IDEAL: An Approach for Interactive Detection and Explanation of Anomalies using Autocorrelation-based LSTM-Autoencoder for Time-Series Data

Authors
Emails
Affiliations

ABSTRACT

Data quality significantly impacts the effectiveness of data analytics, which are playing an increasingly important role. Anomaly detection techniques are needed to identify incorrect data. While machine learning approaches exist for detecting anomalies, techniques are lacking for (1) the identification of the underlying constraints that were violated by the anomalous data and (2) the generation of explanations of these violations in a human-interpretable form. We propose the IDEAL framework that addresses this problem for time series data using an LSTM-Autoencoder. Our approach discovers complex constraints from the univariate or multivariate time series data in big datasets and reports subsequences that violate the constraints as outliers. We propose an automated autocorrelation-based windowing approach to adjust the network input size, thereby improving the correctness and performance of constraint discovery over manual approaches. The anomalies can be visualized as decision trees extracted from a random forest classifier that are comprehensible to the domain experts. Domain experts can then interactively provide their feedback to retrain the learning model and improve the accuracy of the process. We evaluate the effectiveness of our approach using Yahoo servers and NASA Shuttle datasets. Using mutation analysis, we show that our approach can detect different types of injected faults. We also demonstrate that the accuracy improves after incorporating domain expert feedback.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Anomaly detection**.

KEYWORDS

Anomaly detection, Auto-correlation, Data quality tests, Explainability, LSTM-Autoencoder, Time series.

ACM Reference Format:
Authors. 2020. IDEAL: An Approach for Interactive Detection and Explanation of Anomalies using Autocorrelation-based LSTM-Autoencoder for Time-Series Data. In *ARES '20: International Conference on Availability, Reliability and Security August 24–28, 2020, Dublin, Ireland*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/1122445.1122456

ARES '20, August 24–28, 2020, ARES, Ireland

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

A time series, also known as sequence data [5], is a set of time-ordered records [11]. Large volumes of real-world time series data are increasingly collected from various sources, such as Internet of Things (IoT) sensors, network servers, and patient medical flow reports [6, 11, 28]. A time-series data can get corrupted because of incorrect data collection, data transformation, and also because of intrusions and malicious insider attacks. Inaccurate data can produce incorrect analysis.

Existing approaches for finding anomalies by discovering the constraints in individual data records cannot be used for testing time series data as there may be constraints over multiple attributes and records in a time series. For example, in a univariate time series we must check that the *patient_weight* growth rate change is positive and in the range [*4 lb*, *22 lb*] for every infant. In a multivariate time series, we must also check for the relationship between the *patient_weight* and *blood_pressure*, and their growth rates over time for the adult patients. The records in a sequence have strong correlations and dependencies with each other, and their violations cannot be discovered by analyzing records in isolation [16].

Existing machine learning-based approaches that discovers constraints in time-series data have several limitations. Constraints are discovered within an input subsequence, the size of which is typically selected based on a fixed-sized window [18] or by using an exhaustive brute-force approach [25]. Since the window size can considerably affect the correctness of the discovered constraints, fixed-sized windows are not appropriate. Brute-force window-size tuning can be expensive. Moreover, the discovered associations are in the form of complex equations that are not human interpretable.

We propose an approach for **I**nteractive **D**etection and **E**xplanation of **A**nomalies using an **L**STM-autoencoder (IDEAL) to address these limitations. IDEAL uses a deep network called LSTM-Autoencoder [13] that discovers constraints involving long-term non-linear associations among multivariate time series data records and attributes. Subsequences and records that do not conform to the constraints are flagged as suspicious, which are then validated by domain experts. Expert feedback is incorporated and the analysis is repeated. The key contributions of this paper are as follows:

- To the best of our knowledge, IDEAL is the first approach to find anomalies in multivariate time series big data, explain them in terms of domain concepts, and illustrate the anomalous records and sequences using decision trees.
- A systematic autocorrelation-based windowing technique is proposed that automatically adjusts the input size based on how far the records are related to their past values. This results in a higher constraint-discovery effectiveness of the LSTM-Autoencoder model than when manually set fixed

window sizes are used, and a faster window size determination than when exhaustive brute-force window sizing approaches are used.
- Domain experts inspect and provide feedback on the suspicious sequences to reduce false alarms [12]. This feedback is captured as record label which is added as a new attribute and the learning model is retrained. Interactive learning is implemented by extending the LSTM-Autoencoder network and redefining the LSTM-Autoencoder loss function [30] based on the record labels.

We evaluated the constraint discovery, anomaly detection, and anomaly explanation effectiveness of IDEAL using the Yahoo server [23] and NASA Shuttle [20] datasets.

We compared the effectiveness and efficiency of our autocorrelation-based windowing approach with a brute-force window-size tuning approach. Mutation analysis showed that the true positive rate and false negative rates improve after incorporating ground truth knowledge about the injected faults and retraining the interactive-based LSTM-Autoencoder model. We showed that the visualization plots correctly explain the reason behind the reporting of the suspicious sequences.

The rest of the paper is organized as follows. Section 2 summarizes background material on time series data. Section 3 describes IDEAL and Section 4 presents its evaluation. Section 5 describes related work and discusses their limitations. Finally, Section 6 concludes the paper and outlines directions for future work.

2 BACKGROUND ON TIME SERIES

Table 1: Time Series Features [22]

Feature	Description
F_1: Mean	Mean value
F_2: Variance	Variance value
F_3: Lumpiness	Variance of the variances across multiple blocks
F_4: Lshift	Maximum difference in mean between consecutive blocks
F_5: Vchange	Maximum difference in variance between consecutive blocks
F_6: Linearity	Strength of linearity
F_7: Curvature	Strength of curvature
F_8: Spikiness	Strength of spikiness based on the size and location of the peaks and troughs
F_9: Season	Strength of seasonality based on a robust STL [26] decomposition
F_{10}: Peak	Strength of peaks
F_{11}: Trough	Strength of trough
F_{12}: BurstinessFF	Ratio between the variance and the mean (Fano Factor)
F_{13}: Minimum	Minimum value
F_{14}: Maximum	Maximum value
F_{15}: Rmeaniqmean	Ratio between interquartile mean and the arithmetic mean
F_{16}: Moment3	Third moment
F_{17}: Highlowmu	Ratio between the means of data that is below and upper the global mean
F_{18}: Trend	Strength of trend based on robust STL decomposition

A time series T is a sequence of d-dimensional records [11] described using the vector $T =< R_1, ..., R_n >$, where $R_i = (a_i^1, ..., a_i^d)$ is a record at time i, for $1 \leq i \leq n$ and a_i^j is the j^{th} attribute of the i^{th} record. Existing data analysis approaches [11] assume that the time gaps between any pair of consecutive records differ by less than or equal to an epsilon value, i.e., the differences between the time stamps of any two consecutive records are nearly the same.

A time series can be *univariate* (d=1) or *multivariate* (d>1) [6]. A univariate time series has one time-dependent attribute. For example, a univariate time series can consist of daily temperatures recorded sequentially over 24-hour increments. A multivariate time series is used to simultaneously capture the dynamic nature of multiple attributes. For example, a multivariate time series can consist of precipitation, wind speed, snow depth, and temperature.

The research literature [22, 24] uses various features that describe the relationships among the time-series records and attributes. Trend and seasonality [1] are the most commonly used features. Trend is defined as the general tendency of a time series to increment, decrement, or stabilize over time [1]. For example, there may be an upward trend for the number of patients with cancer diagnosis. Seasonality is defined as the existence of repeating cycles in a time series [1]. For example, the sales of swimwear is higher during summers. Table 1 shows a set of features defined by Talagala et al. [22] to describe a time series. We use these features (1) to identify the types of constraint violations reported by IDEAL in Section 3.4 and (2) to define mutation operators that violate the constraints over these features in Section 4.1.

A constraint is defined as a rule over the time-series features. For example, the mean (F_1) value of the daily electricity power delivered by a household must be in the [0.1 KWH, 0.5 KWH] range. We categorize the faults that can violate the constraints over time-series features as *anomalous records* and *anomalous sequences*.

Anomalous records. Given an input time series T, an anomalous record R_t is one whose observed value is significantly different from the expected value of T at t. An anomalous record may violate constraints over the features $F_1, F_2, F_3, F_4, F_5, F_6, F_7, F_8, F_{12}, F_{13}, F_{14}$, F_{15}, F_{16}, and F_{17}. For example, if there is a constraint that imposes a range of values (F_{13}, F_{14}) for the infant patients' weights during their first three months, a record in the first three months with a weight value outside this range must be reported as faulty.

Anomalous sequences. Given a set of subsequences $T = \{T_1, ..., T_m\}$ in a time series T, a faulty sequence $T_j \in T$ is one whose behavior is significantly different from the majority of subsequences in T. A faulty sequence may violate constraints over any of the features F_1 through F_{18} . For example, consider the constraint that imposes an upward trend (F_{18}) for the number of cars passing every second at an intersection from 6 to 7 am on weekdays. A decrease in this trend is anomalous.

3 PROPOSED APPROACH

We illustrate our approach using the Yahoo server traffic datasets in the Yahoo Webscore program [23] and the NASA Shuttle dataset in the UCI ML repository [20]. The Yahoo server traffic datasets contain real and synthetic univariate time series, each of which contains 1,420 time ordered records with one time-dependent attribute called *traffic_value*. These datasets contain time series with random seasonality, trend and noise. Each data-point represents one hour's worth of traffic data. The NASA Shuttle multivariate dataset contains 58,000 time-ordered records with seven time-dependent

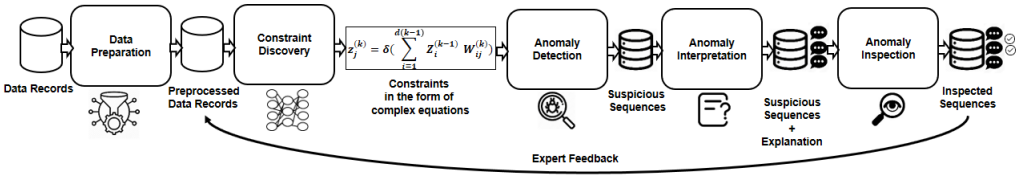


Figure 1: IDEAL Overview

numerical attributes, namely, *Rad_flow*, *Fpv_close*, *Fpv_open*, *High*, *Bypass*, *Bpv_close*, and *Bpv_open*.

Figure 1 shows an overview of our proposed approach. The input is in the form of data records and the output consists of a report showing suspicious subsequences accompanied with an explanation of the violated constraints. There are five components, namely, *data preparation*, *constraint discovery*, *anomaly detection*, *anomaly explanation*, and *anomaly inspection*.

3.1 Data Preparation

IDEAL uses an LSTM-Autoencoder network that takes a sequence of records as input (1) to extract features as complex dependencies among records and attributes of the sequence, and (2) to discover constraints as complex equations over those features [15]. The input dataset must be transformed into a form suitable for analysis before we can fit the LSTM-Autoencoder model to the dataset. One-hot encoding [29] is used for preprocessing the categorical attributes and normalization [21] is used for the numeric attributes.

The LSTM-Autoencoder input is a matrix with three dimensions, namely, *batch size*, *window size*, and *input*. The batch size dimension defines the number of subsequences that are utilized by LSTM-Autoencoder in one epoch (i.e., one iteration of training). The window size is the number of consecutive records in each subsequence. The input dimension is the number of attributes of the records in the subsequence. The number of units in the hidden layers of the LSTM-Autoencoder network depends upon these three dimensions. Figure 4 in Section 1 shows the input to an LSTM-Autoencoder, where the batch size is equal to one, window size is equal to w, and input dimension is equal to $d+1$. To transform the data obtained from the preparation step into the right shape for input to the LSTM-Autoencoder sequential model, *autocorrelation-based reshaping* is proposed.

An LSTM network discovers long-term dependencies between consecutive data records in an input subsequence whose size is an indicator of how far the network connects a data record to its past values. The correctness of the constraints discovered by the network is considerably affected by the correct selection of the window size [25]. For example, consider an LSTM network that is trained to discover dependencies among the hourly temperature values in a weather dataset. The current temperature value may be related to the previous values during the day ($window\ size = 24$), but is less likely to be related to the values during the previous week ($window\ size = 168$).

Existing reshaping techniques use a fixed window size [18]. While using a small window size can result in missing constraints involving dependencies among the records, picking a large window size can result in a large increase in the computational complexity of the network. Other approaches increase the size until reaching the minimum network error based on an exhaustive brute-force approach, which is impractical for real-world big datasets [25].

We propose a systematic reshaping approach based on the autocorrelation of the time series attributes to enable the LSTM-Autoencoder network discover dependencies between the records that are highly correlated. The input size is adjusted based on how far the records are related to their past values. By feeding the LSTM-Autoencoder network with highly correlated records, this reshaping approach prevents the network from incorrectly discovering associations among non-correlated records.

Autocorrelation is defined as the correlation of sequence data records with the records in the previous time steps, called lags [19]. Given dataset d with $R(1), R(2), ..., R(N)$ records at time $t_1, t_2, ..., t_N$, the Autocorrelation Function (ACF) at lag k for an attribute a in this dataset is calculated as follows.

$$ACF(a,k) = \frac{\sum_{i=1}^{N-k}(d[a](i) - \overline{d[a]})(d[a](i+k) - \overline{d[a]})}{\sum_{i=1}^{N}(d[a](i) - \overline{d[a]})^2} \quad (1)$$

where $d(i)$ is the original dataset, $d(i+k)$ is the same dataset as the original, but just shifted by k lags, and $\overline{d[a]}$ is the average of the values of a in the original dataset. The numerator is the covariance between the data and the k-unit lagged data. The denominator is sum of the squared deviations of the original dataset.

An $ACF(a,k)$ value that rises above or falls below a confidence interval is considered to be significantly autocorrelated. The shaded area in Figure 2 shows the confidence interval calculated by equation 2 for the *High* attribute in the NASA Shuttle dataset.

$$\pm Z_{1-\alpha/2}\sqrt{\frac{1}{N}(1+2\sum_{i=1}^{k} d[\bar{A}]^2)} \quad (2)$$

where k is the lag, N is the sample size, z is the cumulative distribution function of the standard normal distribution, and α is the significance level. Based on this equation, the confidence bands increase as the lag increases.

In Figure 2, the height of each spike shows the value of autocorrelation function for the corresponding lag. Autocorrelation with a lag of zero is always equal to 1 (the autocorrelation between each term and itself). A spike being close to zero is evidence against autocorrelation. In this example, all the spikes are statistically significant for all the 20 lags, indicating that the values of the *High* attribute are highly correlated to its 20 past values.

We propose to use the ACF function for all the attributes $a_i \in A$. For each attribute a_i, IDEAL selects the lag value l_i after which the ACF function crosses the confidence interval (i.e., boundary of the shaded area) for the first time. The window size is set to $maximum(l_i)$, where $1 \leq i \leq size(A)$. Figure 3 demonstrates the window size selection based on autocorrelation in a univariate

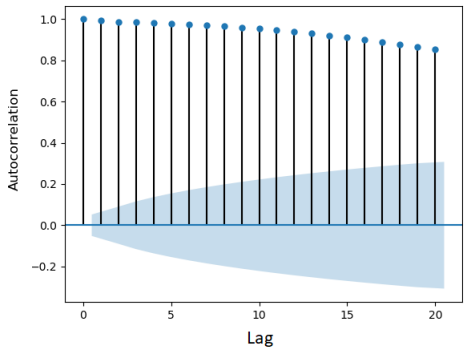


Figure 2: ACF for *High* Attribute in NASA Shuttle for 20 lags

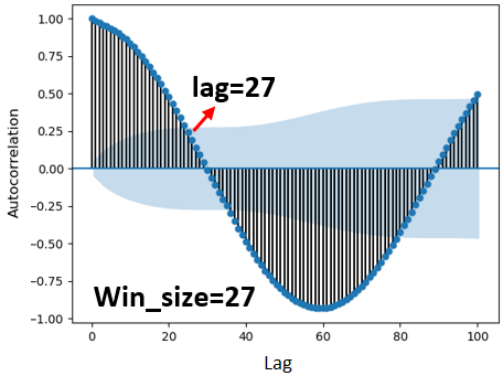


Figure 3: Use ACF to Select Window Size

Yahoo traffic dataset. In this example, l_{27} is the lag after which the ACF function crosses the confidence interval for the first time. Thus, the window size is set to 27.

3.2 Constraint Discovery

Although the deep architecture of an autoencoder can model complex constraints involving non-linear dependencies among multiple data records and attributes, it cannot model the temporal dependencies in the data. Thus, IDEAL uses an LSTM-Autoencoder to discover long-term complex dependencies between the data records and attributes in sequence data. Using LSTM networks in autoencoder layers aids in learning constraints over features involving long-term temporal dependencies among the time series data.

An LSTM [27] is a Recurrent Neural Network (RNN) that contains loops in its structure to allow information to persist and make the network learn sequential dependencies among data records. RNNs can only learn short-term dependencies among data records by using recurrent feedback connections [6]. LSTMs extend RNNs by using specialized *memory cells* in neuron structures to learn long-term dependencies. An LSTM has the ability to remove or add information to the memory cell state by using *gates*, which are defined as weighted functions that govern information flow in the memory cells. Gates are composed of a *sigmoid layer* and a *point-wise operation* to optionally let information through. The sigmoid layer outputs a number between zero (to let nothing through) and one (to let everything through). A *forget gate* decides what information to discard from the memory cell. An *input gate* decides which values must be used from the network input to update the memory state. An *output gate* decides what to output based on the input and the memory state.

An autoencoder is a deep network that discovers constraints in unlabeled data and is composed of an *encoder* and a *decoder*. The encoder compresses the data from the input layer into a short representation, which is a non-linear combination of the input elements. The decoder decompresses this representation into a new one that closely matches the original data. The network is trained to minimize the reconstruction error (RE), which is the average squared distance between the original data and its reconstruction [30].

An LSTM-Autoencoder is a sequence-to-sequence modeling technique [15] used to learn time series dependencies. Figure 4 shows the LSTM-Autoencoder architecture. The input and output are fixed-size time series matrices. $X_j = [x_j^0, ..., x_j^d, L_j]$ is the j^{th} record with $d+1$ attributes, T_i is the i^{th} time series that contains w records, and w is the window size. The network output has the same dimensionality as the network input. The network is composed of two hidden layers that are LSTMs with d' units. The first LSTM layer functions as an encoder that investigates the dependencies from the input sequence and produces a complex hidden context (i.e., d' encoded time series features, where the value of d' depends on the underlying encoding used by the autoencoder). The second LSTM layer functions as a decoder that produces the output sequence, based on the learned complex context and the previous output state. The TimeDistributed layer is used to process the output from the LSTM hidden layer. This layer is a dense (fully-connected) wrapper layer that makes the network return a sequence with shape $(d * w)$. The reconstruction error for this network is defined as follows [30]:

$$RE = \frac{1}{N}\sum_{i=1}^{m}(T_i' - T_i)^2 \tag{3}$$

where T_i and T_i' are the i^{th} network input and output.

The LSTM-Autoencoder is an unsupervised technique that can potentially learn incorrect constraints from invalid data and generate false alarms. We use an interactive learning approach that takes the expert's feedback to retrain the LSTM-Autoencoder model and improve its accuracy. We extend the LSTM-Autoencoder architecture by adding a label as a new input to the network structure. The shaded area in Figure 4 shows the extension. In Section 3.5, we describe how this label (1: faulty, 0.5: suspicious, 0: unknown, and -1: valid) is updated using domain expert feedback in every interaction. We redefine the reconstruction error of LSTM-Autoencoder based on the labels to minimize false alarms. The network is trained to minimize both the difference between the time series and its reconstruction, and the difference between the record labels in a time series and the labels predicted by the network. Equation 4 shows the extended reconstruction error.

$$RE = \frac{1}{N}\sum_{i=1}^{m}((T_i' - T_i)^2 + (mean(L_i') - mean(L_i))^2) \tag{4}$$

where $mean(L_i)$is the arithmetic mean of the record labels in time series T_i and $mean(L_i')$ is the mean of the reconstructed record labels in the reconstructed time series T_i'.

3.3 Anomaly Detection

We define suspiciousness scores (s-score values) to detect suspicious subsequences, records, and attributes that violate the constraints

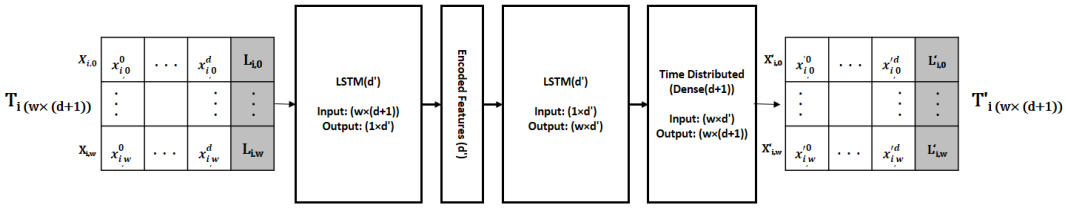


Figure 4: Extending LSTM-Autoencoder by Adding a Label Input

discovered by the LSTM-Autoencoder network. The *s*-scores are defined based on the reconstruction error and the record labels. Using labels in the definition of *s*-scores ensures that no valid subsequences or records are reported as suspicious in the retraining phase, thereby minimizing false alarms.

s-score_per_attribute. This value lies between 0 and 1, and is assigned to each attribute of a subsequence. It indicates the contribution of each attribute to the invalidity of the subsequence. Equation 5 shows how to calculate this value.

$$s\_score\_a_i^j = Normalized(\frac{1}{w}\sum_{k=0}^{w}(x^j_{(i+k)} - x'^j_{(i+k)})^2) \quad (5)$$

where $s\_score\_a_i^j$ is the score assigned to the j^{th} attribute in the i^{th} subsequence and w is the window size.

s-score_per_record. This value lies between 0 and 1, and is assigned to each record of a subsequence to indicate the contribution of each record to the invalidity of the subsequence. Equation 6 shows how to calculate this value.

$$s\_score\_r_{i,q} = Normalized(\frac{1}{d}\sum_{k=0}^{d}(x'^k_{i,q} - x^k_{i,q})^2 + (L'_{i,q} - L_{i,q})^2) \quad (6)$$

where d is the number of attributes and $s\_score\_r_{i,q}$ is the score assigned to the q^{th} record in the i^{th} subsequence.

s-score_per_subsequence. This value indicates the level of invalidity of the subsequence. Equation 7 shows how to calculate it.

$$s\_score_i = (\frac{1}{d}\sum_{j=0}^{d} s\_score_i^j) + max(L_i) \quad (7)$$

where s_score_i is the score assigned to the i^{th} subsequence and $s\_score_i^j$ is the *s*-score_per_attribute for that subsequence. In Equation 7, $(\frac{1}{d}\sum_{j=0}^{d} s\_score_i^j)$ is a value between zero and one and $max(L_i)$ is a value in the $\{-1, 0, 0.5, 1\}$ set and is equal to the maximum value of the record labels in the subsequence. In the retraining phase, a subsequence with all valid records (i.e., all labeled -1) gets an $s\_score \leq 0$ and is not reported as suspicious. A subsequence with at least one invalid record (i.e., at least one record labeled by 1) gets an $s\_score \geq 1$ and is marked as suspicious.

3.4 Anomaly Explanation

The associations learned by the proposed LSTM-Autoencoder are in the form of complex equations that are not human interpretable. To address this problem, we display *s*-score_per_record for the records in the suspicious subsequence to highlight the contribution of each record to the invalidity of the subsequence. We also generate two types of visualization plots to describe the constraints violated by the detected anomalies: (1) *s*-score_per_attribute and (2) decision trees generated using a random forest classifier [10]. These plots help domain experts inspect the suspicious subsequences.

s-score_per_record. Table 2 shows results from a suspicious subsequence from the NASA Shuttle Dataset. The last column shows the *s*-score_per_record values for the suspicious subsequence. In this example, the first record with an out-of-range value for the *Fpv_open* attribute has the largest value of *s*-score and is the major cause of the invalidity of this subsequence.

s-scores_per_attribute plot. This plot displays the contribution of each attribute to the invalidity of the subsequence. The horizontal axis indicates the attribute names and the vertical axis indicates their level of invalidity. Figure 5 shows an example of the *s*-score_per_attribute plot for the suspicious subsequence from the NASA Shuttle dataset. Using a threshold value equal to the average value of the *s*-score_per_attribute in the subsequence, the *Fpv_open* and *Bypass* attributes are determined to be the major causes of invalidity for this subsequence. This plot is only applicable to the multivariate time series in which there are more than one attributes to be compared based on their values of *s*-score_per_attribute.

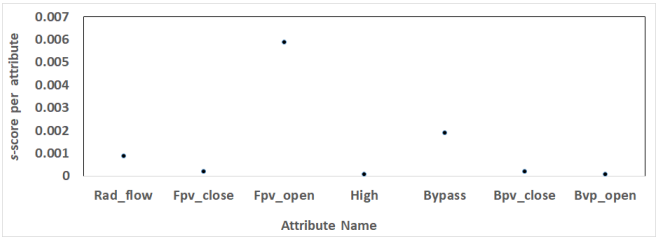


Figure 5: *s*-score per Attribute Plot for Suspicious Subsequence Detected from NASA Shuttle Dataset

Decision tree. This display describes the constraints violated by each of the suspicious subsequences. Decision trees are one of the easiest models to understand [4]. The non-leaf nodes in the decision tree correspond to the subsequence features, the edges correspond to the possible values of the features, and every leaf node contains the label of the path described by the feature values from the root to that leaf node (label 0: valid and label 1: invalid). A random forest [10] generates a number of trees on various subsets of a dataset, whereas the basic decision tree classifier [4] generates only one tree. We use the random forest classifier because it uses the average prediction obtained from the trees to improve the predictive accuracy and to prevent over-fitting [2] to the training dataset.

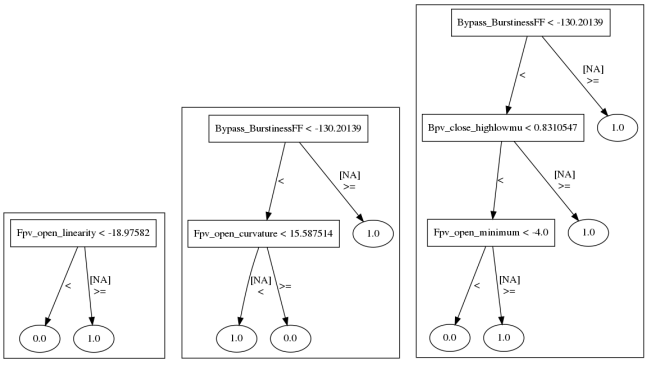
For each attribute of the subsequence, the 18 time series features listed in Table 1 are extracted using Tsfeatures [22] CRAN library. Next, decision trees are generated through a random forest classifier

Table 2: Suspicious Subequence Detected from the NASA Shuttle Dataset

Record Id	Rad_flow	Fpv_close	Fpv_open	High	Bypass	Bpv_close	Bpv_open	*s*-score per record
7102	2.0	88.0	48.0	46.0	6.0	42.0	41.0	0.460979
7103	0.0	83.0	-6.0	46.0	5.0	37.0	36.0	0.014823
7104	2.0	79.0	0.0	46.0	3.0	34.0	33.0	0.010940
7105	3.0	76.0	0.0	46.0	7.0	30.0	30.0	0.044005
7106	2.0	86.0	0.0	46.0	20.0	40.0	39.0	0.109730
7107	2.0	80.0	0.0	44.0	-29.0	34.0	36.0	0.130686
7108	0.0	77.0	0.0	44.0	-20.0	31.0	33.0	0.068818
7109	1.0	83.0	0.0	46.0	11.0	37.0	37.0	0.043986
7110	0.0	84.0	0.0	44.0	-27.0	38.0	40.0	0.099747
7111	4.0	87.0	0.0	46.0	3.0	41.0	41.0	0.016285

using the suspicious subsequences labeled as *invalid* and all the non-suspicious sequences labeled as *valid*. Among all the generated trees, the three with the lowest classification errors are displayed via the web interface of our tool. Domain experts can decide whether or not a suspicious subsequence is actually anomalous by analyzing the constraints represented in the trees and by inspecting the values of the time series features. We limited the number of decision trees displayed for each subsequence to three to make it feasible for domain experts to inspect the constraints represented in the trees.

This plot is applicable to both univariate and multivariate time series. Using a univariant time series as the input dataset, the decision trees identify the constraints violated over the features that are extracted from the values of the only attribute in the dataset. Using a multivariate time series, the decision trees identify the constraints violated over the features that are extracted from the values of all attributes in the dataset.



(a) DT1 **(b) DT2** **(c) DT3**

Figure 6: Decision Trees for Suspicious Sequence in NASA Shuttle Dataset

Figures 6 shows examples of decision trees generated for a suspicious subsequence in the NASA Shuttle dataset. The constraints over time series features that are violated by the suspicious subsequence are in Table 3. The first tree in Figure 6 represents a constraint over the linearity feature of the *Fvp_open* attribute. Based on this tree, if the linearity of *Fvp_open* is less than −18.97582, then the subsequence is valid (the leaf node containing the label of the subsequence contains a value equal to 0.0). If the linearity of *Fvp_open* is equal to NULL or is greater than or equal to −18.97582, then the subsequence is invalid (the leaf node contains a value equal to 1.0)

Table 3: Constraints over Features Violated by Suspicious Sequence in NASA Shuttle Dataset

First Decision Tree
IF linearity(*Fvp_open*) ≥ -18.98 OR linearity(*Fvp_open*) = NULL, THEN sequence=invalid
IF linearity(*Fvp_open*) < -18.98, THEN sequence=valid
Second Decision Tree
IF BurstinessFF(*Bypass*) ≥ -130.20 OR BurstinessFF(*Bypass*)= NULL, THEN sequence=invalid
IF BurstinessFF(*Bypass*) <-130.200 AND (Curvature(*Fvp_open*) < 15.59 OR Curvature(*Fvp_open*) = NULL), THEN sequence= invalid
IF BurstinessFF(*Bypass*) < -130.20 AND Curvature(*Fvp_open*) ≥ 15.59, THEN sequence=valid
Third Decision Tree
IF BurstinessFF(*Bypass*)≥ -130.20 OR BurstinessFF(*Bypass*) = NULL, THEN sequence=invalid
IF BurstinessFF(*Bypass*) < -130.209 AND (Highliwmu(*Bpv_close*)≥ 0.83 OR Highliwmu(*Bpv_close*) =NULL), THEN sequence= invalid
IF BurstinessFF(*Bypass*) < -130.20 AND Highliwmu(*Bpv_close*) < 0.83 AND (Minimum(*Fpv_open*) ≥ -4.0 OR Minimum(*Fpv_open*) = NULL), THEN sequence=invalid
IF BurstinessFF(*Bypass*) < -130.20 AND Highliwmu(*Bpv_close*) < 0.83 AND Minimum(*Fpv_open*) < -4.0, THEN sequence=valid

3.5 Anomaly Inspection

This component takes a domain expert's feedback about the suspicious subsequences through a web-based user interface. Check boxes in the interface allow the expert to appropriately mark the subsequences that are actually anomalous. As described previously, the feedback is used to label the training data records with four possible values. The label is initially 0 for every record. The values of the label are updated based on the feedback. Records in a subsequence marked as suspicious by the anomaly detection component are labeled 0.5, out of which those marked as actually anomalous by the domain expert are labeled 1, and those not marked are labeled -1. Labels of records that are not reported suspicious by IDEAL remain 0. The updated dataset is used to retrain the LSTM-Autoencoder network. As described in Sections 3.2 and 3.3, this label is used by the interactive constraint discovery and anomaly detection components to improve the accuracy of the LSTM-Autoencoder approach.

4 EVALUATION

We evaluated the effectiveness of constraint discovery, anomaly detection, and anomaly explanation as well as the improvement in the accuracy of our interactive approach using Yahoo synthetic

servers dataset and one NASA Shuttle dataset. Due to the absence of ground truth data, we used a mutation analysis technique to inject a set of known faults into the data. In keeping with the spirit of traditional mutation analysis used in software testing [17], we calculate the mutation score, which is the ratio of the number of injected faults reported as anomalies by our approach and the total number of injected faults.

4.1 Mutation Analysis

We defined mutation operators with the goal of violating constraints over the features described in Table 1. These operators result in *mutants*, which are faulty records or sequences that mimic typical anomalies in the sequence data. A mutant is defined to be *killed* when the suspicious subsequences detected by IDEAL contain the mutant records or sequences. Each mutation operator modifies the records in order to violate the constraints relevant to at least one of the time series features. As a result of using the operators, all the features get invalid values (i.e., values that violate constraints over the features). Table 4 shows the mutation operators used to inject faults in the data, the fault types, and the features that must be reported in the constraint violations caused by the operators for a mutant to be killed. We identified these features for each operator based on our observations when the operators were applied to the Yahoo and NASA Shuttle datasets.

Table 4: Injected Faults and Violated Features

Mutation Operator	Fault Type	Violated Features
M_1: Add noise	Anomalous record	F_1, F_2, F_3, F_4, F_5, F_6, F_7, F_8, F_{12}, F_{13}, F_{14}, F_{15}, F_{16}, F_{17}
M_2: Horizontal shift	Anomalous sequence	F_1, F_2, F_3, F_4, F_5, F_6, F_8, F_9, F_{12}, F_{13}, F_{14}, F_{16}, F_{17}, F_{18}
M_3: Vertical shift	Anomalous sequence	F_1, F_6, F_9, F_{13}, F_{14}
M_4: Re-scale	Anomalous sequence	F_1, F_6, F_7, F_9, F_{13}, F_{14}, F_{16}
M_5: Add dense noise	Anomalous sequence	$F_1, F_2, F_3, F_4, F_5, F_6, F_7, F_8, F_9, F_{10}$, $F_{11}, F_{12}, F_{13}, F_{14}, F_{15}, F_{16}, F_{17}, F_{18}$

Our mutation engine takes each operator M_i, dataset D and attribute a as input to mutate the attribute value based on that operator. For the multivariate datasets, we randomly selected k attributes from the uniformly distributed attribute indexes. We used the same operator to mutate all of the k selected attributes. We describe these operators as follows:

M_1 *–Add noise.* Noise can get added to real-world datasets because of sudden malfunctions, disconnections in the sensors or servers, or attackers inserting random values in sequence data. This anomaly can violate constraints relevant to various time series features as shown in the last column of Table 4. For example, the mean value of delivered power by households during the day time is between 0 and 2 KWH. A sudden change of this value to 4 KWH indicates a violation of this constraint.

This mutation operator adds random noise to the corresponding attribute of randomly selected records from the entire dataset.

(1) Select $r \subset D$ as $r = \{R_i | i = random(1, size(D))\}$, where $|r| = random(1, size(D))$
(2) For each $R_i \in r$, change $R_i[a]$ to $\alpha \times a_i$, where $a_i = random(min(D[a]), max(D[a]))$ and $0 \leq \alpha \leq 10$

We chose the maximum value of α to be 10 based on our observations from the NASA Shuttle and Yahoo server datasets where the attributes of anomalous records contain values that are up to 10 times that of the valid attribute values.

The operators $M2$ through $M5$ modify a subset of consecutive records in the dataset. As a result, these operators result in faulty subsequences in the data. For these operators, we first randomly select a subset of consecutive records from the dataset based on the following notation. The size of this subset is between 5 to 10 percentage of the records in the entire dataset.

Select $s \subset D$ as $s = \{R_i | m \leq i \leq m + r\}$, where
$m = random(1, size(D) - 0.1 \times size(D))$ and
$r = random(0.05 \times size(D), 0.1 \times size(D))$

M_2 *–Horizontal shift.* A horizontal shift may occur in real-world datasets due to a temporary change in the regular process of data collection. For example, there may be a constraint over the trend of the power usage in a school from 8 to 11 AM of weekdays. A shift in the school starting hour or attacker manipulation can result in violation on this constraint.

This operator shifts attribute values of the records in the selected subset along the time axis. The operator fills the empty cells with a constant value equal to the first shifted value.

(1) Shift $\{R_i[a] | m \leq i \leq m + r\}$ along the time axis
(2) Fill empty attributes with $A = R_m[a]$

M_3 *–Vertical shift.* This operator adds a random value between the min and the max values of the attribute to all attribute values in the subset records. A vertical shift can occur in real-world datasets as a result of a temporary malfunction of the sensors that capture the sequence data. For example, the level of the value captured by a temperature sensor may temporarily increase/decrease as a result of a manipulation or malfunction of the sensor. This modification may violate constraints over maximum, minimum, and mean features of the modified subset.

M_4 *–Re-scale.* This operator multiplies all the attribute values in the subset of records with a random number, which is between the min and max values of that attribute. Rescaling can occur in real-world datasets as a result of a temporary modification of the sensors or servers that capture the sequence data. For example, if someone temporarily changes the unit of snow depth detector sensor from inches to centimeter, the values stored from the sensor will be 2.54 times greater than the expected values. This modification may violate constraints over maximum, minimum, and mean features of the modified subset.

M_5 *–Add dense noise.* This operator changes all attribute values in the subset of records to randomly selected values. Dense noise can get added to real-world datasets as a result of attacks on the sensors or servers that capture the data. The modified subset of data can violate constraints over multiple features of that subset. For example, if there is a constraint on the strength of the seasonality in the data, this constraint may be violated in that subset.

4.2 Evaluation Goals

We evaluated three aspects of IDEAL: (1) constraint discovery and anomaly detection effectiveness, (2) anomaly explanation effectiveness, and (3) performance. The evaluation was performed using mutated datasets described in Table 5. Synthetic_1 and Synthetic_4 are univariate datasets that are mutated using the operators M_1 through M_5. The number of mutants generated from each dataset is shown in the $|M|$ column. The number of attributes randomly selected from the NASA Shuttle dataset to perform M_1 to M_5 operators is indicated in the $|A|$ column.

Table 5: F1 Score Results for one execution of IDEAL

Dataset ID	Operator	$\|M\|$	$\|A\|$	$F1_t$	$F1_a$	$F1_f$
Synthetic_1	M_1	2	1	1.0	NA	0.6
Synthetic_1	M_2	99	1	0.8	NA	0.45
Synthetic_1	M_3	179	1	0.6	NA	0.49
Synthetic_1	M_4	263	1	0.98	NA	0.62
Synthetic_1	M_5	88	1	0.78	NA	0.56
Synthetic_4	M_1	2	1	1.0	NA	0.6
Synthetic_4	M_2	284	1	0.2	NA	0.45
Synthetic_4	M_3	680	1	0.4	NA	0.49
Synthetic_4	M_4	193	1	1.0	NA	0.69
Synthetic_4	M_5	723	1	1.0	NA	0.57
Shuttle	M_1	21	4	0.23	0.86	0.45
Shuttle	M_2	1593	1	0.21	0.67	0.60
Shuttle	M_3	2123	6	0.88	0.58	0.28
Shuttle	M_4	1472	6	0.57	1.00	0.58
Shuttle	M_5	1561	5	0.6	0.57	0.8

4.2.1 Goal 1. Constraint discovery and anomaly detection effectiveness. We demonstrate these in two parts.

RQ1.a: How effective is our autocorrelation-based windowing approach compared to a brute-force windowing approach?

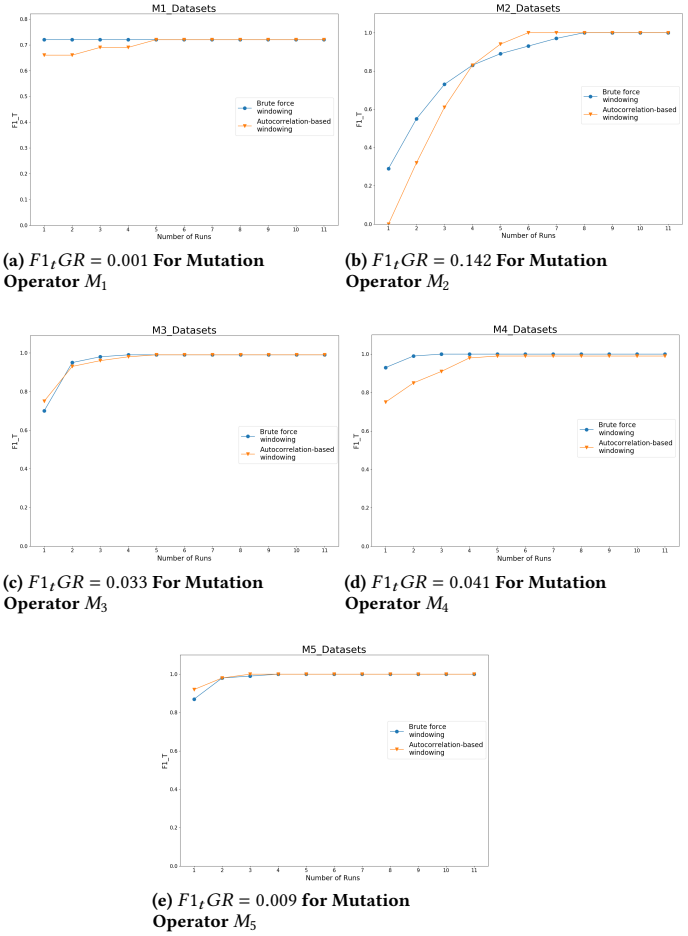
Let P_t be the number of actual faulty subsequences (i.e., has at least one actual faulty record), TP_t be the number of actual faulty subsequences detected as suspicious by the tool, N_t be the number of actual valid subsequences (i.e., does not include any actual faulty record), FP_t be the number of valid subsequences incorrectly detected as suspicious by the tool. We compared the $F1_t$ score of our approach with that of a brute force-based windowing approach to demonstrate the effectiveness of our windowing approach.

$$F1_t = 2 \times \frac{(precision_t \times recall_t)}{(precision_t + recall_t)} \tag{8}$$

where $precision_t = \frac{TP_t}{(TPR_t + FPR_t)}$, $recall_t = \frac{TPR_t}{(TPR_t + FNR_t)}$, $FPR_t = \frac{FP_t}{N_t}$, $TPR_t = \frac{TP_t}{P_t}$, $FNR_t = 1 - TPR_t$, and $TNR_t = 1 - FPR_t$.

Our IDEAL tool is configurable in terms of the input window size. One can choose a fixed size or an autocorrelation-based size. We wrote a script that takes the mutated dataset as input and executes the tool against a dataset multiple times (brute-force approach) using a range of window sizes to pick the best window size that results in the highest $F1_t$ score for the tool. We also ran the tool using our autocorrelation-based windowing. A comparison of the results of these two configurations is shown in Figure 7. It shows the $F1_t$ scores for autocorrelation-based windowing (orange line) and brute-force windowing (blue line) for 10 runs. In each run, we used the knowledge about the injected faults to label the data and retrain the learning model. The average of $F1_t$ scores are for the datasets in Table 5 that are mutated using different operators. For example, each data point in the first plot (Figure 7 (a)) shows the mean value of $F1_t$ scores for the Synthetic_1, Synthetic_4, and Shuttle datasets that are mutated using the $M1$ operator.

In 71% of the cases for all the datasets, the accuracy of our automatic selection of window size is close to that of the brute force approach (i.e., the difference between $F1_t$ of the two approaches was less than 0.04). On average, the mutation score calculated using the TPR_t value for all datasets is 0.60% for the first and 0.94% for the last execution using autocorrelation-based windowing. This score is 0.64% and 0.94% for the first and last run of the tool using the brute-force approach. In Section 4.2.3, we demonstrate that our approach is more efficient than the brute force approach.



(a) $F1_tGR = 0.001$ **For Mutation Operator** M_1

(b) $F1_tGR = 0.142$ **For Mutation Operator** M_2

(c) $F1_tGR = 0.033$ **For Mutation Operator** M_3

(d) $F1_tGR = 0.041$ **For Mutation Operator** M_4

(e) $F1_tGR = 0.009$ **for Mutation Operator** M_5

Figure 7: Average $F1_t$ for Mutated Datasets using Two Types of Windowing

RQ1.b: Does the accuracy of the interactive constraint discovery approach improve after retraining the machine learning model with the help of feedback from domain expert?

Given *NR*, the total number of times an expert revalidates the data until the desired $F1_t$ is reached, we measured $F1_t$ growth rate ($F1_tGR$) to answer this question, which is defined as the percentage change of a $F1_t$ variable within the interactive learning period.

$$F1_tGR = (\frac{F1_{tNR}}{F1_{t1}})^{\frac{1}{NR}} - 1 \quad (9)$$

where $F1_{t1}$ is the $F1_t$ at the first run and $F1_{tNR}$ is the $F1_t$ at the last run. The plots in Figure 7 (a)–(e) show positive $F1_tGR$ scores between 0.001 and 0.142 for all the datasets. This indicates that the accuracy of the approach improves after using ground truth knowledge to retrain the learning model.

4.2.2 Goal 2. Anomaly explanation effectiveness. We demonstrate explanation effectiveness in two parts.

RQ2.a: Are the attributes reported as major causes of invalidity of suspicious subsequences actually invalid?

Let P_a be the number of actual invalid features that must be reported for constraint violations, N_a be the number of actual valid features that must not be reported as constraint violations, TP_a be the number of actual invalid features the decision trees report in their constraint violations, and FP_a be the number of actual valid features that they incorrectly report as constraint violations. We measure $F1_a$ to answer this question.

$$F1_a = 2 \times \frac{(precision_a \times recall_a)}{(precision_a + recall_a)} \quad (10)$$

where $precision_a = \frac{TPR_a}{(TPR_a+FPR_a)}$, $recall_a = \frac{TPR_a}{(TPR_a+FNR_a)}$, $FPR_a = \frac{FP_a}{N_a}$, $TPR_a = \frac{TP_a}{P_a}$, $FNR_a = 1 - TPR_a$, and $TNR_a = 1 - FPR_a$.

RQ2.b: Are the time series features reported in the constraint violations that are represented by the decision trees actually invalid?

Let P_f be the number of actually violated features (i.e., features that must be violated using a mutation operator), N_f be the non-violated features, TP_f be the number of actually violated features the decision trees report as violated, and FP_f be the number of non-violated features that the decision trees incorrectly report as violated. We calculated $F1_f$ to answer this question.

$$F1_f = 2 \times \frac{(precision_f \times recall_f)}{(precision_f + recall_f)} \quad (11)$$

where $precision_f = \frac{TPR_f}{(TPR_f+FPR_f)}$, $recall_f = \frac{TPR_f}{(TPR_f+FNR_f)}$, $FPR_f = \frac{FP_f}{N_f}$, $TPR_f = \frac{TP_f}{P_f}$, $FNR_f = 1 - TPR_f$, and $TNR_f = 1 - FPR_f$.

Table 5 shows different observed $F1$ scores for one execution of IDEAL against the datasets. The $F1_a$ score is not applicable (NA) to univariate datasets (Synthetic_1 and Synthetic_4). Based on this table, the $F1_a$ scores (the accuracy of the s-score per attribute plots) were between 0.57 to 1.0 for the mutated Shuttle datasets. The $F1_f$ scores (the accuracy of the decision tree plots) was between 0.28 to 0.8 for all the mutated datasets. The $F1_a$ and $F1_f$ scores were calculated for the subsequences that were actually faulty (i.e., subsequences that included at least one actual faulty record).

4.2.3 Goal 3: Performance of constraint discover and anomaly detection. We answered the following question.

RQ3: How long does it take to execute IDEAL against a dataset?

We measure the Total Time (TT) (in seconds) it takes to perform the automated steps of IDEAL: perform data preparation, constraint discovery, anomaly detection, and anomaly explanation. The time spent by domain experts is not included because it may vary.

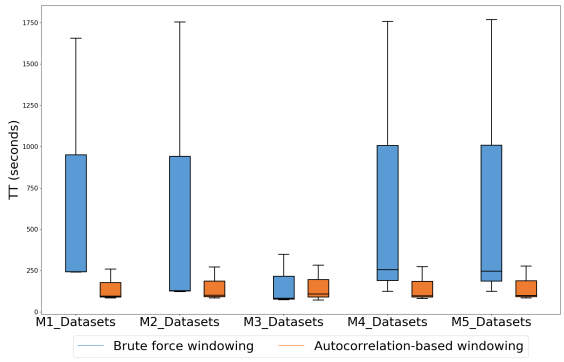


Figure 8: TT Box Plots for Datasets Mutated by M_1–M_5

Figure 8 shows the box plots of TT using brute-force and autocorrelation based windowing for all the datasets based on the mutation operator for one execution of IDEAL. In four out of five cases, the medians of the boxes for the autocorrelation-based approach are lower than the ones for the brute-force windowing approach. Moreover, the interquartile ranges of the boxes for the brute-force approach are considerably wider than the ones for the autocorrelation-based approach, which shows that TT taken by the brute-force approach is affected more by factors such as the dataset size and its number and type of the attributes. On average, it takes 152.90 seconds to run our approach based on autocorrelation windowing and 593.53 seconds to run the approach based on brute-force windowing. Our autocorrelation-based approach is 3.88 times faster than the brute-force windowing approach.

5 RELATED WORK

Systematic testing techniques have been proposed by researchers and practitioners to detect outliers in non time series data [7, 8].

We categorize the approaches that detect anomalous records from time series data in terms of *decomposition* and *modeling* techniques. The former techniques decompose a time series into level, trend, seasonality, and noise components and monitor the noise component to capture the anomalous records [9, 14]. These techniques apply only to univariate time series. The latter techniques model a time series as a linear/non-linear function that associates each current value to its past values, predict the value of a record at a specific time, and report as anomalies those records whose prediction error falls outside a threshold. Stochastic modeling techniques, such as Autoregressive Integrated Moving Average (ARIMA) use statistical measures to calculate the correlation between the data records. These techniques assume that the time series is linear and follows a known statistical distribution, which makes it inapplicable to many practical problems [1]. Machine learning modeling techniques support non-linear modeling, with no assumption about the distribution of the data [1]. Examples are Multi Layer Perceptions (MLPs) and Long Short Term Memory (LSTM). An MLP [3] can only model the short-term dependencies among the records. An LSTM is a Recurrent Neural Network (RNN) that contains loops as well as memory cells in its structure to model non-linear long-term sequential dependencies among the records in univariate/multivariate time series. The trained LSTM network is a complex equation over the attributes of the data records that is not human interpretable.

Existing techniques for anomalous sequence detection split the data into multiple subsequences, typically based on a fix-sized window [18] or an exhaustive brute-force approach [25]. Clustering-based anomalous sequence detection techniques extract subsequence features, such as trend and seasonality, and group the subsequences based on the similarities between their features. An anomalous subsequence is detected as the one that is distantly positioned within a cluster or is positioned in the smallest cluster. These approaches only detect anomalous sequences without determining the records/attributes that are the major causes of invalidity in each subsequence. Autoencoder-based techniques (1) take subsequences as input, (2) use an autoencoder network to reconstruct the subsequences, (3) assign invalidity scores based the reconstruction errors to the subsequences, and (4) detect as anomalous those subsequences whose scores are greater than a threshold. These techniques can learn complex non-linear associations among the attributes in the time series but are not able to model the temporal dependencies among the records in the input subsequence. An LSTM-Autoencoder is an extension of an autoencoder for time series data, which can capture the long-term temporal associations among data records in the input time series in the form of complex equations that are not human interpretable.

6 CONCLUSIONS AND FUTURE WORK

We developed an LSTM-Autoencoder-based approach called IDEAL to find anomalies in multivariate time series data. We proposed autocorrelation-based windowing to automatically identify the input size of the LSTM-Autoencoder network. Decision trees are generated to explain the detected suspicious subsequences and records. Domain expert feedback is used to improve the accuracy of the approach. We demonstrated that IDEAL can detect different types of anomalies which we created using mutation analysis injected in the Yahoo and NASA Shuttle datasets. We demonstrated that the autocorrelation-based splitting of the input data is almost as effective but 3.88 times more efficient than the existing brute force window-size tuning approaches. On average, it took IDEAL 91.52 seconds to run against the mutated univariate datasets with 1,420 records, and 273.83 seconds against the mutated multivariate datasets with 58,000 records.

IDEAL was more effective in finding anomalies (i.e., killing the mutants) in the univariate time series data. The average $F1_t$ score for the mutated univariate datasets was 0.77 and the one for the mutated multivariate datasets was 0.35 for one execution of IDEAL.

We demonstrated that the true positive and false negative rates improved (the growth rate of the $F1_t$ score was between 0.001 to 0.142) after incorporating ground truth knowledge about the injected faults. The average $F1_t$ score after 10 executions of IDEAL was 0.99 for the mutated univariate datasets and 0.84 for the mutated multivariate datasets. Finally, we demonstrated that 28 to 100 percent of the visualization plots could correctly explain the constraints violated by the suspicious subsequences. In the future, we plan to evaluate the approach using other types of real-world time series data. We plan to extend IDEAL to find anomalous records and sequences in streaming data.

REFERENCES

[1] Ratnadip Adhikari and R. K. Agrawal. 2013. *An Introductory Study on Time Series Modeling and Forecasting*. LAP LAMBERT Academic Publishing.

[2] Christopher M. Bishop. 2011. *Pattern Recognition and Machine Learning*. Springer.

[3] Christopher M. Bishop and Professor of Neural Computing Christopher M. Bishop. 1995. *Neural Networks for Pattern Recognition*. Clarendon Press.

[4] Paul B. de Laat. 2018. Algorithmic Decision-Making Based on Machine Learning from Big Data: Can Transparency Restore Accountability. *Philosophy & Technology* 31, 4 (2018), 525–541.

[5] Guozhu Dong and Jian Pei. 2007. *Sequence Data Mining*. Vol. 33. Springer Science & Business Media.

[6] T. Guo, Z. Xu, X. Yao, H. Chen, K. Aberer, and K. Funaya. 2016. Robust Online Time Series Prediction with Recurrent Neural Networks. In *IEEE International Conference on Data Science and Advanced Analytics*. 816–825.

[7] Hajar Homayouni, Sudipto Ghosh, and Indrakshi Ray. 2019. ADQuaTe: An Automated Data Quality Test Approach for Constraint Discovery and Fault Detection. In *IEEE 20th International Conference on Information Reuse and Integration for Data Science*. Los Angeles, CA, 61–68.

[8] Hajar Homayouni, Sudipto Ghosh, Indrakshi Ray, and Michael Kahn. 2019. An Interactive Data Quality Test Approach for Constraint Discovery and Fault Detection. In *IEEE Big Data*. 200–205.

[9] R. J. Hyndman, E. Wang, and N. Laptev. 2015. Large-Scale Unusual Time Series Detection. In *IEEE International Conference on Data Mining Workshop*. 1616–1619.

[10] Bogumil Kaminski, Michal Jakubczyk, and Przemyslaw Szufel. 2018. A Framework for Sensitivity Analysis of Decision Trees. *Central European Journal of Operations Research* 26, 1 (2018), 135–159.

[11] T. Kieu, B. Yang, and C. S. Jensen. 2018. Outlier Detection for Multidimensional Time Series Using Deep Neural Networks. In *19th IEEE International Conference on Mobile Data Management*. 125–134.

[12] R. M. Konijn and W. Kowalczyk. 2010. An Interactive Approach to Outlier Detection. In *Rough Set and Knowledge Technology*, Jian Yu, Salvatore Greco, Pawan Lingras, Guoyin Wang, and Andrzej Skowron (Eds.). Vol. 6401. Springer Berlin Heidelberg, 379–385.

[13] P. Kromkowski, S. Li, W. Zhao, B. Abraham, A. Osborne, and D. E. Brown. 2019. Evaluating Statistical Models for Network Traffic Anomaly Detection. In *Systems and Information Engineering Design Symposium*. 1–6.

[14] Nikolay Laptev, Saeed Amizadeh, and Ian Flint. 2015. Generic and Scalable Framework for Automated Time-series Anomaly Detection. In *21th ACM International Conference on Knowledge Discovery and Data Mining*. 1939–1947.

[15] G. Loganathan, J. Samarabandu, and X. Wang. 2018. Sequence to Sequence Pattern Learning Algorithm for Real-Time Anomaly Detection in Network Traffic. In *IEEE Canadian Conference on Electrical Computer Engineering*. 1–4.

[16] H. Lu, Y. Liu, Z. Fei, and C. Guan. 2018. An Outlier Detection Algorithm Based on Cross-Correlation Analysis for Time Series Dataset. *IEEE Access* 6 (2018), 53593–53610.

[17] Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. 2017. Chapter Six - Mutation Testing Advances: An Analysis and Survey. *Advances in Computers* 112 (2017), 275–378.

[18] D. Park, Y. Hoshi, and C. C. Kemp. 2018. A MultimodalAnomaly Detector for Robot-Assisted Feeding Using an LSTM-Based Variational Autoencoder. *IEEE Robotics and Automation Letters* 3, 3 (2018), 1544–1551.

[19] Kun Il Park. 2017. *Fundamentals of Probability and Stochastic Processes with Applications to Communications* (1st ed.). Springer Publishing Company, Incorporated.

[20] UCI ML Repository. 2020. NASA Shuttle. Retrieved March 7, 2020 from https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle)

[21] L. A. Shalabi and Z. Shaaban. 2006. Normalization as a Preprocessing Engine for Data Mining and the Approach of Preference Matrix. In *International Conference on Dependability of Computer Systems*. 207–214.

[22] Priyanga Dilini Talagala, Rob J. Hyndman, Kate Smith-Miles, Sevvandi Kandanaarachchi, and Mario A. Munoz. 2019. Anomaly Detection in Streaming Nonstationary Temporal Data. *Journal of Computational and Graphical Statistics* (2019), 1–21.

[23] Yahoo Server Traffic. 2020. A Benchmark Dataset for Time Series Anomaly Detection. Retrieved March 30, 2020 from https://yahooresearch.tumblr.com/post/114590420346/a-benchmark-dataset-for-time-series-anomaly

[24] TSfeatures. 2020. CRAN. Retrieved March 5, 2020 from https://cran.r-project.org/web/packages/tsfeatures/vignettes/tsfeatures.html

[25] B. Wang, Z. Wang, L. Liu, D. Liu, and X. Peng. 2019. Data-Driven Anomaly Detection for UAV Sensor Data Based on Deep Learning Prediction Model. In *2019 Prognostics and System Health Management Conference*. 286–290.

[26] Qingsong Wen, Jingkun Gao, Xiaomin Song, Liang Sun, Huan Xu, and Shenghuo Zhu. 2018. RobustSTL: A Robust Seasonal-Trend Decomposition Algorithm for Long Time Series. In *33rd AAAI Conference on Artificial Intelligence*. 5409–5416.

[27] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. 2019. A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures. *Neural Computation* 31, 7 (2019), 1235–1270.

[28] Chuxu Zhang, Dongjin Song, Yuncong Chen, Xinyang Feng, Cristian Lumezanu, Wei Cheng, Jingchao Ni, Bo Zong, Haifeng Chen, and Nitesh V. Chawla. 2019. A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data. In *33rd AAAI Conference on Artificial Intelligence*. 1409–1416.

[29] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep Learning over Multi-field Categorical Data. In *ECIR*. 45–57.

[30] Chong Zhou and Randy C. Paffenroth. 2017. Anomaly Detection with Robust Deep Autoencoders. In *23rd ACM International Conference on Knowledge Discovery and Data Mining*. 665–674.