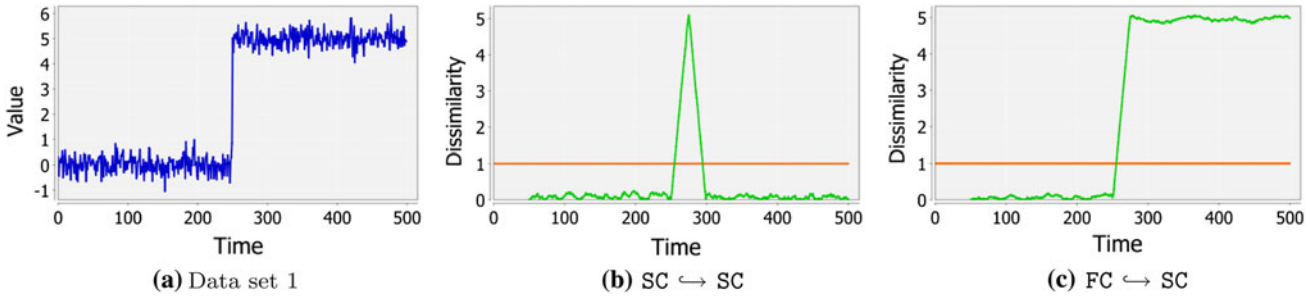
Window: SC $\hookrightarrow$ SC: $\vec{a}(t) = (t-50, t-26, t-25)$ and FC $\hookrightarrow$ SC: $\vec{a}(t) = (1, 25, t-25)$
Model: Mean μ
Dissimilarity: $d(t) = \frac{1}{25}\sum_{i=1\ldots25} |d_{a(t)_1+i} - d_{a(t)_2+i}|$
Detection: Warning if $d(t) > 1$

The results of this analysis can be seen in Figs. 8 and 9. The difference between the result of the two window configurations is relatively clear. The SC $\hookrightarrow$ SC adapts to the changed mean over time and hence for data set 1 an event is reported when the change happens but in contrast to the FC $\hookrightarrow$ SC combination dissimilarity decreases again. For data set 2 this effect is even more pronounced. The SC $\hookrightarrow$ SC reports two events: the rising and the fall of the dissimilarity function. The FC $\hookrightarrow$ SC only reports one larger event. Both results are not wrong, rather they provide a different kind of view on the data.

6.2 Multiple short-term event included in an overall gradual long-term event

As visualized in Fig. 10a the last data set contains an overall, long-term gradual event. In this example we are interested in the short-term events that are included inside this long-term event. To absorb the long-term event a linear model is used for the past. Dissimilarity is measured as the distance between the value predicted by the linear model and the true value. We tested two window combinations to detect the short-term event in the long-term event. The FC $\hookrightarrow$ SC and the FG $\hookrightarrow$ SC are both able to detect 6 out of the 7 introduced short-term events. The detection is done with a control chart (see Figs. 10b, c). However, the linear model would not be able to detect the overall gradual increase of the mean. To detect the overall change a SC $\hookrightarrow$ SC could be started parallel to this detection mechanism.

Window: FG $\hookrightarrow$ SC: $\vec{a}(t) = (0, t-2, t-1)$ and FC $\hookrightarrow$ SC: $\vec{a}(t) = (0, 25, t-1)$
Model: Linear Model $f(t) := mt + b$
Dissimilarity: $d(t) = |d_t - f(d_t)|$
Detection: Warning if $d(t) > \mu_{\{d_0,\ldots,d_t\}} + 3\sigma_{\{d_0,\ldots,d_t\}}$



(a) Data set 1 **(b)** SC $\hookrightarrow$ SC **(c)** FC $\hookrightarrow$ SC

Fig. 8 Data set 1 shows one abrupt event in the middle of the data. The SC $\hookrightarrow$ SC detects this rather nicely whereas the FC $\hookrightarrow$ SC would report one long event starting at the change (*blue*: data, *green*: dissimilarity, *orange*: detection threshold) (colour figure online)

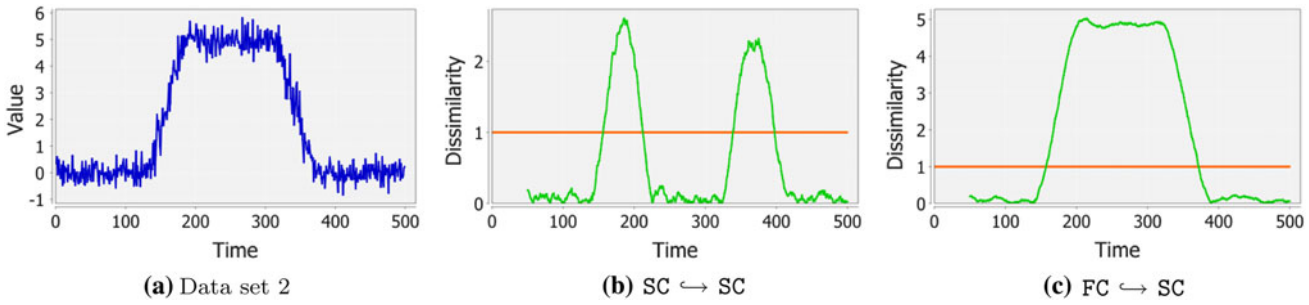


Fig. 9 A reoccurring gradual event has to be found in data set 2. Based on the desired behavior of the event detection methodology this can be detected as one big event (FC ↪ SC) or two short events at the beginning and the start of the change (SC ↪ SC) (*blue*: data, *green*: dissimilarity, *orange*: detection threshold) (colour figure online)

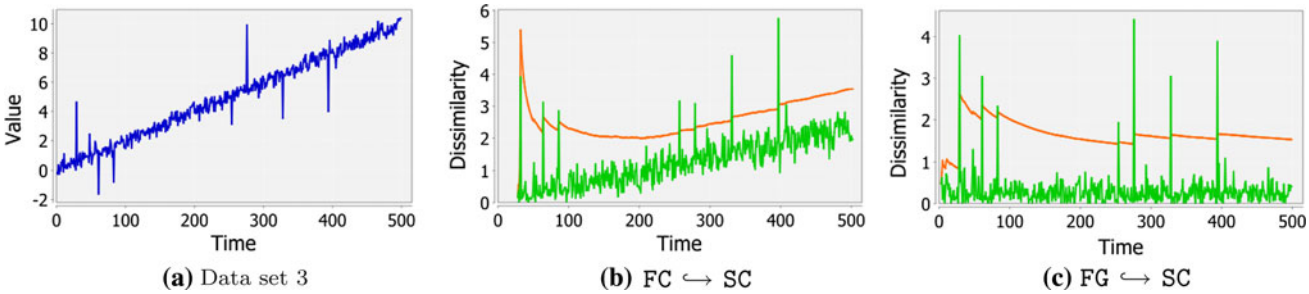


Fig. 10 Multiple short-term events included in an overall gradual long-term event. The FG ↪ SC shows better results for this analysis, as the linear model can be refined with every new data point. However, the FC ↪ SC with its small training size of 25 data points shows good results as well (*blue*: data, *green*: dissimilarity, *orange*: detection threshold) (colour figure online)

The goal of this section was to provide a quick insight how EVE can be used to detect different kind of events. Combining the three main ingredients in a harmonic way is important for achieving good results.

7 Relation to related frameworks

Some work has already been conducted on abstracting the idea of simultaneously finding outliers and changes in a continuous data flow. Most of it can be categorized using EVE, others make different data assumptions.

One of the earliest frameworks was presented by Yamanishi and Takeuchi (2002), in which they use a FG↪SC windowing combination. In this framework different models can be chosen for the past window, e.g. a Gaussian Mixture Model or a Time Series Auto regression model. A window of size 1 is used for the detection of outliers and of size T for changes. Different scoring functions are named to detect the events.

As already mentioned the change detection framework by Kifer et al. (2004) uses the FC↪SC combination. They extend this by recommending starting multiple instances with different window parameters. The overall event is reported if one of the combinations reports an event.

A "forward time slice density estimate" is compared to a "reverse time slice density estimate" in the change detection framework of Aggarwal (2003). Hence, an SC↪SC combination is used, both windows are modeled using density estimation and compared using velocity density function.

All of these frameworks fit directly into the EVE framework. The FLORA family of algorithms (Widmer and Kubat 1996), on the other hand, uses a model/description over all of the previous data, an FG window, and a window of currently trusted examples (SC or SG for FLORA2). FLORA is more specific than EVE, as it assumes binary classified data. And finally, the ADWIN (Bifet 2010) framework uses only one SG-window, but calculates statistics between any two subwindows, which is similar to our proposed dissimilarity calculation.

8 Conclusion

We have proposed EVE, a framework for formalizing the detection of various types of events in an ordered set of data or data stream. The framework not only enables model change and shift detection to be represented, but, by adjusting the level of granularity in the underlying framework, outliers, among others, can also be represented. We

believe this generic modeling approach will greatly assist convergence in the area of stream mining.

Acknowledgements This research was supported in part by the DFG under grant GRK1042 (Research Training Group "Explorative Analysis and Visualization of Large Information Spaces"). Special thanks to the anonymous reviewers for their constructive feedback.

References

Adä I, Berthold MR (2010) The new iris data: modular data generators. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining, KDD '10. ACM, New York, pp 413–422

Adä I, Berthold MR (2011) Unifying change-towards a framework for detecting the unexpected. In: Data mining workshops (ICDMW), 2011 IEEE 11th international conference on data mining

Aggarwal CC (2003) A framework for diagnosing changes in evolving data streams. In: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, SIGMOD '03. ACM, New York, pp 575–586

Baena-García M, Del Campo-Ávila J, Fidalgo R, Bifet A, Gavaldà R, Morales-Bueno R (2006) Early drift detection method. In: Fourth international workshop on knowledge discovery from data streams, vol 6. Citeseer, pp 77–86

Berthold MR, Cebron N, Dill F, Gabriel TR, Kötter T, Meinl T, Ohl P, Sieb C, Thiel K, Wiswedel B (2007) KNIME: The Konstanz Information Miner. In: Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007). Springer, Berlin

Bifet A (2010) Adaptive stream mining: pattern learning and mining from evolving data streams, vol 207. Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam

Bondu A, Grossin B, Picard M (2010) Density estimation on data streams: an application to change detection. EGC (Extraction et Gestion de Connaissances)

Chandola V, Banerjee A, Kumar V (2009) Anomaly detection: A survey. ACM Comput Surv 41(3):15:1–15:58

Dries A, Rückert U (2009) Adaptive concept drift detection. Stat Anal Data Min 2(5–6):311–327. doi:10.1002/sam.10054

Gama J (2010) Knowledge discovery from data streams. Chapman & Hall/CRC, Atlanta

Gama J (2012) A survey on learning from data streams: current and future trends. Prog Artif Intell 1:45–55. doi: 10.1007/s13748-011-0002-6

Gama J, Medas P, Castillo G, Rodrigues P (2004) Learning with drift detection. In: Bazzan A, Labidi S (eds) Advances in artificial intelligence SBIA 2004. Lecture notes in computer science, vol 3171. Springer, Berlin, pp 66–112

Hoens T, Polikar R, Chawla N (2012) Learning from streaming data with concept drift and imbalance: an overview. Prog Artif Intell 1:89–101. doi: 10.1007/s13748-011-0008-0

Japkowicz N, Myers C, Gluck M (1995) A novelty detection approach to classification. In: International joint conference on artificial intelligence

Kifer D, Ben-David S, Gehrke J (2004) Detecting change in data streams. In: Proceedings of the Thirtieth international conference on VLDB, vol 30, VLDB '04, pp 180–191. VLDB Endowment

Kullback S (1987) The Kullback-Leibler distance. Am Stat 41:340–341.

Markou M, Singh S (2003) Novelty detection: a review–part 1: statistical approaches. Signal Process 83(12):2481–2497. doi: 10.1016/j.sigpro.2003.07.018

Minku L, White A, Yao X (2010) The impact of diversity on online ensemble learning in the presence of concept drift. Knowl Data Eng IEEE Trans 22(5):730–742

Ross GJ, Adams NM, Tasoulis DK, Hand DJ (2012) Exponentially weighted moving average charts for detecting concept drift. Pattern Recognit Lett 33(2):191–198

Shewhart W (1980) Economic control of quality of manufactured product, vol 509. American Society for Qualit

Tsymbal A (2004) The problem of concept drift: definitions and related work. Computer Science Department, Trinity College, Dublin

Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. Mach Learn 23(1):69–101

Yamanishi K, Takeuchi J (2002) A unifying framework for detecting outliers and change points from non-stationary time series data. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 676–681

Zliobaite I (2009) Learning under concept drift: an overview. Technical report. Faculty of Mathematics and Informatics, Vilnius University, Vilnius