A MULTI-SCALE METHODOLOGY FOR EXPLAINING DATA STREAMS

Luka Stopar
Jozef Stefan Institute and Jožef Stefan International Postgraduate School,
amova 39, 1000 Ljubljana, Slovenia
Tel: +386 1 477-53-61
e-mail: luka.stopar@ijs.si

ABSTRACT

This paper presents a novel, multi-scale, framework, for the simultaneous analysis of multiple data streams, called StreamStory. The framework models the data streams as a hierarchical Markovian model by automatically learning states and transitios, and aggregating them into a hierarchy of Markov chains. This approach aims to compensate the gap between low-level streaming observations and high-level output/alerts which provide a value for higher levels of streaming data analysis, like inference and prediction, and provides ground for qualitative interpretation of the data.

1 INTRODUCTION

Sensory systems typically operate in cycles with a continuously time-varying component. Such systems can be characterized by a set of states, along with the associated state transitions. These states, on a high level, may include a “day” state, “night” state or maybe states with high and low productivity. For example when a pilot of an aircraft wishes to change the aircrafts heading, they will put the aircraft into state “banking turn” by lowering one aileron and raising the other, causing the aircraft to perform a circular arc. After some time, the wings of the aircraft will be brought level by an opposing motion of the ailerons and the aircraft will go into the “level” state.

Such high-level states can further be split into lower-level states, giving us a multi-resolution view of the system, allowing us to observe the system on multiple aggregation levels. For example, a “banking turn” state can be split by the aircrafts roll and angular velocity, resulting in perhaps three states: “initiate turn”, “full turn” and “end turn”.

StreamStory models the monitored system as a hierarchical Markovian process by automatically learning the typical lowest-level states and transitions, and aggregating them to obtain a hierarchy of Markovian processes. Such a model allows users to observe the monitored system in a unique way and provide a understanding of its dynamics.

Furthermore, we divide the input streams into two sets: observation set and control set. The observation set of paramters are the paramters that tell us the state of the system and, we assume, cannot influence its dynamics. These are parameters that users cannot directly manipulate, like aircraft tilt. They are used to identify, and aggregate, low-level states, detect outliers (anomalies) and determine the current state of the system. In contrast, users can directly manipulate parameters in the control set. These are parameters like the angle of each aileron, and may directly influence the behavior (observation set) and performance of the system. For example when an operator in a steel factory sets the cooling temperature to a high value, the product will take longer to go from state “extremely hot” to state “warm”. As such, control parameters may also influence the occurrence, and expected time, of undesired states that may be associated with some undesired event. Our approach uses control parameters to model state transitions, allowing us to observe the dynamics with respect to the current configuration and gives the user insight into the expected bynamics before changing the configuration.

To implement the framework, we subdivide it into four components: (1) state identifier, (2) hierarchy modeler, (3) state assistant and (4) transition modeler, each responsible for its own subtask and explained in the next section.

The remainder of this paper is structured as follows. Section 2 presents on overview of the systems architecture. The user interface is presented in section 3 and, finally, we provide conclusions and acknowledgements in sections 4 and 5 respectively.

2 STREAMSTORY ARCHITECTURE

This section presents the architecture of the StreamStory system. StreamStory operates in two modes: offline and online. In offline mode the system consumes a batch of the data streams and learns its hierarchical Markovian representation (identifies states, constructs a hierarchy and models transitions). Once the model is learned, it can be applied to the data streams in real-time and offers prediction and anomaly detection services. The component also includes a web-based user interface (UI), where the users can explore and interact with the model.

We continue our discussion with a picture of the architecture in Figure 1.

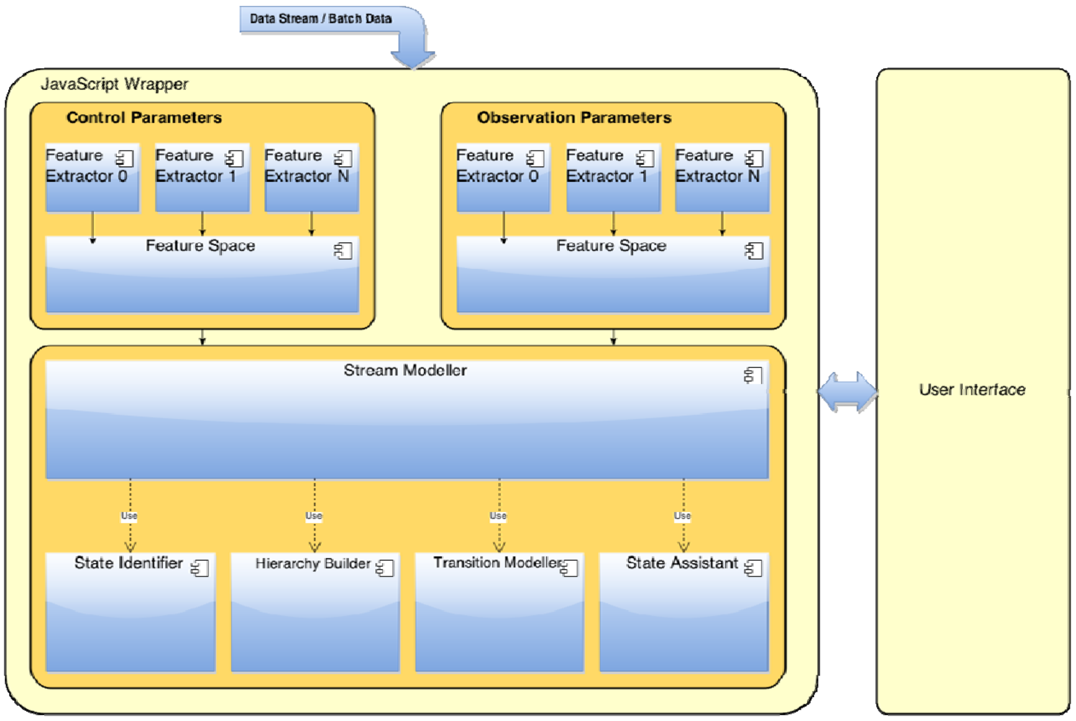

Figure 1: StreamStory architecture.

As shown, the component is a JavaScript wrapper arount two feature spaces (one for each set of parameters) and a stream modeler component, and interacts with the user interface using RESTful web services.

The two feature spaces are responsible for the transformation of the data into (and out of) feature vectors, later used by the algorithms. Each feature space consists of several feature extractors (one for each parameter) which are responsible for transforming a single parameter into a form suitable for machine learning algorithms. The responsibility of the feature space is then to concatenate the ouputs of all the feature extractors into a single feature vector.

The third component is the stream modeler, which is the core component of the system. It is responsible for state identification and assistance, modeling the hierarchy, prediction and anomaly detection. The modeler delegates these tasks to four components: state identifier, hierarchy builder, state assistant and transition modeler, which are explained in the following subsections.

State Identifier

The first of these components is the state identifier. In offline mode it is responsible for the identification and construction of the lowest level states. Once these states are constructed it computes and stores their statistics for further use in the user interface (UI) and detection of anomalies. In online mode, the state identifier is used to identify the current state of the system, from the feature vectors, and for low-level anomaly detection.

To identify the states, the state identifier uses the DPMeans algorithm [1]. The algorithm takes one input parameter λ, which represents the maximum goemetrical radius of each state and is used to control the number of output states. The algorithm is very similar to K-Means [2]. It starts by randomly selecting one of the feature vectors as the initial centroid. In each iteration, it then assigns the feature vectors to their nearest centroid, but unlike K-Means if the vector is outside the radius of all the centroids, it is used to form a new centroid.

When used in online mode, the state identifier is responsible for identifying the state to which the current feature vector belongs. This is done by assigning the feature vector to the state with the nearest centroid. If the feature vector falls outside the radious of all the states, it is marked as an anomaly.

Hierarchy Modeler

Once the low-level states are identified, they are aggregated into a hierarchy. This is precisely the task of the hierarchy modeler. The hierarchy modeler consumer a set of states (centroids), aggregates them into a hierarchy and stores it in two arrays. The first array encodes the topology of the hierarchy, by storing at index *i* the index of *i*-ths parent. The second array stores the hight (level) of each state.

To compute the topology, the hierarchy modeler uses one of several agglomerative clustering strategies: single link, complete link or average link [3].

In online mode the hierarchy modeler is responsible for two tasks: (1) given the current lowest level state, finding its parent states on specific levels and (2) given a state in the middle of the topology, finding all its lowest level successors later used by the transition modeler.

State Assistant

The state assistant is responsible for assisting the users in identifying the meaning of states. For example, when the user clicks on a state in the UI, the state assistant highlights the attributes that are most typical for the state. This is achieved by extracting weights of individual features from a logistic regression model [4] learned by classifying feature vectors of one state (positive label) against feature vectors of all the other states (negative labels). To balance the class distribution, the component samples the larger set.

Transition Modeler

The final component in our framework is the transition modeler. As the name suggests, the transition modeler models transitions between states. It does so by using a continuous time Markov Chain framework [5]. On the lowest level, the model is defined with a transition rate matrix Q, where the element at posisiton (*i*,*j*) represents the rate of going from state *i* to state *j*.

To be able to model the hierarchical dynamics, we need to be able to compute the transition rate matrix for Q_l on each level l of the hierarchy. The transition modeler only stores the lowest-level transition matrix and uses it to construct higher level matrices on the fly. To achieve this, it needs to know which low-level states to aggregate. It gets this information in the form of state-sets $\{S_i\}_i$, where each state-set corresponds to an aggregated state on level l. It then computes the transition rate matrix on level l using the following formula:

$$Q_{S_i S_j} = \frac{\sum_{k \in S_i} \pi_k \sum_{h \in S_j} Q_{kh}}{\sum_{k \in S_i} \pi_k}$$

Where $\pi = (\pi_k)_k$ is the stationary distribution of the lowest level Markov chain and is found as a normalized non-trivial solution to the following system of equations: $\pi Q = 0$.

3 USER INTERFACE AND USER INTERACTION

This section presents the user interface (UI) of the StreamStory system. The UI is designed to allow the user to observe the dynamics of the monitored system, as well as its current state, and allow the user to configure the underlying model, prediction and anomaly detection services. It visualizes the monitored system as a hierarchy of states, along with associated transitions, and offers several services that allow the user to identify the meaning of states as well as a messaging service which displays notifications about predictions and anomalies.

Figure 2 shows a screenshot of our web-based user interface. The UI consists of four main components: visualization component, state information component, notifications component and model configuration component. These will be explained in the next subsections.

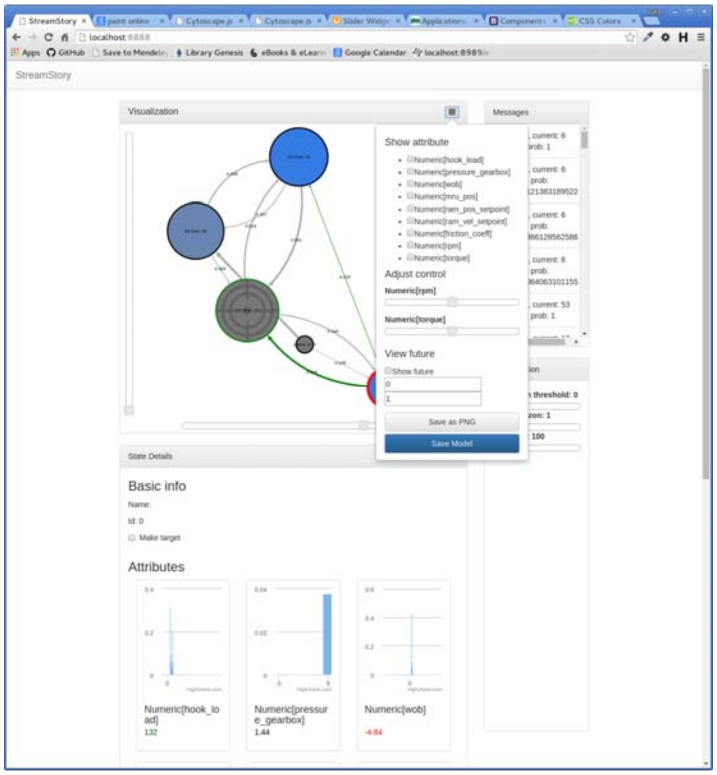

Figure 2: StreamStories web-based user interface .

Visualization Component

The visualization component is the main component of the UI, as it visualizes, and allows the end-users to explore and interact with the model.

States and transitions are represented by a circles and arrows respectively. The size of a circle is proportional to the fraction of time that the system spends in the associated state. Each state displays its identifier (or name if already defined) and the average time spent in the state after arrival. The current and previous states are highlighted with a green and red border respectively, and the most likely future states have a blue background, with the blue component proportional to the probability of jumping into the state. The states with a target in the background are "target" states and the state with the bold bolder is the selected state and its details are shown in the state details component, described in the next subsection. When the system changes states, the UI is automatically updated through web sockets. The thickness of each arrow is proportional to the probability of the corresponding transition, also displayed in the middle of the arrow.

When first opening the UI, the model is shown at the top level with only 2-3 states. The user can use the scroll function to scroll into lower-level states. When scrolling up/down the hierarchy, the states are automatically merged/split.

A popup menu in the top right corner of the component allows the user to:

- Select a target feature: When selecting a target feature, all the states are coloured proportionally to the mean value of the feature in that state. For example, when selecting ambient temperature, the states with higher temperature will become greener than states with lower temperature.
- Observe state probabilities at future times: When moving the spinner at the bottom of the menu states get colored proportionally to the probability of the system being in that state at the appropriate future/past time.
- Simulate control paramters: Using the first group of sliders, the user can simulate a change in parameters in the control set. When adjusting one of the parameters, the transition probabilities are recalculated along with the corresponding holding times and the component automatically redrawn.

State Details Component

As the name suggests, the state details component shows detailed information about the selected state. It provides basic information, like name and id, as well as more detailed information like: the average values of parameters in the state along with their distribution (shown as histograms) and the most typical parameters for the state, which are highlighted (red or green) according to their relevance.

The state details component also allows the user to mark the state as a "target". When a state is marked as a target, notifications about the expected arrival times are displayed in the notifications component presented next.

Notifications Component

The notifications component displays messages to the end-user. These messages include information about the detected anomalies and predictions of arrival into target states. When detailed information about the message is available, the message is clickable and, upon clicking, its details are shown in a popup window.

Model Configuration Component

The model configuration component allows the user to modify the parameters of the underlying model. In the current version, the user can adjust parameters corresponding to the prediction of target states. These include the prediction probability threshold, and the time horizon usied in the calculation of the probability.

4 CONCLUSION

In this paper we presented a novel system for modeling and visualizing data streams called StreamStory. The StreamStory system integrates several machine learning algorithms to model the incoming data streams as a hierarchical Markovian process. As such the system supports several functionalities, including: future state extrapolation, anomaly detection and allows the users to uniquely interprate the stream structure. The system includes a web-based user interface which allows the interaction and exploration of the model.

5 ACKNOWLEDGMENTS

This work was supported by the Slovenian Research Agency and the ICT Programme of the EC under project ProaSense (FP7-ICT-2013-10-612329).

REFERENCES

[1] K. Brian and M. I. Jordan, "Revisiting k-means: New Algorithms via Bayesian Nonparametrics," in *Preceedings of the 29th International Conference on Machine Learning*, 2012.

[2] I. H. Witten, E. Frank and M. A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, Burlington: Elsevier Inc., 2011.

[3] F. Murtagh and P. Contreras, "Algorithms for Hierarchical Clustering: An Overview," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery,* vol. 2, no. 1, pp. 86-97, 2012.

[4] A. J. Dobson, An Introduction to Generalized Linear Models, Boca Raton: Chapman & Hall/CRC, 2002.

[5] J. Norris, Markov Chains, Cambridge: Cambridge University Press, 1997.