SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF COMPUTER SCIENCE

Drift Anomalies: A Intelligent Approach to Visualizing Data using Scatter and Autoencoders

PATH OF EXCELLENCE

Advisor:
Luca Podo

Student:
Alessandro Di Patria

Academic Year 2022/2023

Contents

1 Introduction

Understanding and detecting shift anomalies in time series data can be quite challenging, as these anomalies persistently deviate from the typical trends [1]. Unlike isolated point anomalies, which are easier to spot, shift anomalies disrupt the overall pattern, making their identification crucial across various domains. In this blog post, we introduce an innovative visual approach that combines a straightforward deep learning model, specifically an autoencoder, with scatter plots to enhance the visualization and detection of shift anomalies.

Our approach leverages the power of deep learning models to capture complex patterns and representations within the data. By integrating deep learning algorithms, visual analytics techniques, and domain expert decision processes, we offer a comprehensive solution. The compact architecture of the autoencoder enables effective learning of the time series' latent features, leading to a better understanding and identification of shift anomalies [2]. Additionally, the autoencoder's ability to reconstruct the input data helps to highlight the discrepancies caused by these anomalies, making them visually distinguishable in scatter plots.

To enhance the interpretability of shift anomaly detection, we propose exploiting the inherent visual perception capabilities of human observers. We achieve this by emphasizing anomalies and reducing the emphasis on normality. Empowering domain experts to visually detect and analyze anomalies using our approach facilitates more accurate and context-aware decision-making. It should be noted that our approach serves as a supportive visual tool for exploratory data analysis rather than a fully automated anomaly detection model.

The pipeline described in this paper involves an anomaly detection and visualization method for time-series data. The dataset consists of two columns: a "timestamp" column representing the time of each action or event, and a "values" column containing numeric measurements.

The process begins by loading the dataset, which is then divided into two distinct parts: the training data, encompassing the normal observations, and the unseen data, which is intended for visualization purposes. From the normal data, sub-sequences of a specified length, are extracted. These sub-sequences are transformed into an action matrix labeled as "A".

Moving forward, the sub-sequences originating from the normal data are fed into an autoencoder. This autoencoder architecture involves several stages, including a 1D convolutional layer, a downsampling step utilizing max-pooling, an up-sampling layer, another convolutional layer, and ultimately a final layer dedicated to reconstruction. This intricate architecture is adept at capturing the fundamental patterns inherent within the normal data.

The subsequent stage involves a similarity comparison between different sub-samples,

derived from the transformed matrix "A'". Employing a sliding window technique, the method computes similarity scores by contrasting these sub-samples. Mean vectors of similarity scores are then calculated, resulting in the creation of a reference representing the normality of the data. To establish a flexible threshold for detecting anomalies, the mean and standard deviation of this vector of mean scores are computed.

Considering the challenge posed by the reduction in matrix size, a solution is implemented involving the padding of similarity score vectors. This is achieved by sampling values from the uniform distribution of both normal and unseen score vectors.

Next, the focus shifts to the analysis of unseen data. By passing the unseen data through the autoencoder, its latent representation is obtained. Deviations from the established normal patterns yield distinct representations, which are meticulously examined and subsequently visualized.

The method culminates in the presentation of the data through scatter plots. This form of visualization effectively enhances anomaly detection by accentuating and isolating anomalous behaviors, thus distinguishing them from the normal patterns that are filtered out.

In summary, our novel approach combines deep learning with scatter plots to detect and understand shift anomalies in time series data. By integrating human expertise and emphasizing anomalies, we provide a valuable tool for domain experts to make informed decisions based on visual data exploration.

2 Background

2.1 Drift anomalies

Drift anomalies in time series data refer to the gradual and sustained changes in the underlying statistical properties of the data over time. Unlike sudden spikes or drops, drift anomalies occur progressively, making them harder to identify. These changes can impact the overall behavior of the data, leading to shifts in trends, seasonality, and variations.

Drift anomalies can arise due to both internal and external factors. Internally, changes in system parameters, variations in data generation processes, or shifts in user behavior can lead to drift anomalies. Externally, factors like economic changes, policy shifts, or environmental events can influence the time series data. Additionally, technical issues such as sensor degradation, data transmission errors, or data preprocessing errors may introduce drift anomalies.

Several real-world case studies demonstrate the significance of drift anomaly detection in time series data. Examples include financial market analysis, climate change monitoring, and industrial process control. In each case, identifying and addressing drift anomalies have proven crucial for accurate forecasting and decision-making.

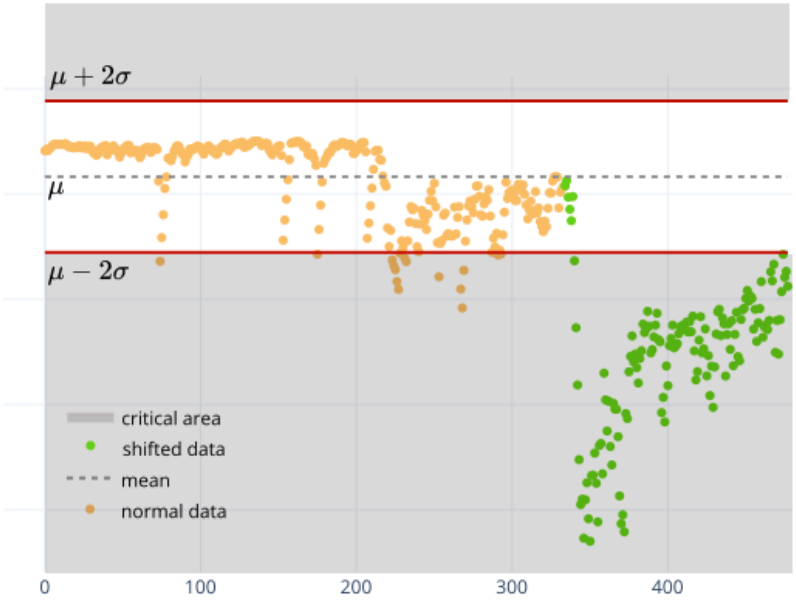

Figure 1: A shift anomaly in a sample time series, represented by a green data point that begins to deviate from the average or normal behavior by changing its values.

2.1.1 Real-World Shift Anomaly

The Heatwave Scenario: To gain a better understanding of drift anomalies, let's take a look at a real-world case involving the measurement of temperature values in a region

renowned for its stable climate and consistent seasonal temperature fluctuations, such as the Amazon rainforest. Imagine a situation where a high-pressure system suddenly settles over the area, giving rise to a heatwave. Consequently, the region experiences an unforeseen and substantial rise in temperature for an extended period, surpassing all historical temperature records for that particular season. This abnormal shift in temperature creates extreme heat conditions, resulting in numerous challenges like heatwaves, increased energy demands, and potential risks to human health and the environment. This example vividly illustrates how drift anomalies can have far-reaching consequences beyond just financial contexts.

2.2 Autoencoder

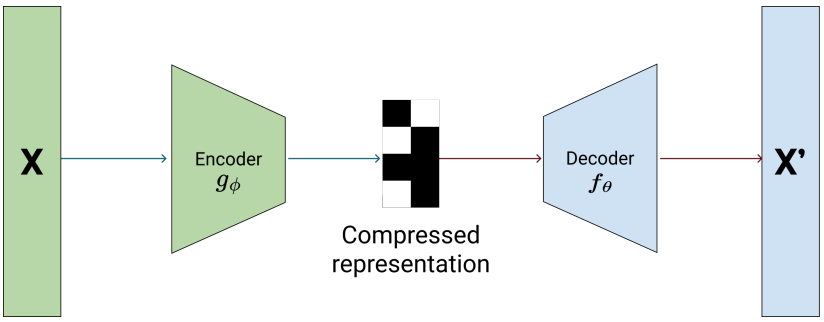

Figure 2: Autoencoder Structure

The autoencoder is a neural network architecture comprising two primary components: the encoder and the decoder. Unlike conventional supervised learning models, the autoencoder operates unsupervised, discovering data patterns without explicit labeling.

The encoder's role is to compress the input data into a lower-dimensional latent space, capturing its essential features. Conversely, the decoder takes this compressed information and reconstructs it, generating an output that closely resembles the original input.

The bottleneck layer acts as the interface between the encoder and decoder, holding the dense representation of the compressed data. It contains critical information in a lower-dimensional form.

In the autoencoder structure, we denote the encoder function as g_ϕ (where ϕ represents parameters) and the decoder function as f_θ (where θ represents parameters). The latent space is represented as $z = g_\phi(x)$ [3], where x is the input data. The reconstructed information is denoted as $x' = f_\theta(g_\phi(x))$. During training, the functions' parameters are adjusted to minimize the loss function, aiming to generate an output closely resembling the input data.

By iteratively adjusting the parameters, the autoencoder learns to compress and reconstruct input data, capturing underlying patterns. The ultimate goal is to obtain

an output that closely resembles the original input, effectively learning a compact representation in the latent space.

3 Methodology

3.1 Description of the input

The method requires a dataset with two columns, specifically:

- timestamp column must contain the observation timestamp of each action or event. This column serves as a temporal reference and enables chronological ordering of the data instances.
- values column must contain the corresponding value of the variable measured at the given moment in time. This column captures the numeric information associated with each action or event. The nature of the variable may vary depending on the problem domain, ranging from numerical to categorical. In this blog post, we only deal with numerical cases.

In certain scenarios, the actions or events may span across two consecutive days, with their start times falling on the boundary of midnight. To ensure proper handling of such edge cases, a preprocessing step is necessary to split these instances into two separate rows. This division accurately represents the time sequence and prevents potential data inconsistencies or biases due to the spanning events.

In this example we analyzed datasets from *Numenta Anomaly Benchmark* [4] which is an open-source benchmark created by Numenta, an artificial intelligence research company. The NAB dataset consists of a collection of real-world time-series data with labeled anomalies, making it suitable for evaluating the performance of various anomaly detection algorithms.

3.2 Data transformation

After loading the dataset, it is split into two distinct sections: the training data, containing the user-selected standard information, and the unseen data, containing fresh data specifically for visualization purposes. Following this, the suggested approach creates data sub-sequences with a specified length t from the dataset and converts them into the action matrix, represented as matrix $\mathbf{A}$. The height of the matrix depends on the size of the input data.

The symbol "t" signifies a hyperparameter within the presented methodology, the manipulation of which permits the controlled observation of the progressive evolution of visualizations.

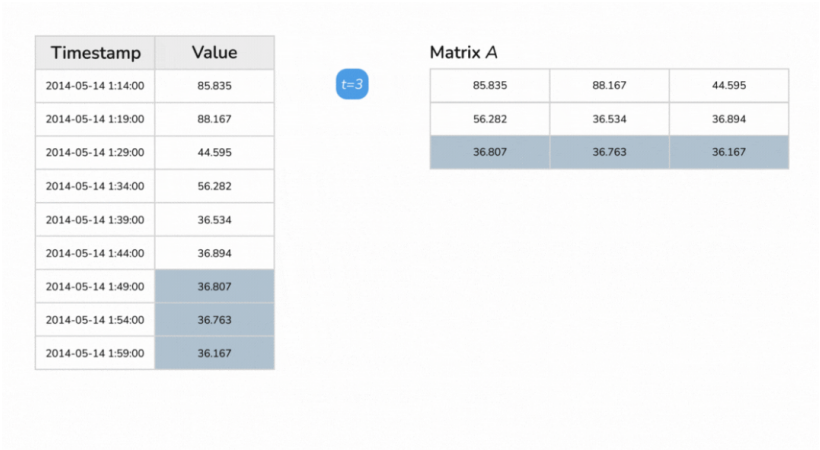

Figure 3: Split data in sub-sequences of length t. In our case t = 20

3.3 Autoencoder architecture

In this process, the aim is to capture the underlying concept of normality in a given dataset. The approach involves using an autoencoder, a type of neural network, for this purpose.

The input data subset is divided into individual sub-sequences, and each of these sub-sequences is fed into the autoencoder. The autoencoder's main task is to learn a compressed representation of the normal data patterns, capturing their essential features.

The model architecture takes 1D input data with a shape of (1, t), where 't' represents the length of the input sequence. The input data is then processed through a 1D convolutional layer with 8 filters, each of size 4, and a Rectified Linear Unit (ReLU) activation function. This step helps in extracting relevant features from the input data.

The resulting feature map is further downsized by applying a max-pooling layer with a pool size of 2. Max-pooling helps in reducing the spatial dimensions of the data while retaining the most significant information, making the subsequent computations more efficient.

Next, the encoded representation, which is the compressed form of the input data, is passed through an up-sampling layer. This layer increases the dimensionality of the data back to its original size, effectively expanding the compressed representation.

The up-sampled feature map is then processed by another 1D convolutional layer with 8 filters, each of size 4, and a ReLU activation function. This step aims to further refine the encoded representation and bring it closer to the original input data.

Finally, the reconstructed output is obtained by applying a 1D convolutional layer with 1 filter of size 3 and a sigmoid activation function. Overall, this model follows an autoencoder architecture, aiming to compress the input data in the encoding layers and then reconstruct it in the decoding layers.

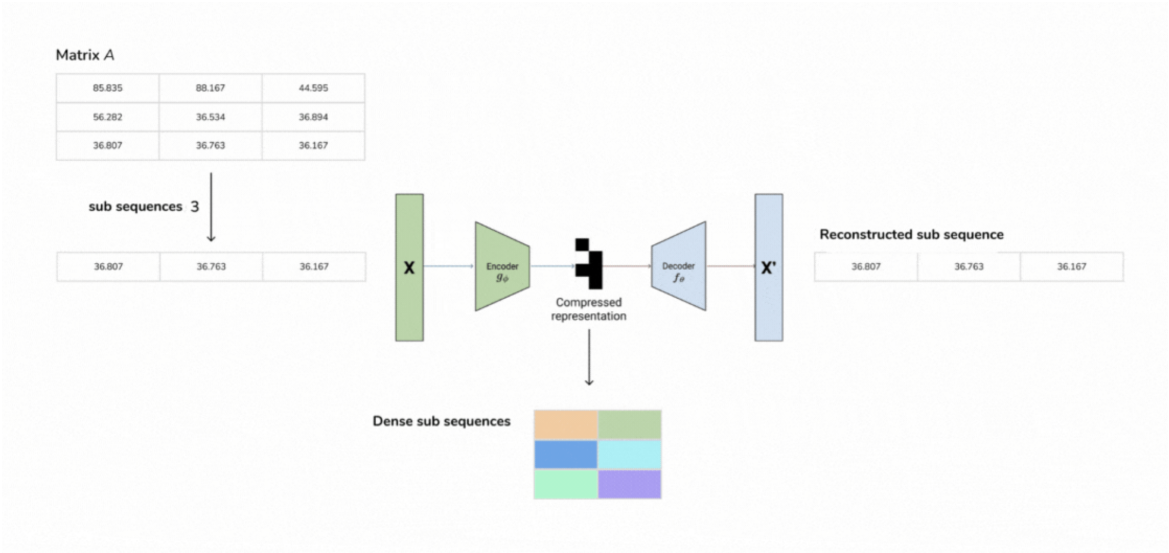

Figure 4: The process of reconstruction of the original input after being compressed and processed by autoencoder

We denote by A' the resulting matrix representing the final output of the described autoencoder process. The matrix represents the reconstruction of the original input after being compressed and processed through the various layers of the autoencoder.

3.4 Profile similarity comparison

The following method outlines a series of steps to analyze a given matrix $\mathbf{A}'$ and generate a reference vector for data normality, allowing for visual comparisons of new, unseen data. The process involves generating sub-samples, calculating similarity scores, creating a mean scores vector, and finally determining thresholds to detect deviations from the expected data behavior.

1. **Sub-sample Generation**: The method starts by creating sub-portions of the matrix $\mathbf{A}'$ with dimensions (t, l). It begins by selecting the first sub-sample from $\mathbf{A}'$, which is represented by the portion of $\mathbf{A}'$ from index 0 to l. The hyperparameter "l" represents the sub-sample dimension that can be adjusted to observe how the visualization evolves.

2. **Similarity Vector Scores Generation**: Once a sub-sample is chosen, a comparison process is initiated. $\mathbf{A}'[0 : l]$ is compared with all other sub-samples by sliding one row at a time on $\mathbf{A}'$. Initially, $\mathbf{A}'[0 : l]$ is compared with $\mathbf{A}'[0 : l]$, essentially representing a self-comparison for the first sub-sample. The comparison process then proceeds by moving one row down, comparing $\mathbf{A}'[0 : l]$ with $\mathbf{A}'[1 : l]$, and generating similarity percentages for each comparison. This iterative process continues until $\mathbf{A}'[0 : l]$ is compared with all sub-samples from $\mathbf{A}'$, resulting in a similarity vector containing the computed scores. This entire procedure is

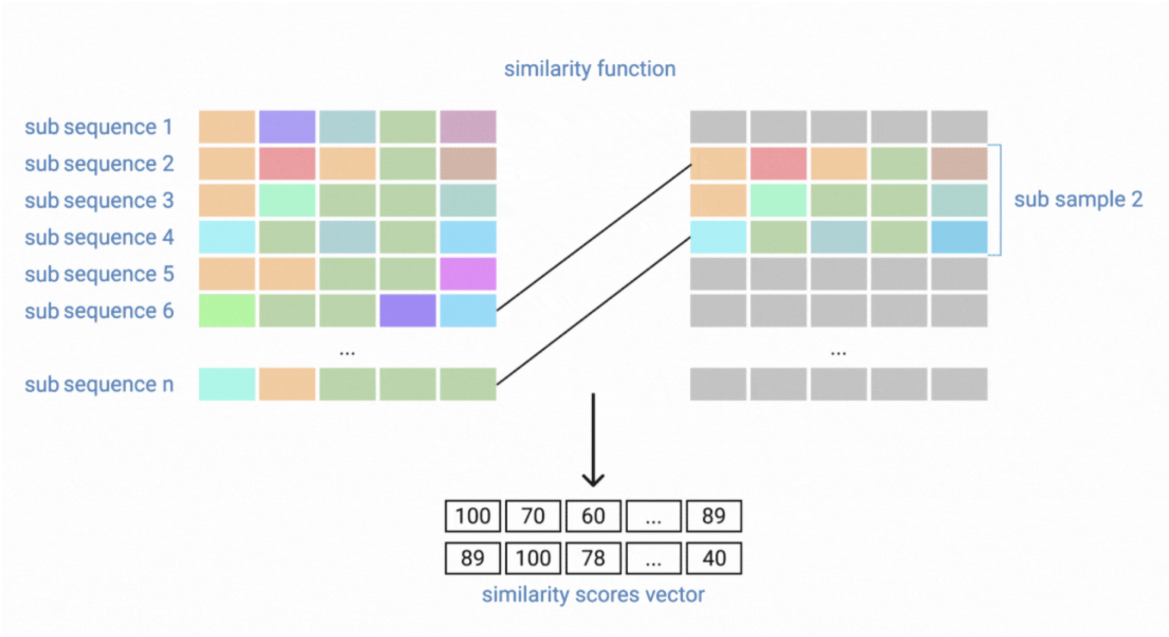

Figure 5: Visualization of the Behavior of the Similarity Function

repeated for all the sub-samples that can be chosen from $\mathbf{A}'$. Consequently, the algorithm generates a matrix, denoted as the "scores matrix," where each row represents a similarity vector for a specific sub-sample.

3. **Mean Scores Vector Generation**: The subsequent step involves calculating the mean value for each row of the scores matrix. This process generates a vector referred to as the "data normality reference." This vector serves as a reference for normality, and it allows the user to visually compare the trend of new, unseen data with this reference vector.

Finally, the algorithm compute the mean and standard deviation of the mean scores vector. These two values make it possible to create a flexible thresholding method to detect an increase or decrease in data trends. The system takes into consideration $2^*\rho$ to set a visual reference for the normal behaviour. If some values are out of this reference, it means the data differs effectively from the expected profile.

The given transformation operations result in reducing the number of rows in the initial matrix A. To address this issue, we devised a solution involving padding the similarity score vector, by sampling l values from the uniform distribution of normal and unseen score vectors.

Once the standard data processing is completed to establish data normality, we feed the unseen subset into the autoencoder to derive its latent representation. If any of the unseen samples deviate from the normal pattern, the autoencoder encounters challenges in reconstructing them, leading to distinct representations compared to the normal ones. Subsequently, we follow the same steps to analyze and visualize this data, ultimately presenting it in the form of a scatter plot.

4 Conclusions and results

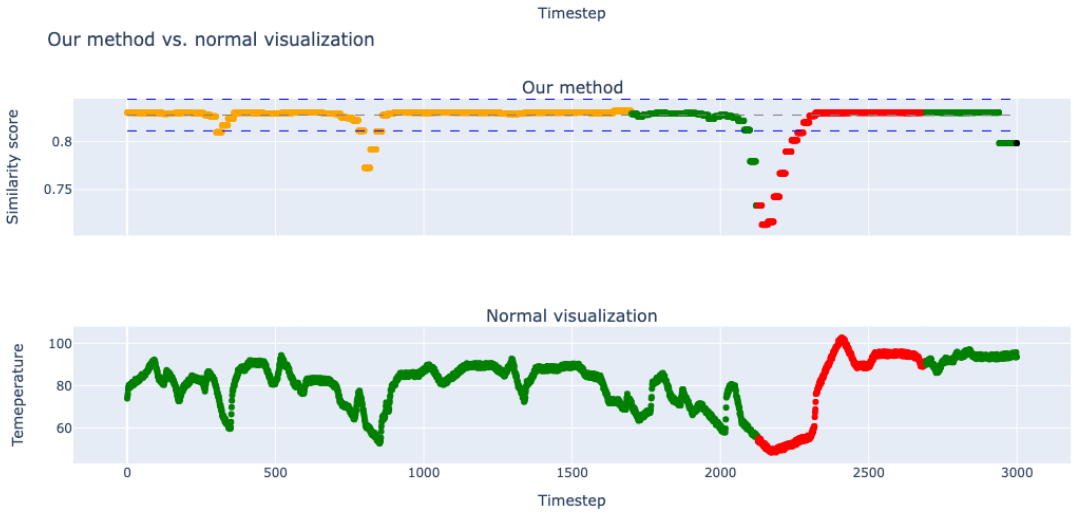

Figure 6: Result obtained from the *machine_temperature_system_failure* dataset from Numenta Anomaly Benchmark.

The visualization process depicts a comparison between the anomaly detection method's output (similarity scores) and the original data over time showcasing the performance of a shift detection method in detecting anomalies. The top subplot focuses on the method's performance in detecting anomalies computing the mean and standard deviation of a subset of data and then establishing a range within which data is considered normal. On the other hand the bottom subplot showcases the actual values.

The graphs are made with plotly [5], an interactive graphing library for creating visualizations in Python. The yellow points in the first subplot represent user-selected "normal data," , the green points indicate the newly introduced data while red points represent the ground truth anomalies. The plot has 3 reference lines corrisponding to mean and a threshold. The importance of this lines lies in establishing a reference range for "normal" behavior in the dataset. By calculating the mean and standard deviation, the code defines a range within which the majority of data points are expected to fall. Anomalies or unusual patterns can be identified by observing data points that fall outside this reference range. Furthermore, this method challenges the conventional notion of regular data by suggesting that certain data points within the normal range may appear peculiar or unusual.In contrast, the second graph shows the original dataset with no transformations, and the red points represent the ground truth anomalies.

A clear comparison between the two charts reveals that our approach significantly enhances anomaly detection compared to the traditional scatter plot. By effectively filtering out normal data, it directs the focus solely on peculiar behaviors, thus improving

anomaly visibility in the relevant regions.

In summary, our method offers a comprehensive and insightful dataset analysis, striking a balance between highlighting anomalies and reducing the visibility of normal data variations. This empowers users to make well-informed decisions and gain valuable insights into the underlying patterns and irregularities in their data.

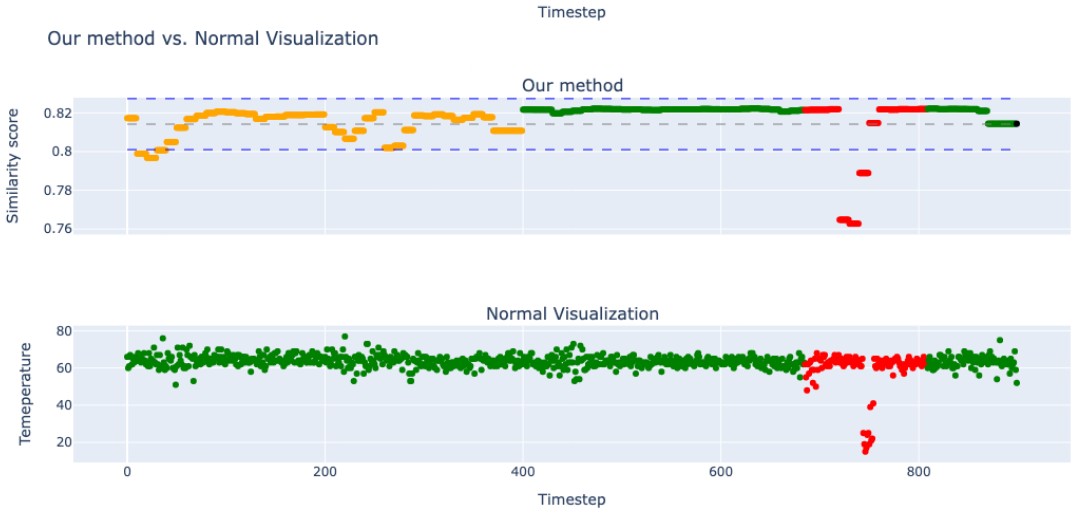

Figure 7: Result obtained from the speed_t4013 dataset from Numenta Anomaly Benchmark.

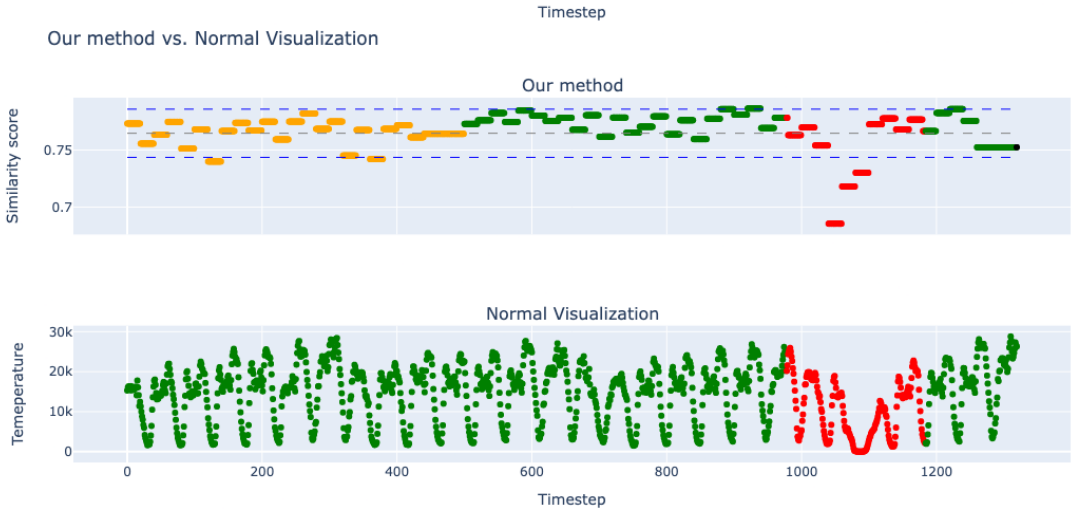

Figure 8: Result obtained from the *nyc_taxi* dataset from Numenta Anomaly Benchmark.

References

[1] Jiayi Liu. Anomaly and change point detection for time series with concept drift. *World Wide Web*, pages 1–24, 2023.

[2] Biagio La Rosa. State of the art of visual analytics for explainable deep learning. *Computer Graphics Forum*, 2023.

[3] Lilian Weng. From autoencoder to beta-vae. *World Wide Web*, 2018.

[4] Numenta. Numenta anomaly benchmark, 2023.

[5] Plotly. Plotly : Low-code data apps, 2023.