A Multi-Scale Methodology for Explaining Data Streams

Luka Stopar, Marko Grobelnik, Dunja Mladenic
{luka.stopar,marko.grobelnik,dunja.mladenic}@ijs.si
Outline

• Introduction
• Implementation
• Use-cases
• Further work
Introduction

- Sensory systems typically operate in cycles with a continuously time-varying component
  - Aircraft in flight
  - Manufacturing systems
  - Weather

- Such systems can be characterized by a set of states along with state transitions
  - “Day” and “night” state
  - States according to an aircrafts roll
  - States with high and low productivity
Introduction

• Such high-level states can be further decomposed into lower-level states:
  – “Day” and “night” into “morning”, “midday”, “evening”, “midnight”
  – Aircrafts turn into $0^\circ$ roll, $10^\circ$ roll, $20^\circ$ roll

• We propose a methodology for modeling such system and present its implementation

• We call our system StreamStory
Introduction - StreamStory

- StreamStory models the monitored system by:
  - Consuming multiple data streams and modeling them as a hierarchical Markovian process
  - Automatically learning the systems typical states and transitions
  - Aggregating states into a hierarchy to obtain a multi-resolution view

- Describing the underlying data and its dynamics in a qualitative manner
  - Helps to compensate the gap between low-level observations and high-level outputs/alerts

- By modelling transition it is able to predict future developments in the real-time data streams
• What if the pilot was to reduce the aircrafts speed during a banking turn?
• StreamStory splits the data streams into two sets of attributes

**Observation set:**
- Operators cannot directly manipulate: ambient temperature
- Cannot influence the systems dynamics
- Used to identify states

**Control set:**
- Attributes like injection pressure that the operator can adjust
- Influence the systems dynamics
- Used to model transitions

• StreamStory allows the user to observe the expected behavior of the system under a modified configuration
Outline

• Introduction
• Implementation
• Use-cases
• Further work
Implementation - Overview

• Take a set of data streams and join them so they are sampled at the same timestamps
• Resample the data streams
• Join the data streams into two sets of feature vectors
• Cluster the first set of feature vectors and use the clusters as states
• Generate a hierarchy of states
• Model state transitions based on the second set of feature vectors
  – Compute transitions for higher levels out of low level transitions online
Implementation

- **StreamStory** is implemented as a client-server application
  - Core functionality written in C++
  - External communication written in server-side JavaScript
  - Web-based user interface
- Each user can build several models
  - For exploration
  - For real-time application
StreamStory Architecture

Server

- Offline model store
  - Model 1
  - Model 2
  - Model N

- Online model store
  - Model 1
  - Model 2
  - Model N

- Data store
  - Data store 1
  - Data store 2
  - Data store N

User Interface

Online data store

- Store 1
- Store 2
- Store N

Merger

Data store

User Interface

Database

Data Source 1

Data Source 2

Data Source N
Model Architecture

Data Stream / Batch

JavaScript Wrapper

Observation Attributes

Feature Extractor 1
Feature Extractor 2
Feature Extractor N
Feature Space

Control Attributes

Feature Extractor 1
Feature Extractor 2
Feature Extractor M
Feature Space

Stream Modeler

State Identifier
Hierarchy Builder
Transition Modeler
State Assistant

User Interface

WebSocket

Req/Res
Stream Modeler

- The core component of the StreamStory system
- Delegates its tasks to four sub-components:
  - State Identifier
  - Hierarchy Builder
  - Transition Modeler
  - State Assistant
- Acts as a glue between different functionalities
State Identifier

• Responsible identification and construction of lowest-level states
  – State construction done by clustering the input streams
  – Uses DPMeans as the clustering algorithm
  – Stores statistics of each state for further use: visualization, anomaly detection
  – Computes 2D coordinates of each state: Multi-Dimensional Scaling (MDS)

• In online mode, the state identifier is responsible for:
  – Identification of the current lowest-level state
  – Anomaly detection: clustering based technique
Hierarchy Builder

- Once the lowest-level states are constructed, the hierarchy builder aggregates them into a hierarchy
  - Supports three agglomerative clustering strategies: single link, complete link and average link

- The hierarchy is encoded using 2 arrays:
  - Topology array
  - State level array
Hierarchy Builder – Finding Roots on Specific Height

- The topology array stores, at index $i$, the index of $i$-th’s parent
- To be able to model the Markov chain on a specific level, the hierarchy builder must determine which states are aggregated on that level
  - These aggregated states are stored into state sets $S_i$
  - State sets are computed using the following algorithm

1. $n \leftarrow$ total number of states
2. $S \leftarrow \{i \mid i \text{ resides on height } l\}, h \leftarrow$ array encoding the topology
3. repeat:
   1. for $i = 1, \ldots, n$ do
      1. if $h_i \notin S \land h_i \neq h_{h_i}$ do $h_i \leftarrow h_{h_i}$
4. while elements of $h$ change

$$S_1, S_2$$
Transition Modeler

- Models transitions between states
  - Using a continuous-time Markov chain framework
- Determines:
  - Size of each state: stationary distribution $\pi_i$
  - Average staying time: $t_{avg} = \frac{1}{-q_{ii}}$
  - Transition probabilities: $p_{ij} = \frac{q_{ij}}{-q_{ii}}$
- Given states aggregated into state sets $\{S_i\}_{i=1}^n$ computes the aggregated Markov chain using the following formula:
  - Preserves the stationary distribution $\pi$

$$q_{S_iS_j} = \frac{\sum_{k \in S_i} \pi_k \sum_{l \in S_j} q_{kl}}{\sum_{k \in S_i} \pi_k}$$
Transition Modeler – Modelling Transitions

- Attributes in the control set influence state transitions
- Transition probabilities are estimated using logistic regression models
  \[ p_{ij}(\Delta t) = \left(1 + e^{-\beta_{(ij)}x_k}\right)^{-1} \]
- Transition rates are calculated from probabilities
  \[ q_{ij} = \frac{p_{ij}}{\Delta t} \]

| state: | time |\| 1\| 1\| 2\| \\
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>rpm</td>
<td>[33.3]</td>
<td>[40.1]</td>
<td>[52.3]</td>
</tr>
<tr>
<td>class trans.1</td>
<td>[1]</td>
<td>[0]</td>
<td>[0]</td>
</tr>
<tr>
<td>rpm</td>
<td>[33.3]</td>
<td>[40.1]</td>
<td>[52.3]</td>
</tr>
<tr>
<td>class trans.2</td>
<td>[0]</td>
<td>[1]</td>
<td>[1]</td>
</tr>
<tr>
<td>rpm</td>
<td>[33.3]</td>
<td>[40.1]</td>
<td>[52.3]</td>
</tr>
<tr>
<td>class trans 3</td>
<td>[0]</td>
<td>[0]</td>
<td>[0]</td>
</tr>
</tbody>
</table>
State Assistant

- Responsible for assisting users in identifying the meaning of states
  - Highlights attributes that are specific for that state
  - Achieved by extracting weights from a logistic regression model
  - On every height we classify instances of one state against the instances of all other states
  - Random sampling to balance the datasets
User Interface

• Web-based
  – HTML + JavaScript

• Consists of 4 panels
  – Visualization panel
  – State Details panel
  – Messages panel
  – Configuration panel
User Interface – Visualization Panel

- Main component of the user interface
- Shows the hierarchical Markovian model
  - Can use the zoom function to expand contract states
- Displays:
  - Circles: states on the current level (with ID/name and average staying time), sizes proportional to the stationary distribution
  - Arrows: state transitions (with probability)
- Highlighted states:
  - Current state: green
  - Most likely future states: blue
  - Previous state: red border
  - Selected state: bold border
- Other features:
  - Select target attribute: states are colored according to the value of the target attribute (green, red)
  - Show probability distribution at future/past times: states get colored according to the probability of the system being in them at a predefined past/future time
User Interface – State Details Panel

• Shows detailed information about the selected state
  - Identifier of the state
  - Name of the state
  - Average values of both sets of attributes
  - How specific each parameter is for that state
  - Distribution of attributes in a state

• Other features:
  - Manual adjustment of control attributes: the user can manually adjust the values of control attributes in the selected state, after adjustment the visualization is automatically updated
  - Naming the selected state: the user can give the selected state a name, which will be displayed in the visualization panel instead of the identifier
  - Mark the state as “target”: predictions about arrival times into target states are shown in the messages panel
Outline

• Introduction
• Implementation
• Use-cases
• Further work
Use-Cases

• StreamStory supports several use-cases:
  – Data exploration / Exploratory data mining
  – Equipment / production monitoring
  – Anomaly detection
  – Root cause analysis
  – Prediction

• Deployed on two case-studies
  – Monitoring drilling equipment
  – Monitoring car lens production
Outline

• Introduction
• Implementation
• Use-cases
• Further work
Further Work

• Transition modeling
  – Transitions are only model at the lowest-level states
  – Include higher-level information

• Implementing new features
Thank You

- Questions?