An Autocorrelation-based LSTM-Autoencoder for Anomaly Detection on Time-Series Data

Hajar Homayouni, Sudipto Ghosh, Indrakshi Ray, Shlok Gondalia
Department of Computer Science
Colorado State University
{hhajar,ghosh,iray}@colostate.edu,
shlok@rams.colostate.edu

Jerry Duggan
Energy Institute
Colorado State University
jerry.duggan@colostate.edu

Michael G. Kahn
Anschutz Medical Campus
University of Colorado
michael.kahn@cuanschutz.edu

***Abstract*—Data quality significantly impacts the results of data analytics. Researchers have proposed machine learning based anomaly detection techniques to identify incorrect data. Existing approaches fail to (1) identify the underlying domain constraints violated by the anomalous data, and (2) generate explanations of these violations in a form comprehensible to domain experts. We propose IDEAL, which is an LSTM-Autoencoder based approach that detects anomalies in multivariate time-series data, generates domain constraints, and reports subsequences that violate the constraints as anomalies. 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 and brute-force approaches. The anomalies are visualized in a manner comprehensible to domain experts in the form of decision trees extracted from a random forest classifier. Domain experts can then provide feedback to retrain the learning model and improve the accuracy of the process. We evaluate the effectiveness of IDEAL using datasets from Yahoo servers, NASA Shuttle, and Colorado State University Energy Institute. We demonstrate that IDEAL can detect previously known anomalies from these datasets. Using mutation analysis, we show that IDEAL can detect different types of injected faults. We also demonstrate that the accuracy improves after incorporating domain expert feedback.**

***Index Terms*—Anomaly detection, Autocorrelation, Data quality tests, Explainability, LSTM-Autoencoder, Time series**

I. Introduction

Voluminous time-series data are increasingly collected from various real-world sources, such as Internet of Things (IoT) sensors, network servers, and patient medical flow reports [1]–[3]. Incorrect data collection, data transformation, intrusions and malicious insider attacks can corrupt a time-series dataset and result in incorrect analysis. Existing anomaly detection approaches [4]–[7] for individual data records cannot be used on time-series data as anomalies involve constraints over multiple attributes and records in a time series [8].

Existing Machine Learning (ML) based approaches [9], [10] that discover constraints in time-series data have limitations. First, the window size used for analysis is typically fixed-sized [11] or calculated using an exhaustive brute-force approach [12]. 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. Second, the discovered constraints are in the form of complex equations incomprehensible to domain experts.

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 [10] that discovers constraints involving long-term non-linear associations among multivariate time-series data records and attributes. Subsequences and records that violate the constraints are flagged as suspicious. Domain experts inspect them and provide feedback to reduce false alarms [13] and retrain the learning model. We believe IDEAL is the first approach to find anomalies in multivariate time-series big data, explain them in terms of constraints violations using domain concepts, and illustrate the anomalous records and sequences using decision trees.

We proposed an autocorrelation-based windowing technique that automatically adjusts the window size based on how far the records are related to their past values. This process is faster than brute-force window sizing. It results in a higher constraint-discovery effectiveness of the LSTM-Autoencoder model than when manually set fixed sizes are used.

We evaluated the constraint discovery, anomaly detection, and anomaly explanation effectiveness of IDEAL using datasets from Yahoo servers [14], NASA Shuttle [15], and Colorado State University Energy Institute [16]. We compared the anomaly detection effectiveness with existing stochastic and ML-based anomaly detection techniques. We also compared the effectiveness and efficiency of autocorrelation-based windowing with a brute-force windowing approach. We used mutation analysis to show that the true positive rate and false negative rate improve after incorporating ground truth knowledge about the injected faults and retraining the interactive-based LSTM-Autoencoder model. We showed that the generated visualization plots help domain experts understand the reason behind the reporting of the suspicious sequences.

The rest of the paper is organized as follows. Section II summarizes the background on time-series data. Section III describes the related work. Section IV describes IDEAL and Section V presents its evaluation. Section VI concludes the paper and outlines directions for future work.

This work was supported in part by funding from NSF under Award Numbers CNS 1650573, CNS 1822118, OAC 1931363, CableLabs, Furuno Electric Company, SecureNok, AFRL, and NIST.
978-1-7281-6251-5/20/$31.00 ©2020 IEEE

II. Background on Time Series

A time series T is a sequence of d-dimensional records [1] described using the vector $T =< R_0, ..., R_{n-1} >$, where $R_i = (a_i^0, ..., a_i^{d-1})$ is a record at time i, for $0 \leq i \leq n-1$ and a_i^j is the j^{th} attribute of the i^{th} record.

A time series can be *univariate* (d=1) or *multivariate* (d>1) [2]. 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 from a climate data store can consist of precipitation, wind speed, snow depth, and temperature data.

Various features [17], [18] are used to describe the relationships among the time-series records and attributes of which *trend* and *seasonality* [19] are the most common. Trend is defined as the general tendency of a time series to increment, decrement, or stabilize over time [19]. Seasonality is defined as the existence of repeating cycles in a time series [19]. A time series is *stationary* (non-seasonal) if all its statistical features, such as mean and variance are constant over time. We use 18 features defined by Talagala et al. [17] to (1) identify the types of constraint violations reported by IDEAL (see Section IV-D), and (2) define mutation operators that violate constraints spanning these features (see Section V-A).

A constraint is defined as a rule over the time-series features. For example, the *mean* value of the daily electricity (*deliveredKWH*) delivered to a household (data classification "Residential") must be in the range 0-20 kWh.

We categorize the data in a time series T that violates these constraints as *anomalous records* and *anomalous sequences*. An anomalous record R_t is one whose observed value is significantly different from the expected value of T at t. Given a set of subsequences $T = \{T_0, ..., T_{m-1}\}$ in T, an anomalous sequence $T_j \in T$ is one whose behavior is significantly different from the majority of subsequences in T.

III. Related Work

Machine Learning-based techniques used for outlier detection in non-sequence data, such as Support Vector Machine (SVM) [4], Local Outlier Factor (LOF) [5], Isolation Forest (IF) [6], and Elliptic Envelope (EE) [7] have also been used to detect anomalous records from time series data [20]. Such approaches do not consider temporal dependencies between data records and can only detect trivial out-of-range outliers.

Techniques that detect anomalous records from time-series data can be categorized as *decomposition* and *modeling* techniques. Decomposition techniques, suitable only for univariate time series, break a time series into level, trend, seasonality, and noise components and monitor the noise components to capture the anomalous records [21], [22]. Modeling techniques represent 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 Moving Average (MA) [23], Autoregressive Integrated Moving Average (ARIMA) [24], and Holt-Winters (HW) [25] 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 make them inapplicable to many practical problems [19]. Machine learning modeling techniques support non-linear modeling, with no assumption about the distribution of the data [19]. Examples are Multi Layer Perceptrons (MLPs) [26], Long Short Term Memory (LSTM) [9], and Hierarchical Temporal Memory (HTM) [27]. Some of these techniques can model multivariate time-series. However, they produce complex equations, which are not human interpretable.

Existing techniques for anomalous sequence detection split the data into multiple subsequences, typically based on a fix-sized window [11] or an exhaustive brute-force approach [12]. 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 and 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 on 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 extends an autoencoder for time series data, and captures long-term temporal associations among data records in the form of complex equations that are not human interpretable.

IV. Proposed Approach

We illustrate our approach using the Yahoo server traffic datasets in the Yahoo Webscore program [14] and the NASA Shuttle dataset in the UCI ML repository [15]. The Yahoo server traffic datasets contain real and synthetic univariate time series, each of which contains 1,420 records with one time-dependent attribute called *traffic_value*. These datasets contain time series with random seasonality, trend and noise. The multivariate NASA Shuttle dataset contains 58,000 time-ordered records with eight time-dependent numerical attributes.

Figure 1 shows an overview of our 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*.

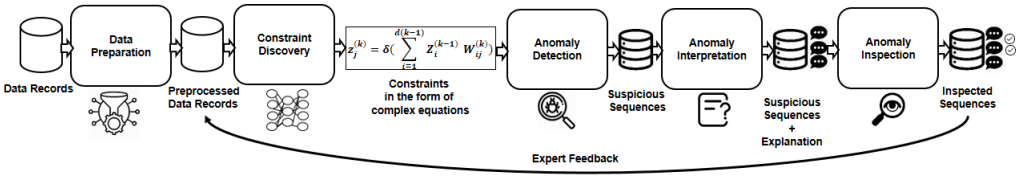


Fig. 1: IDEAL Overview

A. Data Preparation

IDEAL extracts features as complex dependencies among the input records and attributes, and discovers constraints as complex equations over those features [28]. To transform the input dataset into a form suitable for analysis, we use one-hot encoding [29] and normalization [30] for preprocessing the categorical and numeric attributes respectively.

The LSTM-Autoencoder input is a matrix with three dimensions, namely, *batch size*, *window size*, and *attribute size*. Batch size 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 attribute size 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 shows the input to an LSTM-Autoencoder, where the batch size is equal to one, window size is equal to w, and attribute size 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 records in subsequences of the input sequence. The length of the subsequence (window size) determines how far back the network connects a data record to its past values, and affects the correctness of the constraints discovered by the network [12]. For example, 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 [11]. A small window size can result in missing constraints involving dependencies among the records, while a large window size can result in a large increase in the computational complexity of the network. Increasing the size based on an exhaustive brute-force approach until the network error is minimized can be impractical for real-world big datasets [12].

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

Autocorrelation is defined as the correlation of sequence data records with the records in the previous time steps, called lags [31]. 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 shifted by k lags, and $\overline{D[a]}$ is the average value of attribute a over all times in the dataset. The numerator is the covariance between the data and the k-unit lagged data. The denominator is the sum of the squared deviations of the original dataset. An $ACF(a,k)$ value that rises above or falls below a confidence interval is said to be significantly autocorrelated. The shaded area in Figure 2 shows the confidence interval (CI) calculated by Eq. 2 for attribute A_4 in the NASA Shuttle dataset.

$$CI = \pm Z_{1-\alpha/2}\sqrt{\frac{1}{N}(1 + 2\sum_{i=1}^{k} ACF(a,k)^2)} \quad (2)$$

with lag k, sample size N, cumulative distribution function z of the standard normal distribution, and significance level α. The confidence bands increase as the lag increases.

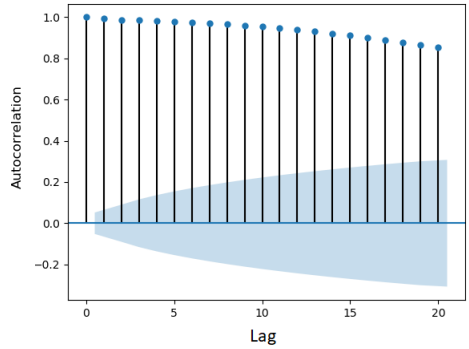


Fig. 2: ACF for A_4 Attribute in NASA Shuttle for 20 lags

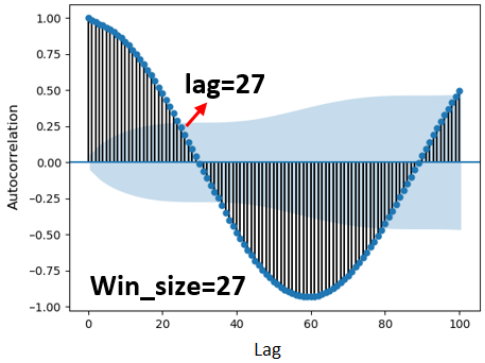


Fig. 3: Use ACF to Select Window Size

In Figure 2, the height of each spike shows the value of ACF for the corresponding lag. Autocorrelation with a lag of zero (i.e., between each record an itself) is always equal to 1.

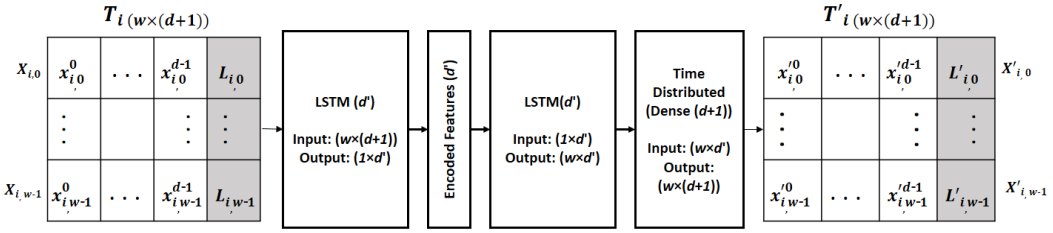


Fig. 4: Extending LSTM-Autoencoder by Adding a Label Input

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 A_4 attribute are highly correlated to its 20 past values.

The ACF is calculated for all the attributes. For each attribute a_i, IDEAL selects the lag value l_i after which ACF 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 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.

B. 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. Thus, IDEAL uses an LSTM-Autoencoder, which is a sequence-to-sequence modeling technique [28] used to learn time series dependencies.

Figure 4 shows the LSTM-Autoencoder architecture. The input and output are fixed-size matrices. $X_j = [x^0_j, ..., x^d_j, L_j]$ 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 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). 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. IDEAL uses grid search [32] to set the LSTM-Autoencoder hyper-parameters. The reconstruction error (RE) [33] for this network is:

$$RE = \frac{1}{m}\sum_{i=0}^{m-1}(T'_i - T_i)^2 \tag{3}$$

where T_i and T'_i are the i^{th} network input and output and m is the total number of windows.

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 an additional input to the network structure. The shaded area in Figure 4 shows the extension. In Section IV-E, 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.

$$RE = \frac{1}{m}\sum_{i=0}^{m-1}((T'_i - T_i)^2 + (mean(L'_i) - mean(L_i))^2) \tag{4}$$

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.

C. Anomaly Detection

We use suspiciousness scores (s-scores) to identify suspicious subsequences, records, and attributes that violate the constraints discovered by the LSTM-Autoencoder network. The scores are 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.

a) s-score_per_attribute: Assigned to each attribute (Eq. 5), it indicates the attribute's contribution to the invalidity of the subsequence. This value lies between 0 and 1.

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

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

b) s-score_per_record: Assigned to each record (Eq. 6), it indicates the contribution of each record to the invalidity of the subsequence. This value lies between 0 and 1.

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

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.

c) s-score_per_subsequence: This value indicates the level of invalidity of the subsequence (Eq. 7).

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

where s_score_i is the score assigned to the i^{th} subsequence and $s\_score^j_i$ is the s-score_per_attribute in that subsequence. In Eq. 7, $(\frac{1}{d}\sum_{j=0}^{d-1} s\_score^j_i)$ is a value between 0 and 1, and $max(L_i) \in \{-1, 0, 0.5, 1\}$ is equal to the maximum

value of the record labels in the subsequence. In the retraining phase, a subsequence with all valid records (labeled -1) gets an $s\_score \leq 0$ and is not reported as suspicious. A subsequence with at least one invalid record (i.e., with label 1) gets an $s\_score \geq 1$ and is marked as suspicious.

TABLE I: Suspicious Subequence Detected in NASA Dataset

Id	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	*s*-score
101	0	78	0	24	24	42	55	14	0.031
102	0	100	0	34	-23	64	67	4	0.029
103	0	79	0	28	10	42	50	8	0.005
104	1	83	0	36	0	45	46	2	0.002
105	0	77	0	6	-22	40	72	32	0.0078
106	0	80	5	10	0	43	70	26	0.004
107	0	78	0	2	-10	41	75	34	0.007
108	0	77	0	8	11	40	69	30	0.006
109	0	107	0	30	3	70	76	6	0.045
110	0	77	7	20	-6	41	57	16	0.003

D. Anomaly Explanation

To help domain experts inspect suspicious subsequences, we generate visualizations. To highlight the contribution of each record to the invalidity of a suspicious subsequence, we display a table showing the *s*-score_per_record. We 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 [34].

a) s-score_per_record.: Table I shows the records in a suspicious subsequence from the NASA Shuttle Dataset. Columns A_1–A_8 are the attributes. The last column shows the *s*-score_per_record values. IDEAL highlights the records (yellow row) that are major causes for invalidity of the subsequence using a threshold T (Eq. 8).

$$T = \begin{cases} P & if\ P > 0, \\ p(s\text{-}scores, 90) & otherwise \end{cases} \quad (8)$$

where P is the percentage of previously known anomalies for the input dataset and p is the percentile function, which returns the value below which a given percentage of records in the dataset fall. If there is a set of previously known anomalies in the dataset, IDEAL detects at least $P\%$ of records as suspicious to ensure that all the previously detected anomalies are reported by the approach. We set the threshold to 10% for datasets with no known anomalies because the average percentage of known anomalies in the datasets used by this study as well as 26 other datasets [35] from the UCI repository is equal to 10%. Domain experts can change this value based on the knowledge of the validity of their datasets.

b) 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 A_2 and A_6 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 is more than one attribute to be compared based on the values of *s*-score_per_attribute.

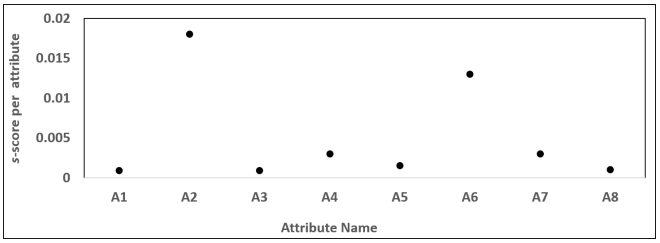


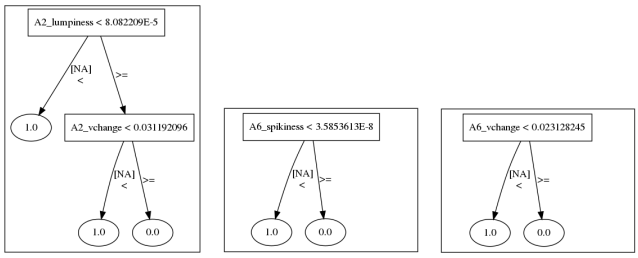
Fig. 5: *s*-score per Attribute Plot for Suspicious Subsequence Detected from NASA Shuttle Dataset

c) Decision tree.: This display describes the constraints violated by each of the suspicious subsequences. Decision trees were chosen because they are one of the easiest models to understand [36]. The non-leaf nodes and edges correspond to the subsequence features and their values respectively. 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 [34] generates a number of trees on various subsets of a dataset, whereas the basic decision tree classifier [36] 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 [37] to the training dataset.

For each attribute of the subsequence, 18 time series features are extracted using Tsfeatures [17] CRAN library. Next, decision trees are generated using a random forest classifier as a supervised technique, which requires labeled data. IDEAL labels the suspicious sequences as *invalid* and all the non-suspicious sequences as *valid* and uses this data. Three decision trees that have the lowest classification error are displayed via the tool's web interface. The number of decision trees displayed is configurable. The default number is three per subsequence to make manual inspection feasible.

The decision trees represent a set of if-then-else decision rules, which describe the constraints that identify sequences as valid or invalid based on their feature values. IDEAL uses random forest classifier on H2O [38] (i.e., a fully open-source, distributed in-memory machine learning platform) based on Classification And Regression Trees (CART) [39] algorithm, which builds binary trees (i.e., trees with nodes that have exactly two outgoing edges) for both numeric and categorical features. A decision tree is built top-down from a root node to the leaf nodes and involves partitioning the data into subsets that contain sequences with similar labels (i.e., homogeneous subsets). The CART algorithm uses an impurity criterion to calculate the homogeneity of a subset. At each level of the tree, the algorithm chooses a feature that results in subsets with the lowest impurity and splits the dataset into subsets based on the values of that feature. The algorithm repeats the same process on every branch of the tree until reaching the homogeneous subsets (i.e., leaf nodes with labels 0.0 or 1.0).

We tuned the maximum depth of the trees generated by the random forest to the value 5 based on our trial experiments; it became too hard for the domain experts to understand the constraints when we used values greater than 5.



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

Fig. 6: Decision Trees for Suspicious Sequence in NASA Shuttle Dataset

Figure 6 shows examples of decision trees generated for a suspicious subsequence in the NASA Shuttle dataset. The subsequence violates constraints over time series features. These constraints are specified in the form of "**If** $Pred$ **then sequence is** <valid|invalid>" rules, which determine whether a subsequence is valid or invalid based on its feature values. $Pred$ is in form of "$(f_1(a_1)\ ROP_1\ v_1)\ LOP_1$ ... LOP_{d-1} $(f_d(a_d)\ ROP_d\ v_d)$", where a_i is an attribute, f_i is a function that extracts feature f_i from the values of a_i, v_i is the value of that feature, ROP_i is a relational operator ($=$, $\geq$, $>$, $<$, $\leq$), and LOP_i is a logical operator (and and or). Below we show the violated constraints shown in Figure 6(a).

- **IF** lumpiness(A_2) $< 8.08e^{-5}$, **THEN** sequence is invalid
- **IF** lumpiness(A_2) $\geq 8.08e^{-5}$ **AND** vchange(A_2) < 0.03, **THEN** sequence is invalid

Decision trees apply to both types of time series. For a univariate time series, decision trees identify violated constraints using features extracted from the values of only sole attribute in the dataset. For a multivariate time series, the constraints are over features extracted from the values of all the time-dependent attributes.

Domain experts analyze the constraints represented in the trees and inspect the values of the time series features to determine whether a suspicious subsequence is actually anomalous.

E. Anomaly Inspection

This component uses a web-based user interface to take a domain expert's feedback about the suspicious subsequences. 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 update the labels of the training data records. The label is initially 0 for every record. The labels are updated based on the feedback. Records in a subsequence reported as suspicious by IDEAL are labeled 0.5, out of which those marked as 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 and improve its accuracy.

V. Evaluation

We evaluated the effectiveness of constraint discovery, anomaly detection, and anomaly explanation of IDEAL using the Yahoo server traffic datasets, the NASA Shuttle dataset, and the real-world Energy datasets in the Colorado State University's Smart Village Microgrid Lab at the Energy Institute [16]. The Energy multivariate dataset contains millions of time-ordered records with one categorical (*Classification*) and one numeric (*deliveredKWH*) attribute. This dataset merges values of electricity power delivered to different residential, commercial, and industrial premises in the city of Fort Collins, Colorado. The *classification* attribute stores values of premise type, which are "Residential", "Commercial", and "Industrial". The *deliveredKWH* attribute stores values of power for the premises. We used a set of previously known anomalies in these datasets to evaluate IDEAL and compare it to existing anomaly detection approaches. Due to the absence of complex anomalies that violate constraints over multiple time series features in these datasets, we used a mutation analysis technique to inject a set of complex anomalies into the data. In keeping with the spirit of traditional mutation analysis used in software testing [40], 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.

A. Mutation Analysis

We defined domain-independent mutation operators, each of which changes certain records or sequences in different ways with the goal of violating at least one constraint over the 18 time series features. These operators result in *mutants*, which are faulty records or sequences that mimic typical anomalies in the sequence data resulting from real-world events, such as sensor malfunctions and malicious insider attacks. A mutant is defined to be *killed* when the suspicious subsequences detected by IDEAL contain the mutant records or sequences. As a result of using the operators, all the features get invalid values, which violate constraints over the features. Table II shows the mutation operators, 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. We removed previously known anomalies from these datasets before applying the mutation operators.

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 below.

a) $\boldsymbol{M_1}$*–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. Noisy data can violate constraints on various time series features shown in column 3 of Table II. For example, the mean value of delivered power to households

TABLE II: Injected Faults and Violated Features

Mutation Operator	Fault Type	Violated Features
M_1: Add noise	Anomalous record	Mean, Variance, Lumpiness, Lshift, Vchange, Linearity, Curvature, Spikiness, BurstinessFF, Minimum, Maximum, Rmeaniqmean, Moment3, Highlowmu
M_2: Horizontal shift	Anomalous sequence	Mean, Variance, Lumpiness, Lshift, Vchange, Linearity, Spikiness, Seasonality, BurstinessFF, Minimum, Maximum, Moment3, Highlowmu, Trend
M_3: Vertical shift	Anomalous sequence	Mean, Linearity, Seasonality, Minimum, Maximum
M_4: Re-scale	Anomalous sequence	Mean, Linearity, Curvature, Seasonality, Minimum, Maximum, Moment3
M_5: Add dense noise	Anomalous sequence	Mean, Variance, Lumpiness, Lshift, Vchange, Linearity, Curvature, Spikiness, Seasonality, Peak, Trough, BurstinessFF, Minimum, Maximum, Rmeaniqmean, Moment3, Highlowmu, Trend

during the day is between 0 and 20 kWh. A sudden change in this value to 100 kWh indicates a violation of this constraint. Mutation operator M_1 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 $0 \leq \alpha \leq 10$

The maximum value of α is set to be 10 because the attributes of anomalous records in the NASA Shuttle and Yahoo server datasets contain values that are up to 10 times that of the valid attribute values.

Operators M_2–M_5 mutate a randomly selected subset of consecutive records containing between 5-10% of the records in the entire dataset, and create faulty subsequences.

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 $r = random(0.05 \times size(D), 0.1 \times size(D))$.

*b) **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, consider a constraint over the trend of the power usage in a school from 8 to 11 AM on weekdays. A shift in the school starting hour or attacker manipulation can result in violation on this constraint. Operator M_2 shifts attribute values of the records in the selected subset along the time axis. Empty cells are filled 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 $A = R_m[a]$.

*c) **M_3**–Vertical shift.:* A vertical shift can occur in real-world datasets as a result of a temporary malfunction of the sensors that capture the data. For example, a manipulation or malfunction of a temperature sensor may temporarily change the level of the value captured by the sensor. Operator M_3 adds a random value between the min and the max values of the attribute to all attribute values in the subset of records.

*d) **M_4**–Re-scale.:* Rescaling can occur in real-world datasets as a result of a temporary modification of the sensors or servers that capture the data. For example, if the unit of a snow depth detector sensor is temporarily changed from inches to centimeters, the values stored from the sensor will be 2.54 times greater than the expected values. Operator M_4 multiplies all the attribute values in the subset of records with a random number between the min and max values of that attribute.

*e) **M_5**–Add dense noise.:* Dense noise can get added to real-world datasets through attacks on the sensors or servers that capture the data. Operator M_5 changes all the attribute values in the subset of records to randomly selected values.

B. Evaluation Goals

We evaluated three aspects of IDEAL: (1) constraint discovery and anomaly detection effectiveness, (2) anomaly explanation effectiveness, and (3) performance. We calculated $F1$ scores [41] to demonstrate the effectiveness of different aspects of our approach. Given the number of positive samples, P, the number of true positives TP, the number of negative samples N, and the number of false positives FP, in a dataset, the $F1$ score is calculated as follows [41].

$$F1 = 2 \times \frac{(precision \times recall)}{(precision + recall)} \tag{9}$$

where $precision = \frac{TP}{(TP+FP)}$ and $recall = TPR = \frac{TP}{P}$. For mutation analysis, TPR represents the mutation score. TP is number of injected faulty subsequences reported as anomalies and P is the total number of injected faulty subsequences.

*1) **Goal 1.** Constraint discovery and anomaly detection effectiveness of IDEAL.:* We demonstrate this in four parts.

***RQ1.a:** How effective is IDEAL in the constraint discovery and anomaly detection on real-world Energy datasets when expert feedback is not used?*

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 calculated the $F1$ score at the time-series level ($F1_t$) to demonstrate the anomaly detection effectiveness of our approach. We used three Energy datasets Premise_41191, Premise_825588, and Premises_Combined with 175,272, 175,296, and 1,048,575 number of records to evaluate this aspect of IDEAL. The first two datasets store data of delivered power to two different Fort Collins premises in two years. The third dataset combines data of 13 residential and commercial premises from year 2015 to 2019. This dataset contains a set of previously known anomalies (0.05%) as a result of malfunctioned or incorrect reading of sensors.

It took 187, 245, and 8100 seconds to perform automated steps of IDEAL once against each of the three datasets respectively. There were two suspicious sequences detected for each of the Premise_41191 and Premise_825588 datasets. The results were validated by a domain expert. All the suspicious sequences for these two datasets were as a result of *unusual* but *valid* data. For example, a subsequence with a half an hour peak in the delivered power at 3 PM was reported as suspicious. The domain expert had knowledge about an

electric car being charged during that time and thus the subsequence was flagged as valid even though it was an unusual. All the actual anomalous subsequences could be detected by IDEAL ($TPR_t = 1$) from the last dataset. IDEAL detected 24 subsequences as suspicious, out of which 19 were actual faults and 5 were valid subsequences (i.e., subsequences that do not contain the previously known anomalies) incorrectly detected as faulty. The $F1_t$ score for this dataset was equal to 0.88.

RQ1.b: *How effective is IDEAL in comparison to existing anomaly detection approaches?*

Let P_r be the number of actual faulty records, TP_r be the number of actual faulty records marked as suspicious by the tool, N_r be the number of actual valid records, and FP_r be the number of valid records incorrectly marked as suspicious by IDEAL. We calculated the $F1$ score at the record level ($F1_r$) to demonstrate the anomalous record detection effectiveness of our approach in comparison to existing approaches. IDEAL was executed only once without using expert feedback.

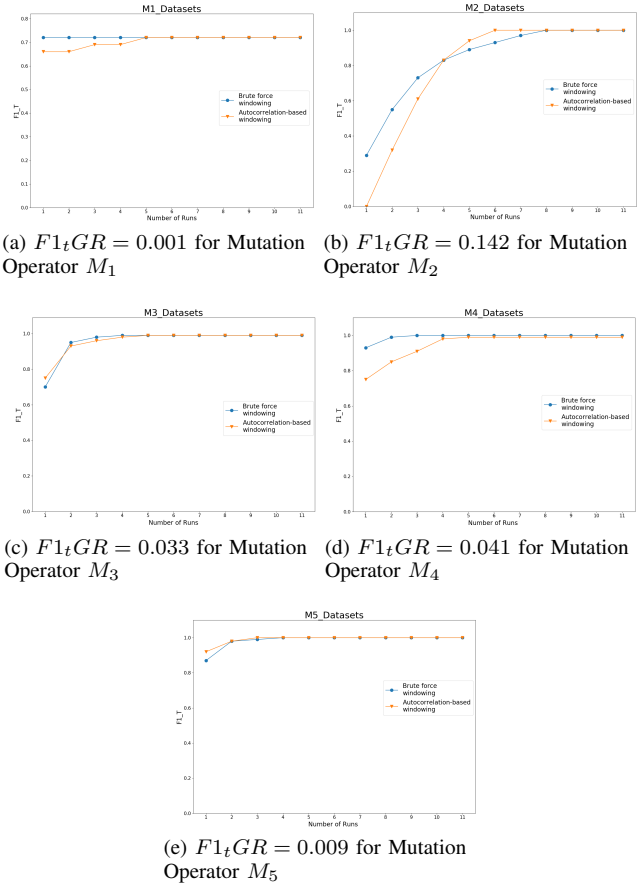
We compared IDEAL's $F1_r$ score with those of ARIMA, MA, HTM, and HW for univariate time series, which were evaluated by Hasani et al. [25] using the same univariate Yahoo synthetic datasets containing known anomalous records. Table III shows that IDEAL detected all the existing anomalies and was at least as effective as the existing approaches.

We also compared IDEAL's $F1_r$ score with those of the OneClass SVM, LOF, IF, EE, and SVM for multivariate time series, which were evaluated by Shriram and Sivasankar [20] using the same multivariate NASA Shuttle dataset containing known anomalous records. The last five columns of Table III shows the results. With 96% effectiveness in detecting the anomalous records, IDEAL was more effective than OneClass SVM, LOF, EE, and SVM. However, IDEAL's $F1_r$ score was 2% less than that of the IF approach. The previously defined anomalies in this dataset were trivial out-of-range outliers (i.e., records whose attributes have extremely large or small values in comparison with the majority of the records in the dataset). The IF approach effectively detected these outliers for non-sequence data. This shows that out-of-range outliers can be effectively detected from time-series data without considering the temporal dependencies between data records in the dataset.

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

This evaluation was performed using mutated datasets described in Table IV. Synthetic_1 and Synthetic_4 are univariate datasets that are mutated using the operators M_1–M_5. Column $|M|$ shows the number of mutants generated from each dataset. Column $|A|$ shows the number of attributes randomly selected from the NASA Shuttle dataset to apply the operators M_1–M_5.

We demonstrate the effectiveness of our windowing approach by comparing its $F1_t$ score with that of a brute force-based windowing approach. The window size is configurable in IDEAL. Using the mutated dataset as input, we executed IDEAL against the dataset multiple times using a range of window sizes to select the best window size that results in the highest $F1_t$ score for the brute-force approach. We also ran IDEAL using the autocorrelation-based windowing 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

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

Figure 7 shows a comparison of the results of these two configurations. $F1_t$ scores for autocorrelation-based windowing (orange line) and brute-force windowing (blue line) are shown for 10 runs (i.e., interactions using the feedback loop). In each run, we used the knowledge about the injected faults to automatically label the data and retrain the learning model. This labeling process simulates the help of an expert to inspect the results of every run and mark the data for the subsequent runs. The average of $F1_t$ scores are for the datasets in Table IV 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 $F1_t$ score using autocorrelation based windowing is close (i.e., within 0.04) to that using the brute force approach. On average, using autocorrelation-based windowing, the mutation score (TPR_t) for all the datasets is 0.60% and 0.94% for the first and last runs respectively. Using the brute-force approach, the corresponding scores are 0.64% and 0.94%.

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

TABLE III: $F1_r$ Scores of Different Approaches [20], [25] Using Yahoo Synthetic and NASA Shuttle Datasets

Dataset ID	IDEAL	ARIMA	MA	HTM	HW	OneClass SVM	LOF	IF	EE	SVM
Synthetic_1	1.00	0.66	0.73	1.00	1.00	-	-	-	-	-
Synthetic_2	1.00	1.00	1.00	0.80	1.00	-	-	-	-	-
Synthetic_3	1.00	0.50	0.40	1.00	1.00	-	-	-	-	-
Synthetic_4	1.00	0.57	0.50	1.00	1.00	-	-	-	-	-
Shuttle	0.96	-	-	-	-	0.87	0.72	0.98	0.85	0.82

TABLE IV: 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

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

$$F1_tGR = (\frac{F1_{tNR}}{F1_{t1}})^{\frac{1}{NR}} - 1 \quad (10)$$

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, which indicates that IDEAL's accuracy improves after using ground truth knowledge to retrain the learning model.

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 and N_a be the numbers of actual invalid and valid attributes that must and must not be reported for constraint violations. TP_a is the number of invalid attributes reported as valid by the decision tree report. FP_a is the number of valid attributes incorrectly reported as constraint violations. We measure the $F1$ score at the attribute level ($F1_a$).

RQ2.b: *Are the time series features reported in the constraint violations 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 the $F1$ score at the feature level ($F1_f$) to answer this question.

Table IV shows the $F1$ scores for one execution of IDEAL against 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). The $F1_a$ score is not applicable (NA) to univariate datasets (Synthetic_1 and Synthetic_4). The $F1_a$ scores were between 0.57 to 1.0 for the mutated Shuttle datasets. The $F1_f$ scores were between 0.28 to 0.8 for all the mutated datasets. The low values of $F1_f$ scores were as a result of displaying only three decision trees out of all those generated by the random forest classifier, which report fewer features in comparison to all the features identified as "must be reported" in Tables II.

3) ***Goal 3:*** *Performance of constraint discovery and anomaly detection.:* We answered the following question.

RQ3: *Is our autocorrelation-based windowing more efficient than the brute force approach?*

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

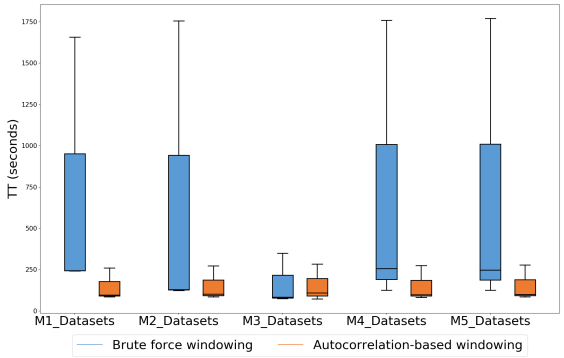


Fig. 8: TT Box Plots for Datasets Mutated by M_1–M_5

Figure 8 shows the box plots for TT in seconds using brute-force and auto-correlation based windowing for all the datasets based on a given mutation operator for one run of IDEAL. $M_i\_Datasets$ in the horizontal axis indicates the datasets that are mutated using the M_i operator. 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 value of the brute-force approach is affected more by factors such as the dataset size and its number and type of the attributes than the autocorreclation-based approach. On average, it takes 152.90 and 593.53 seconds to run the autocorrelation based and brute-force windowing approaches respectively. The autocorrelation-based approach is 3.88 times faster than the brute-force windowing approach.

C. Threats to Validity

Internal validity. Decision trees express constraint violations using domain attributes, which may not represent the con-

straints discovered by the LSTM-Autoencoder. We assume that the mutated datasets have no other faults than the seeded ones, and that the other datasets only have known anomalies.

External validity. The features selected for reporting constraint violations during mutation analysis were based on our observations when the Yahoo and NASA datasets were used. Different features may be affected in other datasets, which will affect the calculation of the $F1_f$ score.

VI. Conclusions and Future Work

We developed an LSTM-Autoencoder-based approach 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 previously known anomalies in the Energy dataset and also those 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 faster than the existing brute force window-size tuning approaches. In the future, we will evaluate the approach using other types of real-world time series data. We plan to extend IDEAL to find anomalies in streaming data.

References

[1] T. Kieu, B. Yang, and C. S. Jensen, "Outlier Detection for Multidimensional Time Series Using Deep Neural Networks," in *Proc. of MDM*, 2018, pp. 125–134.

[2] T. Guo, Z. Xu, X. Yao, H. Chen, K. Aberer, and K. Funaya, "Robust Online Time Series Prediction with Recurrent Neural Networks," in *Proc. of DSAA*, 2016, pp. 816–825.

[3] C. Zhang, D. Song, Y. Chen, X. Feng, C. Lumezanu, W. Cheng, J. Ni, B. Zong, H. Chen, and N. V. Chawla, "A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data," in *Proc. of AAAI*, 2019, pp. 1409–1416.

[4] Y. Chen and W. Wu, "Application of One-class Support Vector Machine to Quickly Identify Multivariate Anomalies from Geochemical Exploration Data," *Geochemistry: Exploration, Environment, Analysis*, vol. 17, no. 3, pp. 231–238, 2017.

[5] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying Density-Based Local Outliers," in *Proc. of SIGMOD/MOD*, 2000, p. 93–104.

[6] Z. Cheng, C. Zou, and J. Dong, "Outlier Detection Using Isolation Forest and Local Outlier Factor," in *Proc. of RACS*, 2019, p. 161–168.

[7] R. Thomas and J. Judith, "Voting-Based Ensemble of Unsupervised Outlier Detectors," in *Proc. of ComNet*, 2020, pp. 501–511.

[8] H. Lu, Y. Liu, Z. Fei, and C. Guan, "An Outlier Detection Algorithm based on Cross-Correlation Analysis for Time Series Dataset," *IEEE Access*, vol. 6, pp. 53 593–53 610, 2018.

[9] Y. Yu, X. Si, C. Hu, and J. Zhang, "A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures," *Neural Computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[10] P. Kromkowski, S. Li, W. Zhao, B. Abraham, A. Osborne, and D. E. Brown, "Evaluating Statistical Models for Network Traffic Anomaly Detection," in *Proc. of SIEDS*, 2019, pp. 1–6.

[11] D. Park, Y. Hoshi, and C. C. Kemp, "A MultimodalAnomaly Detector for Robot-Assisted Feeding Using an LSTM-Based Variational Autoencoder," *RA-L*, vol. 3, no. 3, pp. 1544–1551, 2018.

[12] B. Wang, Z. Wang, L. Liu, D. Liu, and X. Peng, "Data-driven Anomaly Detection for UAV Sensor Data Based on Deep Learning Prediction Model," in *Proc. of PHM*, 2019, pp. 286–290.

[13] R. M. Konijn and W. Kowalczyk, "An Interactive Approach to Outlier Detection," in *Proc. of RSKT*. Springer Berlin Heidelberg, 2010, vol. 6401, pp. 379–385.

[14] "Yahoo Server Traffic: A Benchmark Dataset for Time Series Anomaly Detection," https://yahooresearch.tumblr.com/post/114590420346/a-benchmark-dataset-for-time-series-anomaly (Accessed 2020-07-01).

[15] "NASA Shuttle," https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle) (Accessed 2020-05-15).

[16] "Energy Data," https://energy.colostate.edu/ (Accessed 2020-05-03).

[17] P. D. Talagala, R. J. Hyndman, K. Smith-Miles, S. Kandanaarachchi, and M. A. Munoz, "Anomaly Detection in Streaming Nonstationary Temporal Data," *JCGS*, pp. 1–21, 2019.

[18] "TSfeatures from CRAN Library," https://cran.r-project.org/web/packages/tsfeatures/vignettes/tsfeatures.html (Accessed 2020-05-15).

[19] R. Adhikari and R. K. Agrawal, *An Introductory Study on Time Series Modeling and Forecasting*. LAP, 2013.

[20] S. Shriram and E. Sivasankar, "Anomaly Detection on Shuttle data using Unsupervised Learning Techniques," in *Proc. of ICCIKE*, 2019, pp. 221–225.

[21] R. J. Hyndman, E. Wang, and N. Laptev, "Large-scale Unusual Time Series Detection," in *Proc. of ICDM*, 2015, pp. 1616–1619.

[22] N. Laptev, S. Amizadeh, and I. Flint, "Generic and Scalable Framework for Automated Time-series Anomaly Detection," in *Proc. of KDD*, 2015, pp. 1939–1947.

[23] C. Kuster, Y. Rezgui, and M. Mourshed, "Electrical Load Forecasting Models: A Critical Systematic Review," *SCS*, vol. 35, pp. 257–270, 2017.

[24] P. M. Maçaira, A. M. T. Thomé, F. L. C. Oliveira, and A. L. C. Ferrer, "Time Series Analysis with Explanatory Variables: A Systematic Literature Review," *Environmental Modelling & Software*, vol. 107, pp. 199 – 209, 2018.

[25] Z. Hasani, B. Jakimovski, G. Velinov, and M. Kon-Popovska, "An Adaptive Anomaly Detection Algorithm for Periodic Data Streams," in *Proc. of IDEAL*, 2018, pp. 385–397.

[26] C. M. Bishop, *Neural Networks for Pattern Recognition*. Clarendon Press, 1995.

[27] J. Wu, W. Zeng, and F. Yan, "Hierarchical Temporal Memory method for time-series-based anomaly detection," *Neurocomputing*, vol. 273, pp. 535 – 546, 2018.

[28] G. Loganathan, J. Samarabandu, and X. Wang, "Sequence to Sequence Pattern Learning Algorithm for Real-Time Anomaly Detection in Network Traffic," in *Proc. of CCECE*, 2018, pp. 1–4.

[29] W. Zhang, T. Du, and J. Wang, "Deep Learning over Multi-field Categorical Data," in *ECIR*, 2016, pp. 45–57.

[30] L. A. Shalabi and Z. Shaaban, "Normalization as a Preprocessing Engine for Data Mining and the Approach of Preference Matrix," in *Proc. of DepCoS*, 2006, pp. 207–214.

[31] K. I. Park, *Fundamentals of Probability and Stochastic Processes with Applications to Communications*, 1st ed. Springer Publishing Company, Incorporated, 2017.

[32] J. Bergstra and Y. Bengio, "Random Search for Hyper-parameter Optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[33] C. Zhou and R. C. Paffenroth, "Anomaly Detection with Robust Deep Autoencoders," in *Proc. of KDD*, 2017, pp. 665–674.

[34] B. Kaminski, M. Jakubczyk, and P. Szufel, "A Framework for Sensitivity Analysis of Decision Trees," *EJOR*, vol. 26, no. 1, pp. 135–159, 2018.

[35] "Outlier Detection Datasets," http://odds.cs.stonybrook.edu/ (Accessed 2020-06-30).

[36] P. B. de Laat, "Algorithmic Decision-Making based on Machine Learning from Big Data: Can Transparency Restore Accountability," *Philosophy & Technology*, vol. 31, no. 4, pp. 525–541, 2018.

[37] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2011.

[38] "H2o Random Forest," http://h2o-release.s3.amazonaws.com/h2o/master/1752/docs-website/datascience/rf.html, (Accessed 2020-07-01).

[39] N. Bhargava, S. Dayma, A. Kumar, and P. Singh, "An Approach for Classification Using Simple CART Algorithm in WEKA," in *Proc. of ISCO*, 2017, pp. 212–216.

[40] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. L. Traon, and M. Harman, "Chapter six - Mutation Testing Advances: An Analysis and Survey," *Advances in Computers*, vol. 112, pp. 275–378, 2017.

[41] M. Kay, S. N. Patel, and J. A. Kientz, "How Good is 85%? A Survey Tool to Connect Classifier Evaluation to Acceptability of Accuracy," in *Proc. of CHI*, 2015, p. 347–356.