DISSERTATION

ANOMALY DETECTION AND EXPLANATION IN BIG DATA

Submitted by

Hajar Homayouni

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2021

Doctoral Committee:

Advisor: Sudipto Ghosh
Co-Advisor: Indrakshi Ray

James M. Bieman
Indrajit Ray
Leo R. Vijayasarathy

ABSTRACT

ANOMALY DETECTION AND EXPLANATION IN BIG DATA

Data quality tests are used to validate the data stored in databases and data warehouses, and to detect violations of syntactic and semantic constraints. Domain experts grapple with the issues related to the capturing of all the important constraints and checking that they are satisfied. The constraints are often identified in an ad hoc manner based on the knowledge of the application domain and the needs of the stakeholders. Constraints can exist over single or multiple attributes as well as records involving time series and sequences. The constraints involving multiple attributes can involve both linear and non-linear relationships among the attributes.

We propose ADQuaTe as a data quality test framework that automatically (1) discovers different types of constraints from the data, (2) marks records that violate the constraints as suspicious, and (3) explains the violations. Domain knowledge is required to determine whether or not the suspicious records are actually faulty. The framework can incorporate feedback from domain experts to improve the accuracy of constraint discovery and anomaly detection. We instantiate ADQuaTe in two ways to detect anomalies in non-sequence and sequence data.

The first instantiation (ADQuaTe2) uses an unsupervised approach called autoencoder for constraint discovery in non-sequence data. ADQuaTe2 is based on analyzing records in isolation to discover constraints among the attributes. We evaluate the effectiveness of ADQuaTe2 using real-world non-sequence datasets from the human health and plant diagnosis domains. We demonstrate that ADQuaTe2 can discover new constraints that were previously unspecified in existing data quality tests, and can report both previously detected and new faults in the data. We also use non-sequence datasets from the UCI repository to evaluate the improvement in the accuracy of ADQuaTe2 after incorporating ground truth knowledge and retraining the autoencoder model.

The second instantiation (IDEAL) uses an unsupervised LSTM-autoencoder for constraint discovery in sequence data. IDEAL analyzes the correlations and dependencies among data records to discover constraints. 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 of the approach improves after incorporating ground truth knowledge about the injected faults and retraining the LSTM-Autoencoder model.

The novelty of this research lies in the development of a domain-independent framework that effectively and efficiently discovers different types of constraints from the data, detects and explains anomalous data, and minimizes false alarms through an interactive learning process.

ACKNOWLEDGEMENTS

I would like to thank my advisors, Prof. Sudipto Ghosh and Prof. Indrakshi Ray, for their guidance in accomplishing this project. I also wish to thank the members of my committee, Prof. James M. Bieman, Prof. Indrajit Ray, and Prof. Leo R. Vijayasarathy for generously offering their time and guidance. I also thank Dr. Laura Moreno Cubillos and the Software Engineering group for their constructive comments during my presentations.

I am grateful to Prof. Michael Kahn for his support and feedback. I also thank the CSU Plant Diagnostic Clinic that gave us access to their data and Dr. Ana Cristina Fulladolsa Palma for her feedback. I thank Jerry Duggan from the CSU Energy Institute who helped us get access to the Energy data and gave us feedback on our tool outputs. I would like to express my gratitude to the Computer Science staff for their help throughout my study at Colorado State University.

This research was supported by grants from the Anschutz Medical Campus at University of Colorado Denver. It was also supported in part by the US National Science Foundation (OAC-1931363, CNS-1650573, and CNS-1822118) together with funding from AFRL, Cable Labs, Furuno Electric Company, and SecureNok.

DEDICATION

To my parents.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Enterprises use databases and data warehouses to store, manage, access, and query the data for making critical decisions. Records can get corrupted because of how the data is collected, transformed, and managed, and also because of malicious activities. Incorrect records may violate constraints pertaining to the attributes and records. Inaccurate data can lead to incorrect decisions. Thus, rigorous data quality testing approaches are required to ensure that the data is correct.

Data quality tests validate the data in data stores to check for violations of syntactic and semantic constraints. Syntactic constraint validations check for the conformance of an attribute with the structural specifications in the data model. For example, in a health data store, *patient_age* must take numeric values. Semantic constraint validations check for the conformance of the record and attribute values with the specifications stated by domain experts. Semantic constraints can exist over single attributes (e.g., *patient_age* $>= 0$) or multiple attributes (e.g., *pregnancy_status* = *true* $\rightarrow$ *patient_gender* = *female*). Moreover, these constraints can exist over multiple records in time-series data. For example, semantic constraint validations check that the *patient_weight* growth rate change is positive and in the range [4, 22] lb for every infant. The validations also check for the relationship between the *patient_weight* and *blood_pressure*, and their growth rates over time for the adult patients.

1.1 Problem Statement

Data quality tests rely on the specification of constraints, which are typically identified by domain experts but often in an ad hoc manner based on their knowledge of the application domain and the needs of the stakeholders. For example, a data record in a health data store may contain an incorrect value for the day's supply of a drug. However, the constraint that restricts the values for the drug may be missing. Incorrect values in attributes pertaining to medications and prescriptions can have disastrous consequences for both patient health and research outcomes if

the data is used for patient treatment and in medical research [9]. Tools that automatically generate syntactic constraints also exist, but they only check for trivial ones, such as the not-null and uniqueness checks [10]. Existing machine learning-based approaches can automatically discover some non-trivial semantic constraints from the data and report the anomalous records as outliers [11]. However, these approaches do not explain which constraints are violated by those records. As a result, domain experts have to validate a huge number of outliers to determine whether or not they are actually faulty and to find the reason behind the invalidity of those records. Moreover, these approaches have the potential to learn incorrect constraints pertaining to the invalid data and generate false alarms, which can make the anomaly inspection process overwhelming for domain experts [12] especially when the size of the data is large.

1.2 Proposed Approach

We propose ADQuaTe as an **A**utomated **D**ata **Qua**lity **Te**st framework that provides generic functionality for constraint discovery and anomaly detection, which we instantiate to develop specific applications for non-sequence and sequence data. ADQuaTe automatically discovers complex semantic constraints from the data in a flat data model (i.e., a model that consists of a single, two-dimensional array of data records), marks records/sequences that violate the constraints as suspicious, and explains the violations. ADQuaTe uses unsupervised deep learning techniques based on autoencoders [13] to discover the constraints associated with the unlabeled records (i.e., records whose validity is not known in advance).

ADQuaTe assigns a suspiciousness score (*s*-score) to each record/sequence. Records/sequences whose *s*-score is greater than a threshold are flagged as suspicious. Decision trees are generated using a Random Forest classifier [14] to identify the constraints violated by the suspicious records/sequences.

While domain expert intervention is not required for using our framework, ADQuaTe can minimize false alarms through an interactive learning process [15], which incorporates domain expert feedback (when available) to improve the accuracy of the approach. ADQuaTe provides a web-

based interface to allow domain experts to inspect the suspicious records and sequences and mark records and sequences that are actually faulty. This feedback is incorporated to retrain the machine learning model and improve the accuracy of constraint discovery and anomaly detection.

ADQuaTe uses a grid search-based technique to select the best learning model for constraint discovery. The original grid search technique for autoencoders selects the model that generates the lowest value of reconstruction error [13]. This model has the potential to overfit on the training data and generate false alarms. We propose to use ground truth data with a set of known faults to select a model that maximizes the true positive rate. We measure the true positive rate for different deep network architectures to select a network with the highest accuracy. Moreover, we use an early stopping technique [16] based on the true positive rate to avoid overfitting on training data.

We instantiate ADQuaTe for non-sequence data (ADQuaTe2 [17,18]) using an autoencoder [13] as an unsupervised deep learning technique to discover the constraints involving both linear and non-linear relationships among the data attributes. Records that do not conform to the discovered constraints are flagged as suspicious. To reduce the time needed to inspect a large number of suspicious records, the Self Organizing Map (SOM) clustering technique is used to identify a small number of record groups such that the records in each group are likely to violate the same constraints. ADQuaTe2 uses a grid search technique based on ground truth data to tune the autoencoder parameters.

We instantiate ADQuaTe for sequence data (IDEAL [19]) using an LSTM-Autoencoder [20] to discover complex constraints from univariate or multivariate time-series data in big datasets. IDEAL reports subsequences that violate the constraints as anomalies. We propose an automated autocorrelation-based windowing approach to adjust the LSTM-Autoencoder network input size, thereby improving the correctness and performance of constraint discovery over manual approaches.

1.3 Evaluation

We have implemented the components of ADQuaTe in an open-source web-based tool [21]. We evaluated the constraint discovery and fault detection effectiveness of ADQuaTe2 without domain expert intervention using datasets from a health data warehouse [22] and a plant diagnosis database [23]. We demonstrated that our approach can discover new constraints that were missed by domain experts and can also detect new faults in these datasets. We also evaluated the improvements in the accuracy of ADQuaTe2 using datasets with ground truth data (i.e., a set of known faults) from the UCI repository [2]. We measured the fault detection effectiveness and efficiency of ADQuaTe2 using this data as well. We showed that ADQuaTe2 can detect the previously known faults in these datasets and the accuracy of the approach improves after incorporating the ground truth knowledge and retraining the learning model. We demonstrated that the true positive rate increases and the false negative rate decreases after incorporating the ground truth knowledge and retraining the learning model.

We evaluated the constraint discovery, anomaly detection, and anomaly explanation effectiveness of IDEAL using the Yahoo server [24], NASA Shuttle [25], and Energy [26] datasets. We compared the anomaly detection effectiveness of IDEAL with existing stochastic and Machine Learning-based anomaly detection techniques. Moreover, we compared the effectiveness and efficiency of our autocorrelation-based reshaping approach with a brute-force approach. Mutation analysis showed that the true positive 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.

1.4 Contributions

To the best of our knowledge, ADQuaTe is the first framework to find anomalies in both non-sequence and sequence big data and explain them in terms of constraints violations using domain concepts. The key contributions of this research are as follows:

- ADQuaTe uses unsupervised deep learning techniques that effectively discover from unlabeled data different types of constraints involving linear and non-linear associations among data records and attributes.

- ADQuaTe helps a domain expert interpret the detected anomalies by (1) highlighting the contribution of each record and attribute to the invalidity of the anomalies and (2) generating decision trees where paths indicate the constraints violated by the anomalies.

- ADQuaTe minimizes false alarms using expert feedback to retrain the machine learning model and improve the accuracy of constraint discovery and anomaly detection.

- ADQuaTe uses a grid search technique based on ground truth data to tune parameters of learning models in a way to avoid overfitting on training data.

- ADQuaTe uses an autocorrelation-based approach to automatically adjust the input size for the constraint discovery component to improve the effectiveness and efficiency of the component than when manually set fixed input sizes or brute-force approaches are used.

The rest of the dissertation is organized as follows. Chapter 2 describes related work on data quality test approaches and discusses their limitations. Chapter 3 presents ADQuaTe as our proposed data quality test framework. Chapter 4 and 5 describe instantiations of ADQuaTe for non-sequence and sequence data respectively and report on evaluating the instantiations. Finally, Chapter 6 concludes the dissertation and outlines the directions for future work.

Chapter 2

Related Work

We categorize existing data quality test approaches into two groups: approaches for testing non-sequence data and those for testing sequence data. This chapter summarizes the approaches and describes their limitations.

2.1 Data Quality Test Approaches for non-Sequence Data

Non-sequence data [27] is a set of unordered records. Large volumes of real-world non-sequence data are collected from various sources, such as patient medical reports and bank records. In this section, we describe existing data quality testing approaches for non-sequence data and classify them into two categories based on their constraint identification methods (manual and automatic).

2.1.1 Non-sequence Data

A non-sequence dataset *D* is a set of *d*-dimensional records described using the set $D = \{R_0, ..., R_{n-1}\}$, where $R_i = (a_i^0, ..., a_i^{d-1})$ is a record, for $0 \leq i \leq n-1$ and a_i^j is the j^{th} attribute of the i^{th} record. No order is assumed for the non-sequence data records by existing data analysis approaches [28].

A non-sequence dataset can have a single attribute (d=1) or multiple attributes (d>1). For example, a one-attribute breast cancer dataset [29] may contain values of *tumor size* for different patients. A multiple-attribute glass identification dataset [30] may contain the concentration values of different elements, such as *Sodium*, *Magnesium*, *Aluminum*, *Silicon*, *Potassium*, *Calcium*, *Barium*, and *Iron* that form a *glass type*.

A constraint for non-sequence data is defined as a rule over the data attributes. For example, the *tumor size* must be in a specific range for all breast cancer patients. Moreover, the value of

Sodium must be in the 10.73–17.38 range for *glass type=vehicle windows*. Anomalies are records that violate the constraints over single or multiple attributes in non-sequence data.

2.1.2 Approaches based on Manual Constraint Identification

There are two phases involved in these approaches [31–33]: (1) specifying data quality constraints and (2) generating test assertions.

2.1.2.1 Specify Data Quality Constraints

Syntactic and semantic constraints are specified by domain experts as mathematical formulas, natural language, and database queries. The following paragraphs describe the syntactic and semantic constraints from the papers published by Golfarelli and Rizzi [31], Gao et al. [32], Darkory et al. [33], and Kahn et al. [34].

- **Syntactic constraint**: This constraint specifies that the syntax of an attribute must conform to the data model used to describe the data in a store. This constraint is also called *data correctness* [32] and *data conformance* [34] in different papers. Examples of constraints imposed by the data model are data type and integrity.

  - Data type: A data type is a classification of the data that defines the operations that can be performed on the data and the way the values of the data can be stored [35]. The data type can be numeric, text, boolean, or date-time; these types are defined in different ways in different languages. For example, the *Sex* attribute of patient records takes one ASCII character.
  - Data integrity: A data integrity constraint imposes restriction on the values that an attribute or a set of attributes can take in a data store. Primary key, foreign key, uniqueness, and not-null constraints are typical examples. For example, a *Person_ID* attribute must take unique values.

- **Semantic constraint**: This constraint specifies the content of an attribute. The same constraint is also called *accuracy* [32, 33] and *plausibility* [34]. This constraint can exist over single or multiple attributes.

  - Single attributes: This constraint is defined as the conformance of individual attribute values to the application domain specification. For example, the *Sex* attribute in the previous example can take only 'M' for male, 'F' for female or 'U' for undefined values.

  - Multiple attributes: This constraint is defined as the conformance of an attribute content to the contents of other relevant attributes in the data store. This constraint is also called *data coherence* [31] and *logical constraint consistency* [36]. This constraint ensures that the logical relationships between multiple attributes are correct with respect to the business requirements. For example, *postal code=33293* does not apply to streets where *city=Berlin* since the postal codes in Berlin are between 10115 and 14199.

Quality Assurance (QA) by the National Weather Service (NWS) [37] and the US Forest Service's i-Tree Eco [38] are examples of approaches that rely on manual identification of the constraints for weather and climate data. Achilles [39], proposed by the Observational Health Data Sciences and Informatics (OHDSI) [40] community, and PEDSnet [41], proposed by the Patient-Centered Outcomes Research Institute (PCORI) [42], are examples of approaches for validating electronic health data. The *Data Quality Constraint* column in Table 2.1 presents examples of constraints that are specified using natural language for a weather data warehouse [38]. Such a data warehouse gathers observations from stations all around the world into a single data store to enable weather forecasting and climate change detection.

GuardianIQ [43] is a data quality test tool that does not define specific data quality constraints but allows users to define and manage their own expectations from the data in a data store as constraints for data quality. The GuardianIQ tool provides a user interface to define, browse, and

Table 2.1: Data Quality Constraints and Test Assertions for Weather Records

	Data Quality Constraint	**Query**
1	Relative_humidity must be in the range [0,1].	Select count(*Relative_humidity*) from *Weather_fact* where *Relative_humidity* > 1 or *Relative_humidity* < 0
2	Temperature must be a numeric value.	Select count(*Temperature*) from *Weather_fact* where data_type(*Temperature*)!= integer
3	Temperature must not be null.	Select count(*Temperature*) from *Weather_fact* where *Temperature* is null
4	If Rain_fall is greater than 80%, Relative_humidity cannot be zero.	Select count(*) from *Weather_fact* where *Rain_fall* > 0.8 and *Relative_humidity* = 0

edit a rule base in an editor. The example in Table 2.2 is a rule specified by a user to verify the consistency property in a customer data warehouse:

Table 2.2: A Data Quality Constraint Defined by a GuardianIQ User and the Corresponding Test Assertion

	Data Quality Constraint	**Query**
1	If the customer's age is less than 16, then the driver's license field should be null.	Insert into *tbl_test_results* (*status*, *description*) values ('Failed', 'Invalid value for driver's license') from *Customers* where (*age* < 16 and *driver_license* != null))

2.1.2.2 Generate Test Assertions

Data quality tests are defined as a set of queries that verify the constraints. The *Query* column in Table 2.1 shows data quality test assertions defined as queries to verify the constraints presented in the *Data Quality Constraint* column of the same table. After executing a query in this table, a positive value of count indicates that the corresponding assertion failed.

GuardianIQ [43] transforms declarative data quality constraints into SQL queries that measure data quality conformance with the user's expectations. The *Query* column in Table 2.2 is a SQL query that is automatically generated by the tool to implement the constraint in the *Data Quality Constraint* column of the same table. The tool executes the queries against the data and calculates to what extent the data matches the user's expectations. This tool allows sharing the constraints

across multiple users of the same domain with the same expectations. The interface allows users to quickly browse and easily edit the constraints.

2.1.3 Approaches based on Automated Constraint Identification

In these data quality test approaches, the constraints are automatically identified from the data. These approaches are based on Machine Learning (ML) techniques that can discover semantic constraints from the data. In this section, we describe ML-based approaches and discuss their specific challenges and open problems.

ML-based data quality test approaches have been proposed by researchers to detect anomalous records as outliers in the data [44]. Outliers are also referred to as abnormalities, discordants, deviants, and anomalies in the literature [11]. Depending on the availability of labeled data, these techniques can be classified as *supervised*, *semi-supervised*, and *unsupervised*.

2.1.3.1 Supervised Outlier Detection Techniques

These techniques train a binary classifier using a training dataset where the data records are labeled valid or invalid. The trained classifier is applied afterward to classify the unseen (testing) data records as valid or invalid. Examples of supervised outlier detection techniques are classification tree, Naive Bayesian, Support Vector Machine (SVM), and Artificial Neural Network (ANN).

Classification Tree [45–47]. This method uses a tree-structured classifier to label the records as valid and invalid. In this structure the non-leaf nodes correspond to the attributes, the edges correspond to the possible values of the attributes, and every leaf node contains the label of the records (0: *valid* and 1: *invalid*) described by the attribute values from the root node to that leaf node. Classification trees are one of the easiest to understand machine learning models [48]. One can analyze the tree to determine the constraints that are violated by each invalid record. However, these trees are prone to overfitting [49]. Random Forest [50] and Gradient Boosting [51] methods address overfitting by training multiple trees using independent random subsets taken from

the training data records. As a result, the chance for overfitting is reduced and the entire forest generalizes well to the new data records.

Naive Bayesian [52, 53]. This method uses a probabilistic classifier that calculates the probability of a data record belonging to a certain class. This classifier calculates $p(C|X)$ as the probability of belonging to a class $C \in \{valid, invalid\}$ given a data record *X* with n attributes. The objective is to determine whether $X \in\ valid$ or $X \in\ invalid$ based on the following decision rule.

$$X \in \begin{cases} valid & p(C = valid|X) > p(C = invalid|X) \\ invalid & otherwise. \end{cases} \tag{2.1}$$

According to the Bayes theorem [54], this decision rule can be rewritten as follows.

$$X \in \begin{cases} valid & \frac{p(X|C=valid)}{p(X|C=invalid)} \geq \frac{p(C=invalid)}{p(C=valid)} \\ invalid & otherwise. \end{cases} \tag{2.2}$$

The Naive Bayesian classifier assumes a strong independence between the record attributes. As a result, the $p(X|C)$ probability can be calculated based on the multiplication rule for independent events as $\prod_{i=1}^{n} p(X_i|C)$. The values of $p(X_i|C = valid)$ and $p(X_i|C = invalid)$ are computed using a training set of labeled records. In the Naive Bayesian classifier all the attributes independently contribute to the probability that a data record belongs to a class. However, attributes are typically related (i.e., not independent) in the real-world data sets. This approach cannot discover constraints that involve relationships among multiple related attributes.

Support Vector Machine (SVM) [55]. The objective of the SVM classifier is to train a hyperplane function in the attribute space that best divides a labeled data set into valid and invalid classes. The hyperplane is used afterwards to determine the class of each testing record based on the side of the hyperplane where it lands. Figure 2.1 shows a dividing hyperplane formed by an SVM for a simple outlier detection task. The data records in this example have only two attributes. The

records nearest to the hyperplane are called support vectors. These records are considered the critical elements of a data set because, if removed, the position of the hyperplane would change.

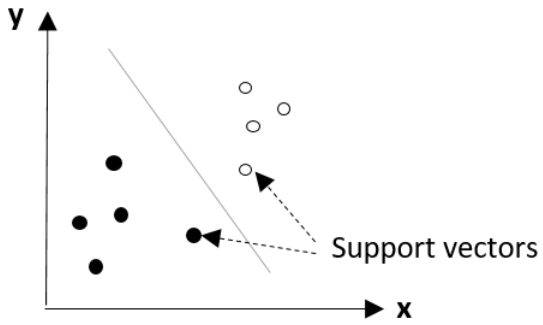


Figure 2.1: SVM Divides Valid/Invalid Records by a Linear Hyperplane [5]

The objective of the SVM is to position the hyperplane in a manner that the data records fall as far away from the hyperplane as possible, while remaining on the correct side. Unlike Figure 2.1, data records in typical data sets are not completely separated. When these records are hard to separate (Figure 2.2), the SVM method maps the data into a higher dimension. This approach is called kernelling. Figure 2.2 shows that the data records can now be separated by a plane. In the kernelling approach, the data continues to be mapped into higher attribute dimensions until a hyperplane can be formed to divide it.

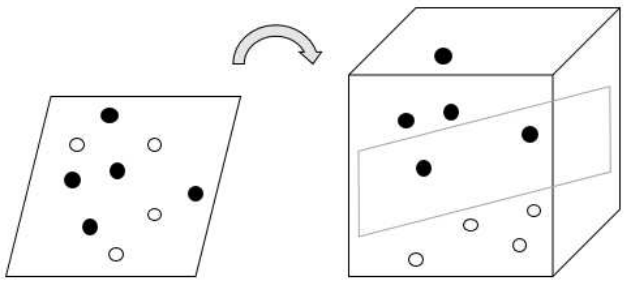


Figure 2.2: SVM Maps Records into a Higher Dimension [5]

SVM is applicable to both Linearly Separable and Non-linearly Separable labeled data records. However, the kernelling approach is sensitive to overfitting [56], especially when the generated

hyperplane is complex. Moreover, the trained hyperplane is an equation over data attributes that is not human interpretable.

Artificial Neural Network (ANN) [57, 58]. This method uses labeled records to train a network of information processing units that mimic the neurons of the human brain. The objective is to use this network to classify the testing records as valid and invalid. A neural network consists of an input layer, one or more hidden layers, and one output layer. Each layer includes a set of nodes. The node interconnections are associated with a scalar weight, which is adjusted during the training process. These weights are initialized with random values at the beginning of the training phase. Then, the algorithm tunes the weights with the objective of minimizing the error of mis-classification, which is measured based on the distance between the predicted label and the actual label of the data records. A neural network can be viewed as a simple mathematical function $f : X \rightarrow C$, where X is the input record with n attributes and C is the label assigned to the record by the network function. A widely used function is the nonlinear weighted sum of the input attributes, $\sigma(\sum_{i=1}^{n} x_i w_i)$, where σ is an activation function, such as hyperbolic tangent and sigmoid. An ANN is applicable to both Linearly Separable and Non-linearly Separable labeled data records. However, the trained network for labeling the testing data records is in the form of complex equations, which is not human interpretable.

2.1.3.2 Semi-supervised Outlier Detection Techniques

These techniques train a supervised learning model using the data that only consists of valid records [59]. The model of the valid data is used afterward to detect the outliers that deviate from that model in the testing data records. An example of the semi-supervised outlier detection techniques is one-class Support Vector Machine (OC-SVM).

One-Class Support Vector Machine (OC-SVM) [60, 61]. This method is a SVM-based classification technique that is trained only on the valid data. This method can be viewed as a regular two-class SVM where all the valid training data records lie in the first class, and the second class has only one member, which is the origin of the attribute space. This approach results in a hyper-

plane that captures regions where the probability density of the valid data lives. Thus, the function returns valid if a testing record falls in this region and invalid if it falls elsewhere. Like the two-class SVM classifier, this approach is applicable to both Linearly Separable and Non-linearly Separable data records. However, it is sensitive to overfitting and is not human interpretable.

2.1.3.3 Unsupervised Outlier Detection Techniques

These techniques [62] detect invalid records whose properties are inconsistent with the rest of the data in an unlabeled dataset. No prior knowledge about the data is required and there is no distinction between the training and testing data sets. Examples of the unsupervised techniques are clustering and representation learning.

Clustering [63, 64]. Clustering is an unsupervised learning technique that has been widely used to detect outliers. The constraints are investigated by grouping similar data into several categories. The similarity of the records is measured using distance functions, such as Euclidean and Manhattan distances. External outliers are defined as the records positioned at the smallest cluster. Internal outliers are defined as the records distantly positioned inside a cluster [65]. Distance-based clustering algorithms, such as *K*-prototypes [66] cannot derive the complex non-linear relationships that exist among attributes of the data in their clusters [67]. This is a problem in real-world applications, where non-linear associations are prevalent among the data attributes. Moreover, the clusters do not determine the violated constraints.

Local Outlier Factor (LOF [68]). LOF is an unsupervised technique that assigns to each data record a degree of being an outlier. This degree is called the local outlier factor of the record and is calculated based on how isolated the record is with respect to its surrounding neighborhood. LOF degree calculation is based on a fixed number of neighbors k. The approach compares the density of data records neighborhood (i.e., local density) to assign the LOF degree to the records. Data records that have a substantially lower density than their neighbors are considered to be anomalous. The local density is calculated by a typical distance measure. It is not straight forward to choose the correct value for the k parameter. A small value of k results in only consideration of nearby data

records in the degree calculation, which is erroneous in presence of noise in the data. A large value of k can miss local outliers. This approach is distance-based, which compares the records only with respect to their single attribute values and not based on the relationships among the attribute values. Moreover, the approach does not determine the constraints violated by the outliers.

Isolation Forest (IF [69]). Isolation Forest is an unsupervised anomaly detection technique that is built on an ensemble of binary decision trees called isolation trees. This technique isolates anomalous data records from valid ones. For this purpose, the technique recursively generates partitions on a dataset by (1) randomly selecting an attribute and (2) randomly selecting a split value for that attribute (i.e., between the minimum and maximum values of that attribute). This partitioning is represented by an isolation tree (Figure 2.3). The number of partitions required to isolate a point is equal to the length of the path from the root node to a leaf node (i.e., a data record) in the tree. As anomalous records are easier to separate (isolate) from the rest of the records, compared to valid records, data records with shorter path lengths are highly likely to be anomalous. This technique is faster than distance-based techniques, such as clustering and LOF, because it does not depend on computationally expensive operations like distance or density calculation [70]. The partitioning process is based on single attribute values and not on the relationships among the values. Moreover, this technique does not determine the violated constraints.

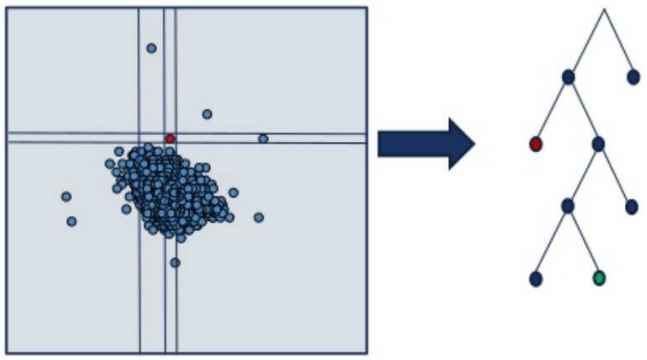


Figure 2.3: Isolation Forest for Anomaly Detection [6]

Elliptic Envelope (EE [71]). Elliptic Envelope is an unsupervised technique that fits a high dimensional Gaussian distribution [72] with possible covariances between attribute dimensions to the input dataset. Records that stand far enough from the fit shape are identified as anomalous. An ellipse is drawn around the data records, classifying any record inside the ellipse as valid and any record outside the ellipse as anomalous. A FAST-Minimum Covariance Determinate based on Mahalanobis distance [73] is used to estimate the size and shape of the ellipse. This technique assumes that the data comes from a known distributions, which is not practical for real-world datasets.

Representation Learning [74]. Representation learning is an unsupervised learning technique that investigates associations among the data attributes by capturing a representation of the attributes present in the data and flags as anomalous those records that are not well explained using the new representation. Principal Component Analysis (PCA) [12] is a representation learning approach that investigates the relationships among the data attributes by converting a set of correlated attributes into a set of linearly uncorrelated attributes called principal components. PCA representation learning can only investigate linear relationships among the data attributes, not non-linear ones. Moreover, the representations investigated by these methods are not human interpretable.

2.1.4 Summary

Table 2.3 summarizes and describes existing data quality test approaches in terms of their applicability and the steps they perform. The blank cells mean "not applicable to". We have identified the following open problems in testing the non-sequence data.

Inapplicable to multiple domains. Approaches based on manual constraint identification validate syntactic and semantic constraints that are specified by domain experts. These approaches are only applicable to a single domain. We propose a domain-independent approach based on machine learning techniques.

Lacking labeled data. Supervised ML-based techniques require labeled data for training the machine learning model. The existing data quality test approaches rely on manual labeling of

Table 2.3: Existing Data Quality Testing Approaches

Approach	Applicability		Steps			
	Domain-specific	Domain-independent	Constraint identification		Anomaly detection	Anomaly interpretation
			Manual	Automated		
Golfarelli and Rizzi [31] Dakrory et al. [33] Gao et al. [32]		✓	✓			
Kahn et al. [34]	✓		✓			
QA [37] i-Tree Eco [38] Achilles [39] PEDSnet [41]	✓		✓		✓	✓
GuardianIQ [43]		✓	✓		✓	
Informatica [75]		✓		✓	✓	
ML-based: Classification Tree [45–47] Naive Bayesian [52, 53] Support Vector Machines [55] Artificial Neural Network [57, 58] One-Class Support Vector Machine [60, 61] Clustering [63, 64] Representation Learning [12] Local Outlier Factor [68] Isolation Forest [69] Elliptic Envelope [71]		✓		✓	✓	

training data by the domain experts. Moreover, they restrict the training phase to a set of labeled data that are biased towards the domain expert's knowledge. Semi-supervised techniques require providing a clean data set for the training phase. These techniques are also biased towards the definition of valid records by the domain experts. We use an unsupervised technique that does not require labeled data.

Potential to generate false alarms. Unsupervised techniques have the potential to generate false alarms, which can make analysis overwhelming for a data analyst [12]. We propose to use an interactive learning-based Autoencoder to minimize the false alarms.

Lacking explanation. Unsupervised techniques report the anomalous records but do not determine the constraints that are violated by those records. However, specifying the reason behind invalidity is critical for domain experts to investigate the anomalies and prevent any further occurrences. We generate visualization diagrams of two types to describe the detected faults: (1) suspiciousness scores per attribute and (2) decision tree.

2.2 Data Quality Test Approaches for Sequence Data

Sequence data, also known as time-series data [76], is a set of time-ordered records [77]. 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 [77–79].

A time series *T* is a sequence of *d*-dimensional records [77] 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. Existing data analysis approaches [77] 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) [78]. 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 [80] can consist of precipitation, wind speed, snow depth, and temperature data.

The research literature [1, 82] uses various features that describe the relationships among the time-series records and attributes. Trend and seasonality [83] are the most commonly used features. Trend is defined as the general tendency of a time series to increment, decrement, or stabilize over

Table 2.4: Time Series Features [1]

Feature	Description
F_1: Mean	Mean value of time series
F_2: Variance	Variance value of time series
F_3: Lumpiness	Variance of the variances across multiple blocks in time series
F_4: Lshift	Maximum difference in mean between consecutive blocks in time series
F_5: Vchange	Maximum difference in variance between consecutive blocks in time series
F_6: Linearity	Strength of linearity, which is the sum of squared residuals of time series from a linear autoregression
F_7: Curvature	Strength of curvature, which is the amount by which a time series curve deviates from being a straight line and calculated based on the coefficients of an orthogonal quadratic regression
F_8: Spikiness	Strength of spikiness, which is calculated based on the size and location of the peaks and troughs in time series
F_9: Season	Strength of seasonality, which is calculated based on a robust STL [81] decomposition
F_{10}: Peak	Strength of peaks, which is calculated based on the size and location of the peaks in time series
F_{11}: Trough	Strength of trough, which is calculated based on the size and location of the troughs in time series
F_{12}: BurstinessFF	Ratio between the variance and the mean (Fano Factor) of time series
F_{13}: Minimum	Minimum value of time series
F_{14}: Maximum	Maximum value of time series
F_{15}: Rmeaniqmean	Ratio between interquartile mean and the arithmetic mean of time series
F_{16}: Moment3	Third moment, which is a quantitative measure that identifies the skewness of time series
F_{17}: Highlowmu	Ratio between the means of data that is below and upper the global mean of time series
F_{18}: Trend	Strength of trend, which is calculated based on a robust STL decomposition

time [83]. 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 [83]. For example, the sales of swimwear is higher during summers. A time series is *stationary* (non-seasonal) if all its statistical features, such as mean and variance are constant over time. Table 2.4 shows a set of features defined by Talagala et al. [1] to describe a time series.

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 range 0.1–0.5 KWH.

We categorize the faults that violates 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 $T = \{T_0, ..., T_{m-1}\}$ 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*. An anomalous 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.

Machine Learning-based techniques for outlier detection for non-sequence data, such as Support Vector Machine (SVM) [84], Local Outlier Factor (LOF) [68], Isolation Forest (IF) [69], and Elliptic Envelope (EE) [71] have been used in the literature to detect anomalous records from a time series [4]. These approaches discover the constraints in individual data records and cannot be used for testing time-series data as constraints may exist over multiple attributes and records in a time series. The records in a sequence have strong correlations and dependencies with each other, and constraint violations over multiple records cannot be discovered by analyzing records in isolation [85].

We classify the approaches that detect anomalies in time-series data into two groups based on anomaly types they can detect from input datasets; these are anomalous record detection and anomalous sequence detection. Figure 3 shows the classification framework we propose for anomaly detection techniques based on anomaly types they can detect in time-series data. The

framework presents *what* is detected in terms of anomaly types and *how* they are detected. A rounded rectangle represents a class and an edge rectangle represents a technique.

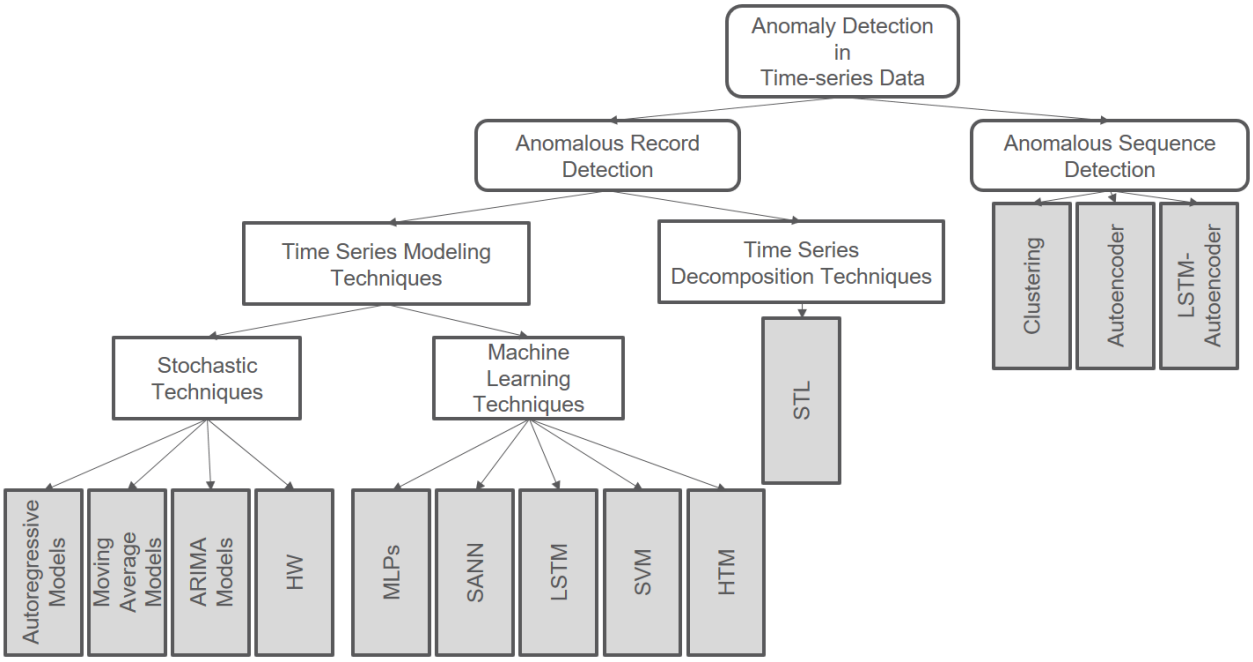


Figure 2.4: Classification Framework for Anomaly Detection Approaches for Sequence Data

2.2.1 Approaches to Detect Anomalous Records

We categorize these approaches based on how they analyze the time-series data as *time series modeling* and *time series decomposition* techniques.

2.2.1.1 Time Series Modeling Techniques

Given a time series $T = \{R_t\}$, these techniques model the time series as a linear/non-linear function f that associates current value of a time series to its past values. Next, the techniques use f to provide the predicted value of R_t at time t, denoted by R'_t, and calculate a prediction error $PE_t = |R_t - R'_t|$. The techniques report R_t as outlier if the prediction error falls outside a fixed threshold value. Every model f has a set of parameters, which are estimated using *stochastic* or *machine learning* techniques.

In the stochastic modeling techniques, a time series is considered as a set of random variables $T = \{R_t,\ t = 0, ..., n\}$, where R_t is from a certain probability model [83]. Examples of these techniques are *Autoregressive (AR)*, *Moving Average (MA)*, and *Autoregressive Integrated Moving Average (ARIMA)* and *Holt-Winters (HW)* models.

Autoregressive (AR) models [86]. In an Autoregressive model, the current value of a record in a time series is a linear combination of the past record values plus a random error. An autoregressive model makes an assumption that the data records at previous time steps (called as lag variables) can be used to predict the record at the next time step. The relationship between data records is called correlation. Statistical measures are typically used to calculate the correlation between the current record and the records at previous time steps. The stronger the correlation between the current record and a specific lagged variable, the more weight the autoregressive model puts on that variable. If all previous records show low or no correlation with the current one, then the time series problem may not be predictable [87]. Equation 2.3 shows the mathematical expression for an AR model.

$$R_t = \sum_{i=1}^{p} A_i R_{t-i} + E_t \tag{2.3}$$

where R_t is the record at time t and p is the order of the model. For example, an autoregressinve model of order two indicates that the current value of a time series is a linear combination of the two immediately preceding records plus a random error. The coefficients $A = (A_1, ..., A_p)$ are weights applied to each of the past records. The random errors (noises) E_t are assumed to be independent and following a Normal $N(0, \sigma^2)$ distribution. Given the time series T, the objective of AR modeling is to estimate the model parameters (A, σ^2). The linear regression estimators [88], likelihood estimators [89], and Yule-Walker equations [83] are typical stochastic techniques used to estimate this model parameters.

The AR model is only appropriate for modeling univariate stationary time-series data [83]. Moreover, it does not consider the non-linear associations between the data records in a time series.

Moving Average (MA) models [90]. In these models, a data record at time t is a linear combination of the random errors that occurred in past time periods (i.e. E_{t-1}, E_{t-2},...,E_{t-p}). Equation 2.4 shows the mathematical expression for an MA model.

$$R_t = \mu \sum_{i=1}^{p} B_i E_{t-i} + E_t \tag{2.4}$$

Where μ is the series mean, p is the order of the model, and $B = (B_1, ..., B_p)$ are weights applied to each of the past errors. The random errors E_t are assumed to be independent and following a Normal $N(0, \sigma^2)$ distribution.

The MA model is appropriate for univariate stationary time series modeling [83]. Moreover, it is more complicated to fit an MA model to a time series than fitting an AR model. Because in an MA model, the random error terms are not foreseeable [83].

Autoregressive Integrated Moving Average (ARIMA) models [86]. ARIMA is a mixed model, which incorporates: (1) Autoregression (AR) model, (2) an Integrated component, and (3) Moving Average (MA) model. The integrated component stationarized the time series by using transformations like differencing [91], logging [92], and deflating [93]. ARIMA can model time series with non-stationary behaviour. However, this model assumes that the time series is linear and follows a known statistical distribution, which makes it inapplicable to many practical problems [83].

Holt-Winters (HW [3]). This technique uses exponential smoothing [94] to model three features of a time series: (1) mean value, (2) trend, and (3) seasonality. Exponential smoothing assigns to past records exponentially decreasing weights over time. The objective is to decrease the weight put on older data records. Three types of exponential smoothing (i.e., triple exponential smoothing) are performed for the three features of a time series. The model requires multiple hyper-parameters: one for each smoothing, one for the length of a season, and one for the number of periods in a season. Hasani et al. [3] enhanced this technique (HW-GA) using a Genetic Algorithm [95] to optimize the HW hyper-parameters. The HW model is only appropriate for modeling

univariate time-series data. Moreover, it does not consider the non-linear associations between the data records in a time series.

In Machine Learning-based modeling techniques, a time series is considered to follow a specific pattern. Examples of these techniques are *Multi Layer Perceptron (MLP)*, *Seasonal Artificial Neural Networs (SANN)*, *Long Short Term Network (LSTM)*, and *Support Vector Machine (SVM)* models for big data and *Hierarchical Temporal Memory (HTM)* for streamed data (i.e., data captured in continuous temporal processes).

Multi Layer Perceptron (MLP) [96]. This technique is a type of Artificial Neural Network (ANN) [97], which supports non-linear modeling, with no assumption about the statistical distribution of the data [83]. An MLP model is a fully connected network of information processing units that are organized as input, hidden, and output layers. Equation 2.5 shows the mathematical expression of an MLP for time series modeling.

$$R_t = b + \sum_{j=1}^{q} \alpha_j g \left(b_j + \sum_{i=1}^{p} \beta_{ij} R_{t-i} \right) + E_t \tag{2.5}$$

where R_{t-i} $(i = 1, .., p)$ are p network inputs, R_t is the network output, α_j and β_{ij} are the network connection weights, E_t is a random error, and g is a non-linear activation function, such as logistic sigmoid and hyperbolic tangent.

The objective is to train the network and learn the parameters of the non-linear functional mapping f from the p past data records to the current data record R_t (i.e., $R_t = f(R_{t-1}, ..., R_{t-p}, w) + E_t$). Approaches based on minimization of an error function (equation 2.6) are typically used to estimate the network parameters. Examples of these approaches are Backpropagation and Generalized Delta Rule [97].

$$Error = \sum_{t} e_t^2 = \sum_{t} (R_t - R'_t)^2 \tag{2.6}$$

where R'_t is the actual network output at time t.

An MLP can model non-linear associations between data records. However, it is appropriate for univariate time series modeling. Moreover, because of the limited number of network inputs, it can only discover the short-term dependencies among the data records.

A Seasonal Artificial Neural Network (SANN) model is an extension of MLPs for modeling seasonal time-series data. The number of input and output neurons are determined based on a seasonal parameter *s*. The records in the i^{th} and (i+1)th seasonal period are used as the values of network input and output respectively. Equation 2.7 shows the mathematical expression for this model [83].

$$R_{t+l} = \alpha_l + \sum_{j=1}^{m} w_{1jl} g \left(\theta_j + \sum_{i=0}^{s-1} w_{0ij} R_{t-i} \right) \tag{2.7}$$

where $R_{t+l}(l = 1, .., s)$ are *s* future predictions based on the *s* previous data records ($R_{t-i}(i = 0, ..., s-1)$); w_{0ij} and w_{1jl} are connection weights from the input to hidden and from hidden to output neurons respectively; g is a non-linear activation function and α_l and θ_j are network bias terms.

This network can model non-linear associations in seasonal time-series data. However, it is appropriate for modeling univariate time series. Moreover, the values of records in a season are considered to be dependent only on the values of the previous season. As a result, the network can only learn short-term dependencies between data records.

Long Short Term Network (LSTM) [98]. An LSTM is a Recurrent Neural Network (RNN) [7] that contains loops in its structure to allow information to persist and make network learn sequential dependencies among data records [98]. An RNN can be represented as multiple copies of a neural network, each passing a value to its successor. Figure 2.5 shows the structure of an RNN [7]. In this Figure, *A* is a neural network, X_t is the network input, and h_t is the network output.

The original RNNs can only learn short-term dependencies among data records by using the recurrent feedback connections [78]. LSTMs extend RNNs by using specialized gates and memory cells in their neuron structure to learn long-term dependencies.

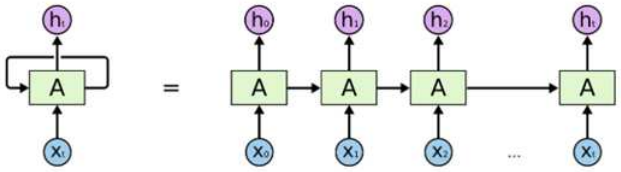


Figure 2.5: An Unrolled RNN [7]

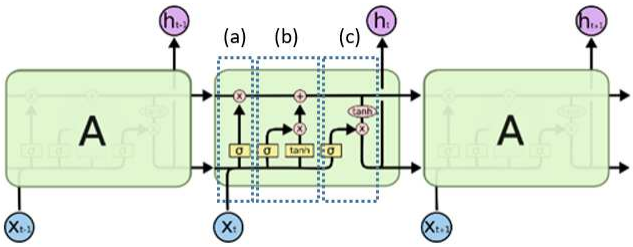


Figure 2.6: LSTM Structure [7]

Figure 2.6 shows the structure of an LSTM network. The computational units (neurons) of an LSTM are called *memory cells*. The horizontal line passing through the top of the neuron is called the memory cell state. An LSTM has the ability to remove or add information to the memory cell state by using *gates*. The gates are defined as weighted functions that govern information flow in the memory cells. The 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).

There are three types of gates, namely, *forget*, *input*, and *output*.

- *Forget gate* (Figure 2.6 (a)): Decides what information to discard from the memory cell. Equation 2.8 shows the mathematical representation of the forget gate.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{2.8}$$

where W_f is the connection weight between the inputs (h_{t-1} and x_t) and the sigmoid layer; b_f is the bias term and σ is the sigmoid activation function. In this gate, $f_t = 1$ means that completely keep the information and $f_t = 0$ means that completely get rid of the information.

- *Input gate* (Figure 2.6 (b)): Decides which values to be used from the network input to update the memory state. Equation 2.9 shows the mathematical representation of the input gate.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.9}$$

where C_t is the new memory cell state and C_{t-1} is the old cell state, which is multiplied by f_t to forget the information decided by the forget gate; $\tilde{C}_t$ is the new candidate value for the memory state, which is scaled by i_t as how much the gate decides to update the state value.

- *Output gate* (Figure 2.6 (c)): Decides what to output based on the input and the memory state. Equation 2.10 shows the mathematical representation of the output gate. This gate pushes the cell state values between -1 and 1 by using a hyperbolic tangent function and multiplies it by the output of its sigmoid layer to decide which parts of the input and the cell state to output.

$$\begin{aligned} o_t &= \sigma(W_o.[h_{t-1}, x_t] + b_o) \\ h_t &= o_t * tanh(C_t) \end{aligned} \tag{2.10}$$

An LSTM network for time series modeling takes the values of p past records ($R_{t-i},\ (i = 1, ..., p)$) as input and predicts the value of the current record (R_t) in its output. LSTM modeling techniques can model non-linear long-term sequential dependencies among the data records in univariate/multivariate time series, which makes them more practical for real-world applications. Moreover, LSTMs have the ability to learn seasonality [99]. However, the trained network is a complex equation over the attributes of the data records, which is not human interpretable.

Support Vector Machine (SVM [83]). An SVM model maps the data from the input space into a higher-dimensional feature space using a non-linear mapping (referred to as a Kernel Function) and then performs a linear regression in the new space. The linear model in the new space represents a non-linear model in the original space.

An SVM for time series modeling uses the training data as pairs of input and output, where an input is a vector of *p* previous data records in the time series and the output is the value of the current data record. Equation 2.11 shows the mathematical representation of a non-linear SVM regression model.

$$R_t = b + \sum_{p} \alpha_i \varphi(R_{t-i}) \tag{2.11}$$

where R_t is the data record at time t, φ is a kernel function, such as Gaussian RBF [100], and R_{t-i} is the i^{th} previous record in the time series.

The SVM modeling techniques can model both linear and non-linear functions for predicting time series values. However, these techniques require an enormous amount of computation, which makes them inapplicable to large datasets [83]. Moreover, the trained model is not human interpretable.

Hierarchical Temporal Memory (HTM [101]).

This is an unsupervised technique that continuously models time-series data using a memory based system. An HTM uses online learning algorithms, which store and recall constraints as spatial and temporal patterns in an input dataset. An HTM is a type of neural network whose neurons are arranged in columns, layers, and regions in a time-based hierarchy. This hierarchical organization considerably reduces the training time and memory usage because patterns learned at each level of the hierarchy are reused when combined at higher levels. The learning process of HTM discovers and stores spatial and temporal patterns over time. Once an HTM is trained with a sequence of data, learning new patterns mostly occurs in the upper levels of the hierarchy. An HTM matches an input record to previously learned temporal patterns to predict the next record. It takes longer for an HTM to learn previously unseen patterns. Unlike deep learning techniques that require large datasets to be trained, an HTM requires streamed data. The patterns discovered by this technique are not human interpretable.

2.2.1.2 Time Series Decomposition Techniques

These techniques decompose a time series into its components, namely level (the average value of data points in a time series), trend (the increasing or decreasing value in the time series), seasonality (the repeating cycle in the time series), and noise (the random variation in the time series) [102, 103]. Next, they monitor the noise component to capture the anomalies. These approaches report as anomalous the data record R_t whose absolute value of noise is greater than a threshold.

These techniques consider the time series as an additive or multiplicative decomposition of level, trend, seasonality, and noise. Equation 2.12 and 2.13 shows the mathematical representation of additive and multiplicative models respectively.

$$R_t = l_t + \tau_t + s_t + r_t \tag{2.12}$$

$$R_t = l_t * \tau_t * s_t * r_t \tag{2.13}$$

where R_t is the data record at time t, l_t is the level as the average value of data records in a time series, τ_t is the trend in time series, and s_t is the seasonal signal with a particular period, and r_t is the residual of the original time series after the seasonal and trend are removed and is referred to as *noise*, *irregular*, and *remainder*. In this model, s_t can slowly change or stay constant over time.

In a linear additive model the changes over time are consistently made by the same amount. A linear trend is described as a straight line and a linear seasonality has the same frequency (i.e., width of cycles) and amplitude (i.e., height of cycles) [104].

In a non-linear multiplicative model, the changes increase or decrease over time. A non-linear trend is described as a curved line and a non-linear seasonality has increasing or decreasing frequency or amplitude over time [104].

Different approaches are proposed in the literature to decompose a time series into its components. *Seasonal-Trend decomposition using LOESS* (*STL*) is one of the most commonly used approaches, which is described as follows.

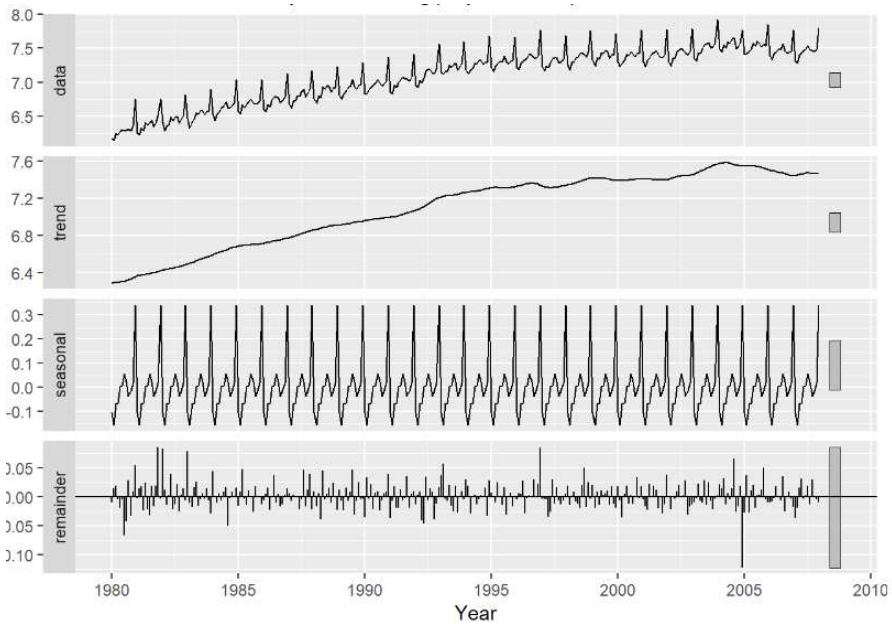


Figure 2.7: STL Decomposition of Liquor Sales Data [8]

Seasonal-Trend decomposition using LOESS (STL) [105]. This approach uses LOESS (LOcal regrESSion) smoothing technique to detect the time series components. LOESS is a non-parametric smoother that models a curve of best fit through a time series without assuming that the data must follow a specific distribution. This method is a local regression based on a least squares method; it is called local because fitting at point t is weighted towards the data nearest to t. The effect of a neighboring value on the smoothed value at a certain point t decreases with its distance to t. Figure 2.7 shows an example of the STL decomposition for a liquor sales dataset. This Figure shows the trend, seasonality, and noise components extracted from an original time-series data.

The time series decomposition techniques provide non-complex models that can be used to analyze the time-series data and detect anomalies in the data. However, in real-world applications, we may not be able to model a specific time series as an additive or multiplicative model, since

real-world datasets are messy and noisy [104]. Moreover, the decomposition techniques are only applicable to univariate time series data.

2.2.2 Approaches to Detect Anomalous Sequences

The approaches proposed in the literature to detect anomalous sequences are based on (1) splitting the time-series data into multiple subsequences, typically based on a fixed size overlapping window, and (2) detecting as anomalous those subsequences whose behavior is significantly different from the majority of subsequences in the time series. Examples of these approaches are *Clustering*, *Autoencoder*, and *LSTM-Autoencoder*.

Clustering [103]. These techniques extract subsequence features, such as trend and seasonality. Table 2.4 shows the time series features from the TSFeatures CRAN library [82]. Next, an unsupervised clustering technique, such as K-means [64] and Self-Organizing Map (SOM) [106] is used to group the subsequences based on the similarities between their features. Finally, *internal* and *external* anomalous sequences are detected. An internal anomalous sequence is a subsequence that is distantly positioned within a cluster. An external anomalous sequence is a subsequence that is positioned in the smallest cluster.

Distance-based clustering algorithms cannot derive relationships among multiple time series features in their clusters [67]. Moreover, these techniques only detect anomalous sequences without determining the records/attributes that are the major causes of invalidity in each sequence.

Autoencoder [77]. An autoencoder is a deep neural network that discovers constraints in the unlabeled input data. An autoencoder 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 representation 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 [13].

The anomalous sequence detection techniques based on autoencoders (1) take a subsequence (i.e., a matrix of m records and d attributes) as input, (2) use an autoencoder to reconstruct the sub-

sequence, (3) assign an invalidity score based on the reconstruction error to the subsequence, and (4) detect as anomalous those subsequences whose invalidity scores are greater than a threshold.

In an autoencoder network for anomalous sequence detection, the input (T_i) and output (T_i') are fixed-size subsequences. T_i is the i^{th} subsequence that contains w records, w is the window size, and $X_{i,j} = [x_{i,j}^0, ..., x_{i,j}^{d-1}]$ is the j^{th} record in T_i with d attributes. The network output has the same dimensionality as the network input. The encoder investigates the dependencies from the input subsequence and produces a complex hidden context (i.e., d' encoded features). The decoder reconstructs the subsequence from the hidden context and returns a subsequence with shape $(d*w)$. The reconstruction error for this network is defined as follows [13]:

$$RE = \frac{1}{m}\sum_{i=0}^{m-1}(T_i' - T_i)^2 \tag{2.14}$$

where T_i and T_i' are the i^{th} network input and output and m is the total number of subsequences.

These techniques can learn complex non-linear associations among data attributes in the time series as a result of using a deep architecture with several layers of non-linearity. However, these techniques are not able to model temporal dependencies among the data records in an input subsequence.

LSTM-Autoencoder [77]. An LSTM-Autoencoder is an extension of an autoencoder for time-series data using an encoder-decoder LSTM architecture. As described in Section 2.6, an LSTM network uses internal memory cells to remember information across long input sequences. As a result, an LSTM-Autoencoder can capture the temporal dependencies among the input records by using LSTM networks as the layers of the autoencoder network.

Figure 5.3 shows the LSTM-Autoencoder architecture. The input and output are fixed-size time series matrices. $X_{i,j} = [x_{i,j}^0, ..., x_{i,j}^{d-1}]$ is the j^{th} record with d 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

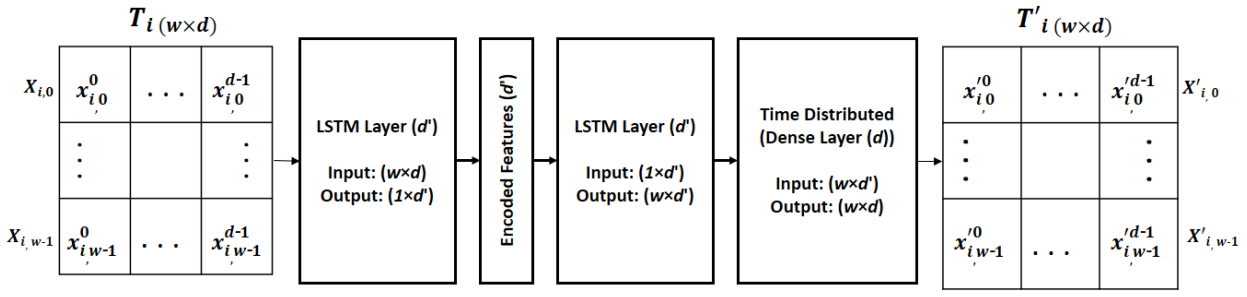


Figure 2.8: An LSTM-Autoencoder Network

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 [13]:

$$RE = \frac{1}{m}\sum_{i=1}^{m}(T'_i - T_i)^2 \tag{2.15}$$

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

These techniques can learn complex non-linear long-term associations among multiple data records and attributes as a result of using a deep network and the memory cells in their architecture. However, these associations are in the form of complex equations that are not human interpretable.

2.2.3 Summary

Table 2.5 summarizes different data quality test approaches for anomalous record and sequence detection. The blank cells mean "not applicable to". We have identified the following open problems in testing the time-series data.

Inapplicability to real-world time series. The stochastic time series analysis approaches only analyze univariate time-series data. Moreover, they assume that the time series is linear and follows

Table 2.5: Data Quality Test Approaches for Sequence Data

Approach	Time Series Type	Anomaly Type	Modeling Non-linearity	Modeling Seasonality	Modeling Long-term Dependencies
AR	Univariate	Records			
MA	Univariate	Records			
ARIMA	Univariate	Records			
SARIMA	Univariate	Records		✓	
HW	Univariate	Records		✓	
MLP	Univariate	Records	✓		
SANN	Univariate	Records	✓	✓	
LSTM	Mulivariate	Records	✓	✓	✓
SVM	Mulivariate	Records	✓		
HTM	Multivariate	Records	✓	✓	✓
STL	Univariate	Records	✓	✓	
Clustering	Univariate	Sequences	✓	✓	
Autoencoder	Multivariate	Sequences	✓		
LSTM-Autoencoder	Multivariate	Sequences	✓	✓	✓

a known statistical distribution. As a result, these classical time series modeling approaches do not apply to real-world multivariate time-series data with non-linear associations among data records and attributes. We propose to use a Machine Learning-based approach, which supports non-linear modeling, with no assumption about the statistical distribution of the data.

Unable to detect both anomaly types. Most of the existing stochastic (AR, MA, ARIMA, and SARIMA) and Machine learning-based approaches (MLP, SANN, LSTM, and SVM) can only detect anomalous records in time-series data. The approaches that detect anomalous sequences (clustering, autoencoder, and LSTM-Autoencoder) do not determine the anomalous records that are the major causes of invalidity in each sequence. We propose to assign a suspiciousness score to each record in an anomalous sequence to indicate the level of invalidity of the record in that sequence.

Unable to model long-term dependencies among data records. Most of the existing stochastic and Machine Learning-based approaches are unable to model long-term dependencies between

data records. These approaches model the time series as a linear or non-linear function that associates current value of a time series to a small number of its past values. We propose to use an LSTM-based approach with memory cells in its structure that can model long-term dependencies between the data records.

Potential to generate false alarms. The unsupervised learning approaches, such as Autoencoder and LSTM-Autoencoder have the potential to learn incorrect constraints pertaining to the invalid data records and sequences and generate false alarms. False alarms can make the anomaly inspection overwhelming for the domain experts [12]. We propose to use an interactive learning-based LSTM-Autoencoder to minimize the false alarms.

Lacking a systematic approach to set input size. In the existing Anomalous sequence detection approaches, constraints are discovered within an input subsequence, the size of which is typically selected based on a fixed-sized window [107] or by using an exhaustive brute-force approach [108]. 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. We propose a systematic autocorrelation-based windowing technique that automatically adjusts the input size based on how far the records are related to their past values.

Lacking explanation. The existing data quality test approaches for sequence data do not explain which constraints are violated by the anomalous sequences. Moreover, they do not determine the records or attributes that are major causes of invalidity of the anomalous sequences. We generate visualization diagrams of two types to describe the detected faults: (1) suspiciousness scores per attribute and (2) decision tree.

Chapter 3

ADQuaTe Framework

Section 3.1 presents examples datasets that are used to motivate the approach, describe the components, and demonstrate the usefulness of the framework. Section 3.2 describes the components in detail and Section 3.3 shows how parameters are tuned in ADQuaTe. Section 3.4 presents the specifications of the ADQuaTe tool and Section 3.5 describes the infrastructure we designed and implemented to run the evaluation experiments.

3.1 Running Examples

We use non-sequence data from health and plant diagnosis domains and sequence data from Yahoo traffic servers and NASA Shuttle datasets to motivate why a systematic approach is required for constraint discovery and anomaly detection. We show what happens when domain experts miss important constraints that must be checked in these domains.

3.1.1 Non-sequence Data

Health Data Compass [22] integrates patient clinical data from hospitals into a single target data warehouse to support research on diseases, drugs, and treatments. Table 3.1 shows the schema of the *Drug_exposure* table. Table 3.2 shows some of the constraints defined by the Observational Health Data Sciences and Informatics (OHDSI) [40] experts to check *Drug_exposure* data. Data quality tests are then created by the OHDSI developers as queries that verify the constraints. Table 3.3 shows examples of queries that check the constraints presented in Table 3.2. The data is validated against these constraints by running the queries whenever new data is added or previous data is modified. The results are *ID*s of the faulty records and the *Error_message*s that indicate a high-level description of the violated constraints.

Table 3.1: Schema of *Drug_exposure* Table

Attribute Name	Data Type	Description
ID	Numeric	Unique
Name	Text	Nullable
Unit	Text	Nullable
Route	Text	Nullable
Days supply	Numeric	Nullable
Dose	Numeric	Nullable
Quantity	Numeric	Nullable
Refills	Numeric	Nullable

Table 3.2: Constraints Defined for *Drug_exposure* Table

	Constraint name	Constraint
1	Implausible quantity	Drug quantity must be less than 600
2	Too high number of refills	Drug refills must be less than 10
3	Null concept	Drug name must not be NULL
4	Invalid concept	Drug name must be in predefined set of concepts

Table 3.3: Data Quality Tests for *Drug_exposure* Table

	Data Quality Test (Query)
1	Select *ID*, 'Implausible quantity' as *Error_message* from *Drug_exposure* where *Quantity* ≥ 600
2	Select *ID*, 'Too high number of refills' as *Error_message* from *Drug_exposure* where *Refills* ≥ 10
3	Select *ID*, 'Null concept' as *Error_message* from *Drug_exposure* where *Name* is null
4	Select *ID*, 'Invalid concept' as *Error_message* from *Drug_exposure* where *Name* is not in (Select *Concept_name* from *Concepts*)

A data record may contain an incorrect value for the day's supply of a specific drug. However, no constraint is defined in Table 3.2 to restrict the values for different drugs. Incorrect values in this attribute can have disastrous consequences for patients.

The Colorado State University Plant Diagnostic Clinic database [23] contains plant disease information and provides recommendations to clients, such as commercial growers and crop consultants. Table 3.4 shows the schema of the *Plant_diagnosis* table. A record may contain an

incorrect value for the diagnosis of a specific plant category. Incorrect values in plant diagnosis attributes have consequences for plant health and food security in the agricultural industry.

Table 3.4: Schema of *Plant_diagnosis* Table

Attribute Name	Data Type	Description
ID	Numeric	Unique
Host	Text	Nullable
Diagnosis ID	Text	Nullable
Genus Confirmation	Text	Nullable
Diagnosis Needed	Text	Nullable
Suspected Problem	Text	Nullable
Sample Category	Text	Nullable

3.1.2 Sequence Data

We use the Yahoo server traffic datasets in the Yahoo Webscore program [24], the NASA Shuttle dataset in the UCI ML repository [25], and a real-world Energy dataset from the CSU's Smart Village Microgrid Lab at the Energy Institute [26]. 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. Table 3.5 shows the schema of the Yahoo server traffic table. Anomalies in the *Traffic_value* indicate potential security threats to the Yahoo user's data.

Table 3.5: Schema of *Yahoo* Server Traffic Table

Attribute Name	Data Type	Description
ID	Numeric	Unique
Time	DateTime	Nullable
Traffic_value	Float	Nullable

The NASA Shuttle multivariate dataset contains 58,000 time-ordered records with eight time-dependent numerical attributes, namely, A_1 to A_8. Table 3.6 shows the schema of the NASA

Shuttle table. Incorrect values in these attributes have negative consequences for the aerospace industry that conducts research, designs, manufactures, operates, and maintains the spacecrafts.

Table 3.6: Schema of *NASA Shuttle* Table

Attribute Name	Data Type	Description
ID	Numeric	Unique
Time	DateTime	Nullable
A_1	Float	Nullable
A_2	Float	Nullable
A_3	Float	Nullable
A_4	Float	Nullable
A_5	Float	Nullable
A_6	Float	Nullable
A_7	Float	Nullable
A_8	Float	Nullable

The Energy multivariate dataset contains 1,048,575 time-ordered records with one categorical (*Classification*) and one numeric (*deliveredKWH*) attribute. This dataset merges values of power delivered by different residential and commercial premises in the city of Fort Collins, Colorado. The *classification* attribute stores values of premise type, which are "Residential" and "Commercial". The *deliveredKWH* attribute stores values of power in KWH for the premises. Table 3.7 shows the schema of the Energy table.

Table 3.7: Schema of *Energy* Table

Attribute Name	Data Type	Description
ID	Numeric	Unique
Time	DateTime	Nullable
Classification	Text	Nullable
deliveredKWH	Numeric	Nullable

3.2 ADQuaTe Components

Figure 3.1 shows an overview of ADQuaTe. The input is in the form of data records and the output consists of a report showing suspicious groups or sequences accompanied with an explanation of the violated constraints. ADQuaTe uses *data preparation*, *constraint discovery*, *anomaly detection*, *anomaly interpretation*, and *anomaly inspection* components with the following features.

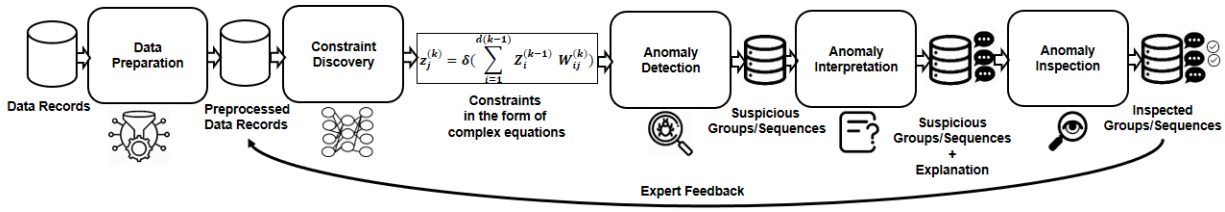


Figure 3.1: ADQuaTe Overview

3.2.1 Data preparation

This component automatically prepares the input data by transforming it from the raw format into a form that is suitable for analysis. Data attributes need to be preprocessed based on their types and values. We use the one-hot encoding [109] method for preprocessing the categorical attributes and normalization [110] method for the numeric attributes in both non-sequence and sequence datasets.

Categorical attributes. The attributes *Name*, *Unit*, and *Route* in Table 3.1 and *Host*, *Diagnosis ID*, *Genus Confirmation*, *Diagnosis Needed*, *Suspected Problem*, and *Sample Category* in Table 3.4 are categorical. Such attributes typically have string values, which cannot be directly used by the machine learning algorithms and must be converted to numeric values. Techniques that do not use preprocessing typically convert numeric string values to their corresponding numeric values and those that are not numeric to label-encoded numeric values. However, as a result, machine learning algorithms will treat all the attributes as numeric and incorrectly assume a natural ordering between the categories. For example, ordering should not be considered for the *Name* attribute

in the *Drug_exposure* table. One-hot encoding [111] addresses this issue by transforming each categorical attribute with *n* possible values into *n* binary attributes, with only one active value for each record. Due to the large number of new binary attributes generated from the categorical attributes, this component uses sparse matrices to store the values of the new attributes and to manage column explosion.

Numeric attributes. The attributes *Days Supply*, *Dose*, *Quantity*, and *Refills* in Table 3.1, *Traffic_value* in Table 3.5, and A_1 to A_8 in Table 3.6 are numeric. Numeric attributes typically vary in magnitude and range. The attributes with high magnitudes or ranges may carry a lot more weight in the learning calculations than the ones with low magnitudes or ranges [112]. To suppress this effect, we need to re-scale all the attributes to the same level of magnitude and range. This can be achieved by min-max scaling [110] that translates each attribute individually such that it is in the given range. Consequently, the attributes that have larger values but are of lower significance will not end up dominating the result. ADQuaTe scales each numeric attribute to a number between -1 and 1.

The sequence data must be transformed into the right shape (i.e., input size) for input to the next component. We instantiate this component for the sequence data using an *autocorrelation-based reshaping* approach that we propose to improve the correctness and performance of constraint discovery over manual approaches. This technique adjusts the input size based on how far the records are related to their past values. Section 5.1.1 describes this instantiation in detail.

3.2.2 Constraint discovery

This component obtains a trained model that represents different types of constraints in the unlabeled input data. The component uses unsupervised autoencoder-based deep learning techniques to discover different types of complex constraints associated with the unlabeled records. The unsupervised techniques remove the need for labels, which are typically unavailable for big datasets. Moreover, even if labels are used, they can be assigned to the records only based on known constraints.

An autoencoder-based network is composed of an *encoder* and a *decoder*. The encoder compresses the data from the input layer into a short representation. An input data X is mapped into a hidden representation Z with, which is non-linear combination of the input elements. The decoder decompresses Z into a new representation X' that closely matches the original data. X' is called a reconstruction of X. The network is trained to minimize the reconstruction error (RE), which is the average squared distance between the original data and its reconstruction [13].

$$RE = \frac{1}{n}\sum_{i=0}^{n-1}(X'_i - X_i)^2 \tag{3.1}$$

where *n* is the total number of inputs, X_i is the i^{th} network input, and X'_i is the i^{th} network output.

The constraints discovered by the deep learning models are in the form of non-linear equations that formulate the associations among data attributes and records and can be extracted from the trained model. These constraints are not human-interpretable. Further steps are required to explain the identified constraints to the domain experts.

This component uses unsupervised techniques 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 learning model and improve its accuracy. We extend the deep network architecture by adding a label as a new input to the network structure. In Section 3.2.5, 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 autoencoder-based network based on the labels to minimize false alarms. The network is trained to minimize both the difference between the input and its reconstruction, and the difference between the record labels and the labels predicted by the network.

We instantiate this component using an autoencoder [13] for non-sequence data and an LSTM-Autoencoder [20] for sequence data. An autonecoder uses a deep architecture that can model constraints involving both linear and non-linear relationships among data attributes. An LSTM-Autoencoder uses memory cells in its architecture that can discover constraints involving long-term

non-linear associations among data records and attributes. Sections 4.1.2 and 5.1.2 describe these instantiations in detail.

3.2.3 Anomaly detection

This component automatically detects suspicious records and sequences that do not conform to the constraints represented by the trained model. Unlike typical data quality testing approaches, anomaly classification in ADQuaTe is non-binary; each record or sequence is assigned a continuous suspiciousness score (*s*-score) between zero and one instead of being classified as valid or invalid. Records or sequences that do not conform to the discovered constraints (i.e., records or sequences whose *s*-score is greater than a threshold) are flagged as suspicious.

Autoencoder-based learning models discover constraints as associations among a majority of the data records. Anomalous records are usually in a minority in typical datasets [113] and are difficult to reconstruct by an autoencoder-based learning model that is trained using such datasets (i.e., the reconstruction error of the anomalous records are greater than the ones of the valid records) [113–116]. We had a similar experience with the health, plant, Yahoo server, NASA Shuttle, and UCI datasets, where the anomalous records were in a minority; the percentage of anomalous records was between 0.02% to 35.89%. As a result, we use the reconstruction error in this component to identify suspicious records or sequences. 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 sequences or records are reported as suspicious in the retraining phase, thereby minimizing false alarms.

The number of suspicious records may be too high for inspection by domain experts. ADQuaTe organizes these records in suspicious sequences for time-series data and in clusters for non-sequence data to make the anomaly inspection feasible. ADQuaTe uses a clustering technique called Self Organizing Map (SOM) [106] to group the non-sequence suspicious records based on their similarity (Section 4.1.3). This approach reduces the inspection time as the number of se-

quences and groups is far smaller than the number of records. The records in one sequence or group are likely to violate the same constraints.

3.2.4 Anomaly interpretation

The associations learned by the deep learning model are in the form of non-linear equations that are not human interpretable. To address this problem, we display *s*-score_per_record for the records in the suspicious group or sequence to highlight the contribution of each record to the invalidity of the group or sequence. 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 [14]. These plots help domain experts inspect the suspicious groups or sequences. Sections 4.1.4 and 5.1.4 describe how this component is instantiated for non-sequence and sequence data respectively.

***s*-score_per_record.** ADQuaTe uses the reconstruction error of the autoencoder-based network to calculate the invalidity level of a record in a group or sequence detected as suspicious by our approach. Sections 4.1.4 and 5.1.4 present how we calculate this value based on the reconstruction errors of the autoencoder and LSTM-Autoencoder for the non-sequence and sequence data respectively.

***s*-scores_per_attribute plot.** This plot displays the contribution of each attribute to the invalidity of a group or sequence. The horizontal axis indicates the attribute names and the vertical axis indicates their level of invalidity. The *s*-score of every attribute of a single record is extracted from the trained autoencoder-based model. The higher the value of *s*-score for an attribute, the more suspicious is the attribute, and as a result, the more likely is the attribute to contribute to the invalidity of the group or sequence. For each group or sequence we generate a plot showing the *s*-score values for all the attributes in the group or sequence.

Decision Tree. This display describes the constraints violated by each of the suspicious groups or sequences. Decision trees are one of the easiest to understand models [48]. In a decision tree structure, the non-leaf nodes correspond to variables (i.e., attributes of records or features

of sequences), the edges correspond to the possible values of the variables, and every leaf node contains the label of the path described by the variable values from the root to that leaf node.

A random forest [14] generates a number of trees on various subsets of a dataset, whereas the basic decision tree classifier [48] 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 overfitting [117] to the training dataset.

The random forest classifier is a supervised technique, which requires labeled data. ADQuaTe labels the suspicious records or sequences as *invalid* and all the non-suspicious records or sequences as *valid* and uses this data to train the classifier. Among all the generated trees, the three ones with the lowest classification error are displayed via the web interface of the ADQuaTe tool.

The decision trees represent a set of if-then-else decision rules. These rules describe the constraints that identify records or sequences as valid or invalid based on their variable values. The random forest classifier uses decision tree algorithms, such as ID3 [118] and Classification And Regression Trees (CART) [119]. ID3 builds a multi-way tree (i.e., a tree with nodes that can have more than two edges) for categorical variables and CART builds a binary tree (i.e., a tree with nodes that have exactly two outgoing edges) for both numeric and categorical variables. ADQuaTe uses a random forest classifier in the H2O platform [120] (i.e., a fully open-source, distributed in-memory machine learning platform) based on the CART algorithm to construct the trees. A decision tree is built top-down from a root node to the leaf nodes and involves partitioning the data into subsets that contain records or sequences with similar labels (i.e., homogeneous subsets). The decision tree algorithms use an impurity criterion to calculate the homogeneity of a subset. If a subset is completely homogeneous (i.e., all the records or sequences are either valid or invalid), then the impurity is equal to zero. If the subset is equally divided (i.e., half of data records or sequences in the subset is valid and the other half is invalid), then the impurity is equal to one. Constructing a decision tree is based on finding the variables that result in the most homogeneous subsets. At each level of the tree, the algorithm chooses a variable that results in subsets with the lowest impurity and splits the dataset into subsets based on the values of that variable. The algo-

rithm 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. As we limited the maximum level of the decision tree, the splitting of the subsets may stop before reaching to homogeneous subsets. As a result, there may exist leaf nodes with labels between zero and one, which indicates the probability of being invalid for the records or sequences described by the path from the root node to that leaf node. For example, a label value equal to 0.6 indicates that in the resulting subset from the root node to that leaf node, 60% of the records or sequences are invalid and 40% of the records or sequences are valid. The probability for the records or sequences described by the values of variables in this path of being invalid is equal to 0.6.

3.2.5 Anomaly inspection

The suspicious groups or sequences of records may contain false alarms as a result of the unsupervised technique used by the constraint discovery component. Unsupervised techniques do not distinguish between valid and invalid data in the training set. These techniques have the potential to learn incorrect constraints from invalid data and generate false alarms.

ADQuaTe minimizes false alarms through an interactive learning process [15] that can incorporate domain expert feedback (when available) to improve the accuracy of the approach. ADQuaTe allows domain experts to inspect the suspicious groups or sequences. Domain experts use a web-based interface to flag as faulty those groups or sequences of records that are actually faulty. The expert feedback is used by ADQuaTe to label the training data records as described in Section 3.2.5.1. Sections 3.2.5.2 and 3.2.5.3 describe how the labels are used by the constraint discovery and anomaly detection components to improve the accuracy of the approach.

3.2.5.1 Update Training Dataset for Retraining

We add a label attribute with four possible values (1: faulty, 0.5: suspicious, 0: unknown, and -1: valid) to each record in the input dataset. The label is initially 0 for every record. The values of the label are updated based on the feedback. Records in a group or sequence 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 ADQuaTe remain 0. The updated dataset is used to retrain the machine learning model. Figure 3.2 shows the process of updating the training set for the retraining phase. The steps are as follows:

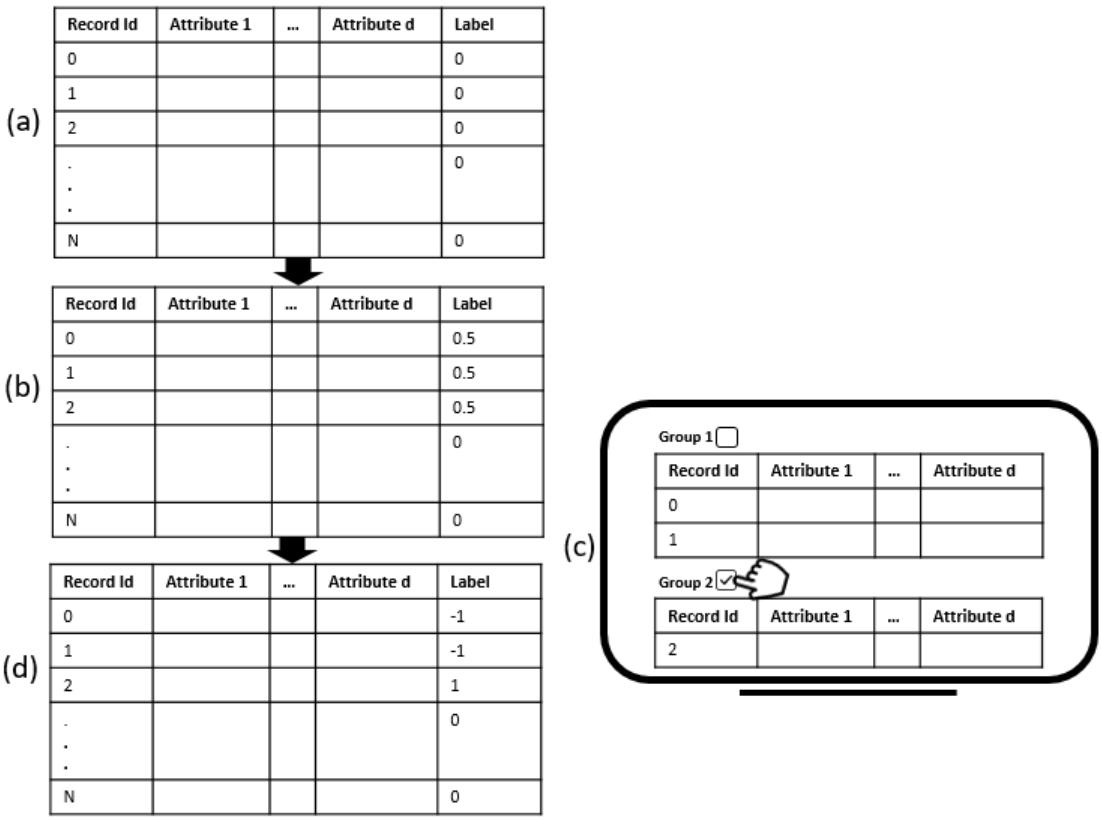


Figure 3.2: Process of Updating Training Dataset

(a) The table in Figure 3.2 (a) shows the initial dataset for the training phase. In this dataset, the labels are zero. The constraint discovery component uses this initial dataset to train the learning model.

(b) The table in Figure 3.2 (b) shows how the anomaly detection component modified the labels from 0 (unknown) to 0.5 (suspicious) based on the calculation of s-score by the anomaly detection component (Section 3.2.5.3).

(c) Figure 3.2 (c) shows the web-based interface of the ADQuaTe tool that reports two groups or sequences of suspicious records. In this example, the domain expert marked *Group 2* as faulty.

(d) The table in Figure 3.2 (d) shows how ADQuaTe updated the label values of the records in the dataset based on the expert feedback. The labels of the marked and unmarked records are updated to 1 and -1 respectively because the domain expert determines a group or sequence as valid by not flagging that group. All the other record labels (i.e., labels of records that are not reported by ADQuaTe as suspicious) remain 0.

3.2.5.2 Incorporate Expert Feedback into Constraint Discovery

We incorporate domain expert feedback into the constraint discovery component by (1) defining the reconstruction error of autoencoder-based network based on the label value and (2) initializing the network parameters for the retraining phase.

Define reconstruction error based on label value. In addition to the existing inputs, we also provide the label to the deep network. Unlike the other attributes, the labels are not preprocessed as their values are already appropriately scaled between -1 and 1. The network is trained not only to minimize the difference between the records and their reconstruction (the original reconstruction error), but also to minimize the difference between the record labels and the labels predicted by the network. Sections 4.1.2 and 5.1.2 describe how we define the reconstruction errors of the autoencoder and LSTM-Autoencoder based on the label values for the non-sequence and sequence data respectively.

Initialize autoencoder parameters for retraining. The original autoencoder-based network uses randomly initialized parameters, such as weight and bias values [97]. To improve the accuracy in each retraining phase, we initialize the network parameters with the values learned in the previous execution. The objective is to ensure that the network does not lose any information from the previous execution and the network is at least as accurate as the previous trained network.

3.2.5.3 Incorporate Expert Feedback into Anomaly Detection

We incorporate domain expert feedback into the anomaly detection component by (1) defining the *s*-score based on the label values and (2) tuning the threshold value.

Define *s*-score based on record labels. ADQuaTe calculates the *s*-score based the reconstruction error and the labels obtained using domain expert feedback. The objective is to ensure that ADQuaTe will not report as suspicious any valid record (i.e., records that ADQuaTe flagged as suspicious in previous executions but not flagged by the expert) in subsequent executions. Moreover, all the records marked as faulty by the domain expert in previous executions will be reported as confirmed faulty in subsequent executions. These records are reported in a separate group from the suspicious records and have higher *s*-score values.

Tune threshold. The threshold value (T) is tuned to reduce the false alarms over time. At the beginning of the training phase, T is equal to the mean of *s*-scores of all the input records. T is updated for the retraining phase.

Given the *s*-scores of unmarked (valid) records in the range [*a*,*b*] and the *s*-scores of marked (faulty) records in the range [*c*,*d*], as $b < c$, there is no overlap between the *s*-scores of faulty and valid records. Equation 3.2 describes how ADQuaTe tunes T for the retraining phase to make sure that no valid records are displayed as suspicious in the next iteration.

$$T = \begin{cases} min(c,\ P) & if\ P > 0, \\ min(c,\ p(s\text{-}scores, 90)) & otherwise \end{cases} \tag{3.2}$$

where P is the percentage of previously known anomalies for the input dataset and p is the percentile function. This function returns a value below which a given percentage of records in the dataset falls. Based on this equation, if there is a set of previously known anomalies in the dataset, ADQuaTe detects at least $P\%$ of records as suspicious to ensure that all the previously detected anomalies are reported in the current iteration. We set the threshold at 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 [121] from the UCI repository is equal to 10%. Domain experts can change this value based on the knowledge of the validity of their datasets.

3.3 Tune ADQuaTe Hyper-Parameters

ADQuaTe uses a grid search approach [122] to tune the parameters of the machine learning models. Grid search is a brute force approach that scans the data to configure optimal hyper-parameters for a given model. Hyper-parameters are parameters that are not automatically learned within the learning process. The grid search technique builds a model for every combination of various hyper-parameters and selects the model that gives the highest accuracy. The grid search technique outperforms the random and optimization [123] techniques because it considers all parameter combinations in the model selection. However, this consideration can be problematic when the number of hyper-parameters is large. In our approach the number of hyper-parameters is small. They are the number of hidden layers, the number of neurons, and the number of epochs [13]. Thus, the number of parameter combinations is not a problem.

Grid search approaches used for autoencoder-based models select a model that minimizes the reconstruction error [124]. However, this model may overfit on the training data and learn incorrect constraints pertaining to the invalid data. In Section 4.2, we demonstrate that a model with the lowest error is not necessarily the most effective model and has the potential to generate false alarms. We propose to use ground truth knowledge (i.e., a set of known anomalies) to select a model that maximizes the true positive rate as the percentage of the previously known anomalies in the data. We use a wide range of values for the autoencoder hyper-parameters to form 300

different models. In Section 4.2, we demonstrate that the selection of the best model is application specific.

We also propose to use an early stopping [16] technique instead of training the network until the error reaches its minimum value [97]. This technique sets the number of epochs for controlling overfitting in machine learning models. By increasing the number of epochs (i.e., by over-training the network), the reconstruction error decreases [97]. However, the network starts learning the incorrect constraints from the anomalous data. As a result, the training process must be stopped before the network is overfitted on the training data. On the other hand, stopping too early may result in under-fitting [125] of the network, in which the model has not completely learned the complex constraints from the data and as a result, is not capable of detecting complex constraint violations. We propose to stop the training process at the point when the true positive rate starts dropping after a certain number of epochs. In Section 4.2, we show how to use the true positive rate to find the best stopping point and demonstrate that this stopping point is application specific.

3.4 ADQuaTe Tool

We implemented the components of ADQuaTe in an open-source web-based tool [21]. We started the implementation of ADQuaTe with a concrete web application that only supported constraint discovery and anomaly detection in non-sequence data. Then we created a new version for sequence data. The components of the two versions have some functionalities in common and others unique to each version. We designed the ADQuaTe framework by grouping the common functionalities and adding the ability to extend or modify the components.

In this section, we describe the design of the ADQuaTe tool. Figure 3.3 shows the logical view of the tool architecture that contains *UI*, *Domain*, and *Library* layers. There are two web pages in the *UI* layer, namely, *Import* and *Validate*. As shown in the *Library* layer, we used H2O [126], TensorFlow [127], and Scikit-learn [128] open source libraries for the implementation of the machine learning algorithms used in the *ConstraintDiscovery*, *AnomalyDetection*, and *AnomalyInterpretation* classes of the *Domain* layer. We used Pandas [129] library for connecting

to the database server by the *AnomalyInspection* and *DataPreparation* classes. We used Statistics [130], Graphviz [131], and CRAN [82] libraries to generate the visualization plots for the *AnomalyInterpretation* class.

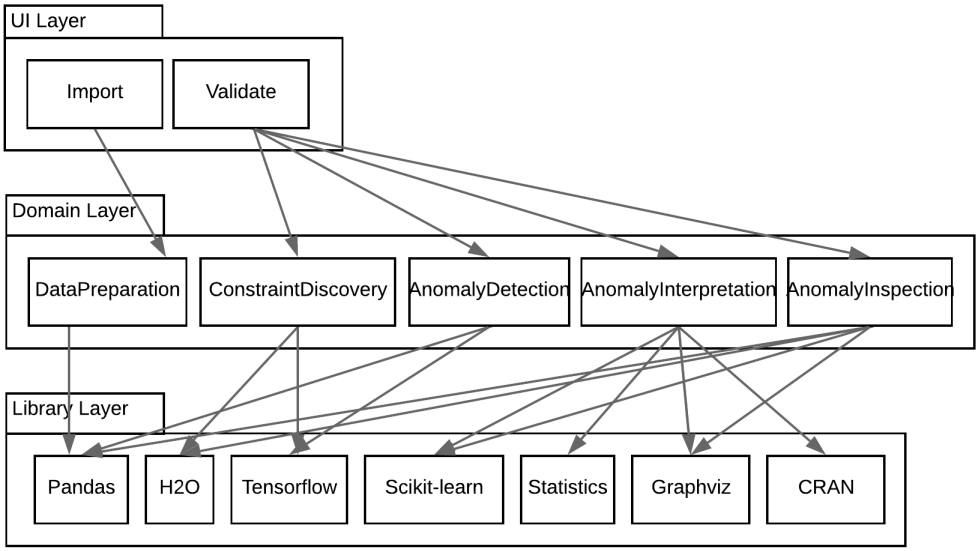


Figure 3.3: Logical View of ADQuaTe Tool Architecture

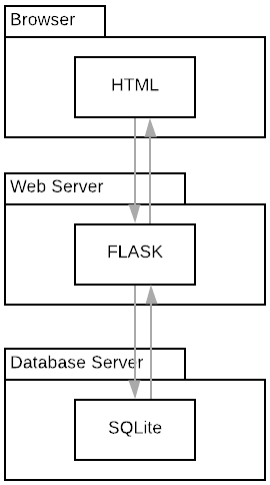


Figure 3.4: Deployment View of ADQuaTe Tool Architecture

Figure 3.4 shows the deployment view of the tool architecture. We used HTML to implement the user interface. A Python-based web framework called Flask was used to develop the web server

on (1) an Ubuntu virtual machine (2 vCPUs, 3.75 GB memory) on the Google Cloud Platform for the health datasets, (2) a Fedora physical machine (4 GHz CPU, 32 GB memory) on the Department of Computer Science at CSU for the UCI datasets, and (3) an Ubuntu virtual machine (2.20 GHz CPU, 8 GB memory) on an Energy Institute server at CSU for the energy datasets. We implemented the tool using Python. We used SQLite as the database dialect to store the intermediate data for the tool.

Figure 3.5 shows the class diagram for the domain layer of the ADQuaTe tool. *ConstraintDiscovery* and *AnomalyInterpretation* are interface classes and *AnomalyDetection* and *AnomalyInspection* are abstract classes implemented by child classes, which use different machine learning algorithms to implement their parent methods.

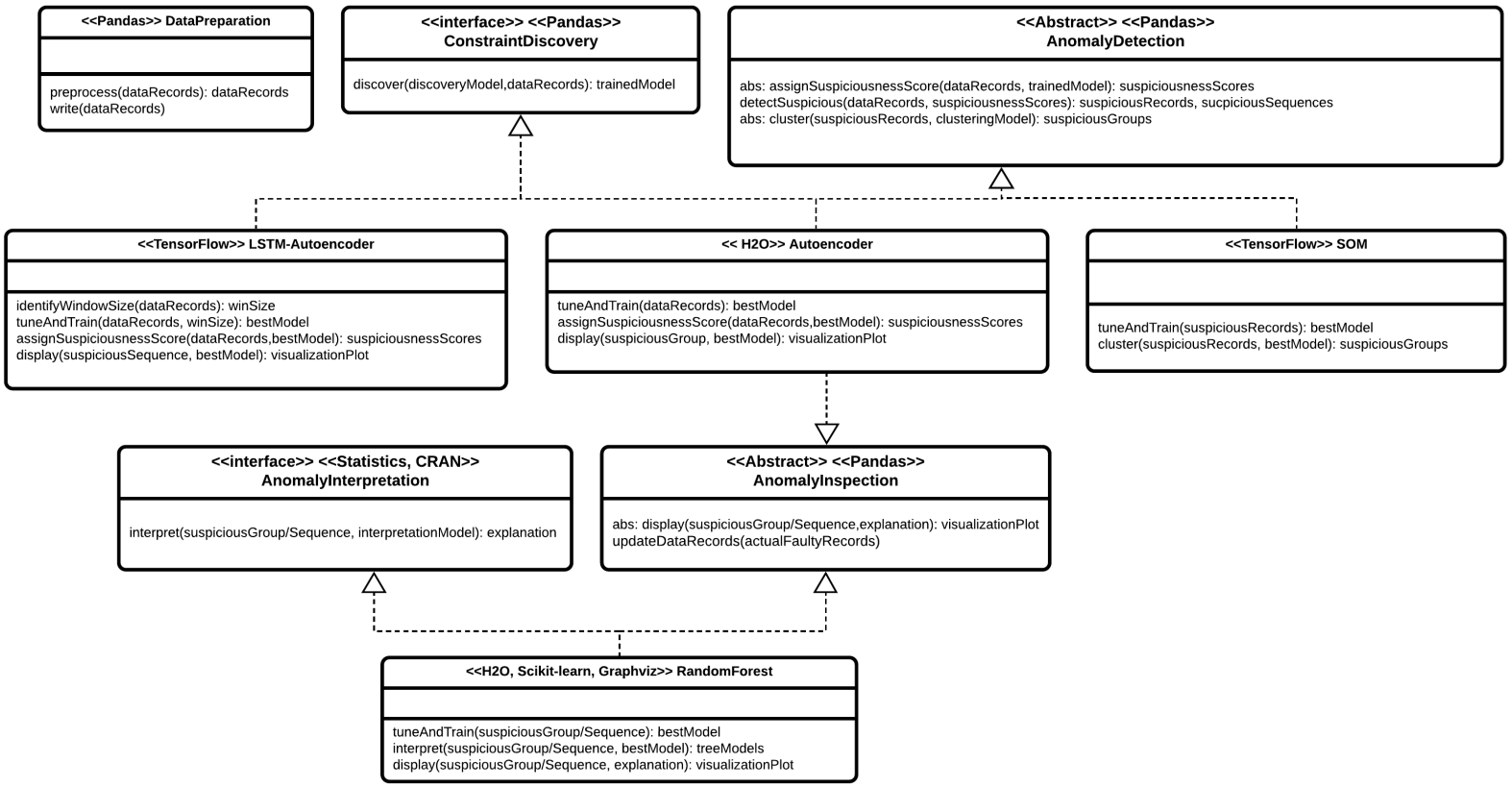


Figure 3.5: Class Diagram for Domain Layer of ADQuaTe

Table 3.8 describes the methods of these classes. The sequence diagrams in Figures 3.6 and 3.7 show how a domain expert interacts with the tool during the entire process. These diagrams depict the interaction between the objects of classes through method calls in the order in which these interactions take place.

As shown in the sequence diagram in Figure 3.6, the domain expert imports the data for analysis and selects machine learning models for constraint discovery, clustering, and anomaly interpretation. The data records are preprocessed and stored in an intermediate data storage. The optional (opt) fragment in the diagram checks whether or not the expert has imported a previously trained model. The *tuneAndTrain* method is called only if the model is not previously trained. This feature considerably reduces the testing time by allowing the domain expert to store and restore the trained model for future use. The output of the *tuneAndTrain* method is a trained model that is passed to the *AnomalyDetection* object for detecting the suspicious records or sequences and grouping the suspicious records. The suspicious groups or sequences of records are passed to the next object for interpretation. The anomaly interpretation object generates visualization plots.

The sequence diagram of Figure 3.7 shows that the expert inspects the suspicious groups or sequences and marks the actual faults. The *AnomalyInspection* object updates the data records in the intermediate storage by modifying the labels of anomalous records from valid to invalid. The second sequence diagram is a loop describing the feedback loop that lets the expert repeat the constraint discovery and anomaly detection processes with the updated data set.

Table 3.8: Class Methods

Method	Class	Description
preprocess	*DataPreparation*	Normalizes the numeric attributes and one-hot encodes the categorical attributes.
write	*DataPreparation*	Imports the data records into the intermediate data storage.
tuneAndTrain	*Autoencoder*:: *ConstraintDiscovery*	Trains the best autoencoder model selected by the grid search technique against the data records.
assignSuspiciousnessScore	*Autoencoder*:: *AnomalyDetection*	Assigns the autoencoder reconstruction error to each data record.
IdentifyWindowSize	*LSTM-Autoencoder*:: *AnomalyInspection*	Identifies the input size for LSTM-Autoencoder based on the autocorrelation of the input records.
tuneAndTrain	*LSTM-Autoencoder*:: *ConstraintDiscovery*	Trains the best LSTM-Autoencoder model selected by the grid search technique against the data records.
assignSuspiciousnessScore	*LSTM-Autoencoder*:: *AnomalyDetection*	Assigns suspiciousness score to each data record, attribute, and subsequence.
detectSuspicious	*AnomalyDetection*	Flags as suspicious those records/sequences whose suspiciousnessScores are greater than a threshold.
cluster	*SOM*:: *AnomalyDetection*	Uses the best clustering model selected by the grid search technique to group the suspicious records based on their similarity.
interpret	*RandomForest*:: *AnomalyInterpretation*	Generates a decision tree model from the suspicious group/sequence.
display	*Autoencoder*:: *AnomalyInspection*	Displays the suspiciousnessScore per attribute plot using the best autoencoder model.
display	*LSTM-Autoencoder*:: *AnomalyInspection*	Displays the suspiciousnessScore per attribute plot using the best LSTM-Autoencoder model.
display	*RandomForest*:: *AnomalyInspection*	Displays the best decision tree using the decision tree model.
updateDataRecords	*AnomalyInspection*	Updates the data record labels based on the actual anomalous records.

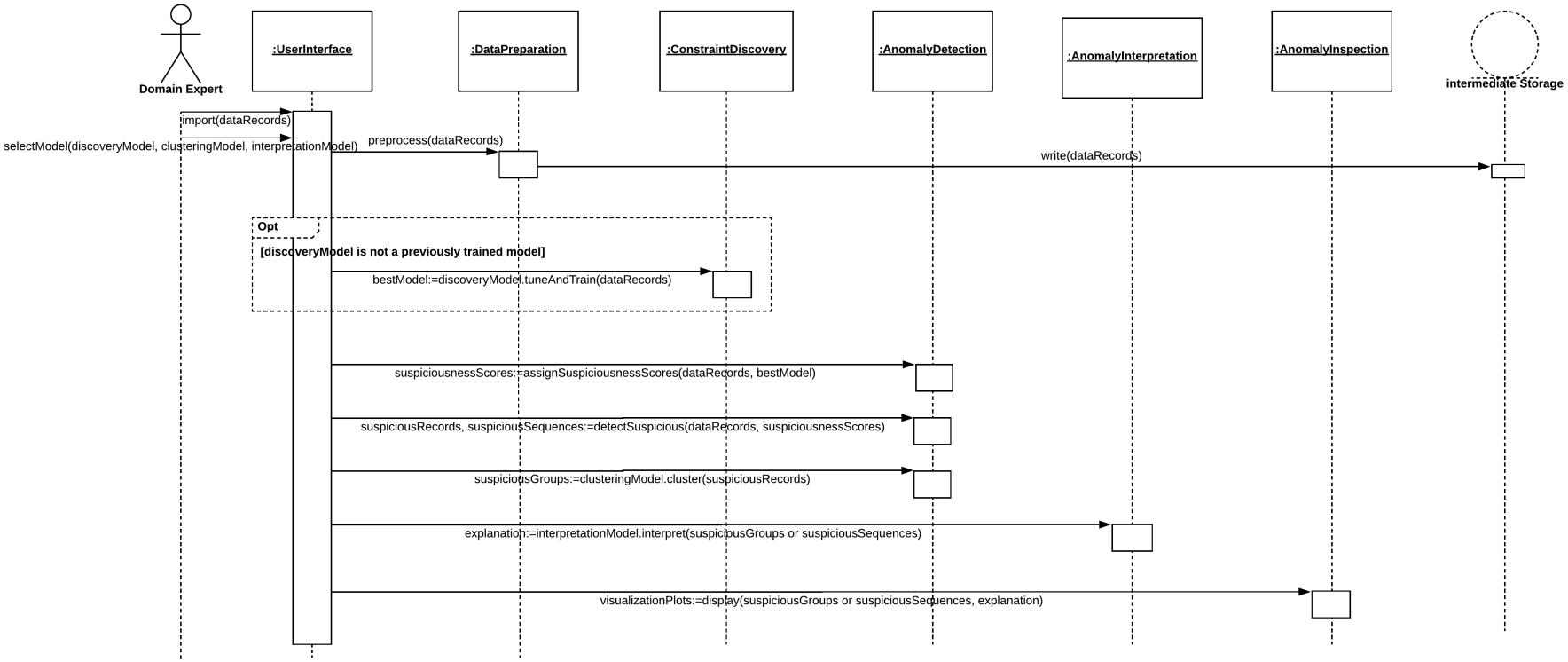


Figure 3.6: Sequence Diagram for Domain Expert Interaction Involving Data Importation

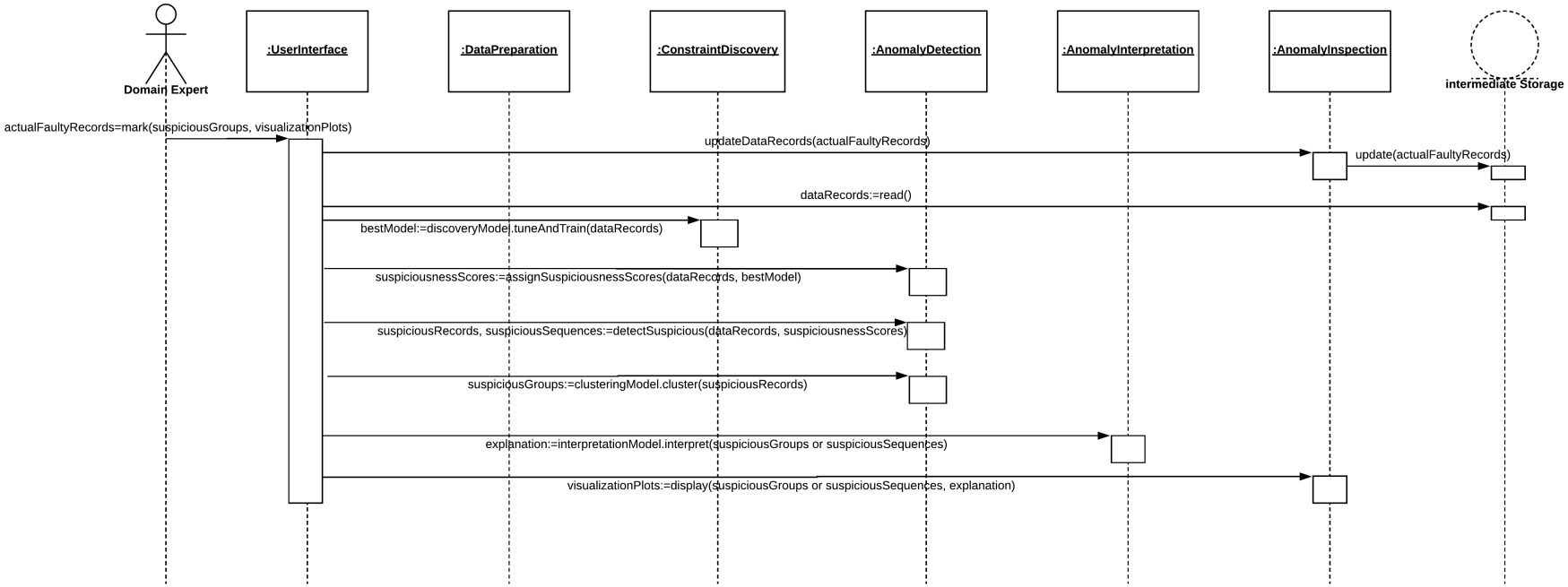


Figure 3.7: Sequence Diagram for Domain Expert Interaction Involving Anomaly Inspection

ADQuaTe has the following characteristics.

Easy to use. ADQuaTe tool implements a user-friendly interface that allows the domain experts who are not necessarily skilled in programming to use the tool to validate their data. Moreover, the process of getting feedback from the domain expert simply involves clicking on the correctly detected groups.

Easy to understand. To reduce the inspection time for domain experts, the ADQuaTe tool identifies groups of suspicious records and adds explanations to each group to make the tool output understandable to the domain experts.

Fully automated. ADQuaTe tool supports fully automated data preparation, constraint discovery, anomaly detection, and anomaly interpretations. The data attributes are automatically preprocessed in the tool and all the machine learning parameters are automatically tuned.

Domain independent. ADQuaTe tool uses unsupervised learning that does not require prior knowledge about the data in its first execution. As a result, this tool is applicable to input datasets from any application domain. However, ADQuaTe uses domain knowledge to improve its results. Moreover, the threshold tuning of ADQuaTe is domain dependent.

3.5 Test-Bed

We have designed and implemented a test-bed [21] that contains different scripts for automatically evaluating the effectiveness and efficiency of ADQuaTe. These scripts are for (1) interactive execution and evaluation of the tool, (2) evaluating the parameter tuning, (3) mutation analysis, (4) comparing windowing approaches, and (5) comparing anomaly detection approaches.

Interactive execution and evaluation of the tool. The objective of this script is to automatically evaluate the interactive process of ADQuaTe. This script automatically executes ADQuaTe against an input dataset and detects suspicious groups/sequences. The script updates the label values from 0.5 to 1 for the suspicious records that are actually faulty (i.e., within the previously known faults)

and from 0.5 to -1 for the ones that are valid. Next, the script retrains the constraint discovery model and reruns the anomaly detection component (Section 3.2.5) and measures the accuracy and performance of ADQuaTe based on the metrics we describe in Sections 4.2 and 5.2. The whole process is performed $t \geq 1$ times. If $t = 1$, the script evaluates the tool when expert feedback is not used.

Evaluating parameter tuning. The objective of this script is to evaluate the parameter tuning effectiveness of ADQuaTe for non-sequence data. The script calculates the accuracy of ADQuaTe for 300 different models (i.e., 300 combination of hyperparameters) based on the metrics we define in Section 4.2 to demonstrate that the original grid search approaches do not select the most effective model. The script also calculates the accuracy of ADQuaTe for different number of epochs (i.e., complete passes through the training data) to demonstrate that we need to stop training based on the true positive rate to avoid overfitting on training data.

Mutation analysis. The objective of this script is to automatically inject different types of anomalies into a sequence dataset and evaluate ADQuaTe to demonstrate that the tool can effectively and efficiently detect these anomalies. Section 5.2.1 describes how the script mutates an input dataset based on different mutation operators that we define for the sequence data.

Comparing windowing approaches. The objective of this script is to compare our autocorrelation-based windowing with a brute force windowing approach. This script takes the mutated sequence 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 accuracy for the tool. The script also runs the tool using our autocorrelation-based windowing. Finally, the script compares the accuracy and performance of the two approaches using metrics we define in Section 5.2.

Comparing anomaly detection approaches. The objective of this script is to compare ADQuaTe with existing anomaly detection approaches for sequence data. This script takes a dataset with a set of known anomalies as input and executes the tool once against the dataset. The scripts calculates

an accuracy measure called $F1$ score to compare the anomaly detection effectiveness of ADQuaTe with existing approaches using the same score reported in two review papers [3, 4].

Chapter 4

ADQuaTe2: An Instantiation of ADQuaTe for Non-Sequence Data

In this chapter, we provide a detailed description of ADQuaTe2 [17, 18], which instantiates ADQuaTe components for non-sequence data. Section 4.1 presents different components of ADQuaTe2. Section 4.2 evaluates ADQuaTe2 and Section 4.3 summarizes the chapter.

4.1 Instantiated Components

In this section, we describe how ADQuaTe2 instantiates the data preparation, constraint discovery, anomaly detection, anomaly interpretation, and anomaly inspection components of ADQuaTe for non-sequence datasets.

4.1.1 Data Preparation

The data preparation component is not customized since ADQuaTe2 uses an approach that is common to both non-sequence and sequence data for this component. This approach is described in the framework components (Section 3.2.1).

4.1.2 Constraint Discovery

ADQuaTe2 instantiates this component using an autoencoder [13] that is known to be effective for attribute representation learning [132] as a result of using a deep learning architecture, which enables learning complex associations among attributes using several layers of non-linearity.

An input to the autoencoder is a record X with d attributes. This input is mapped into a hidden representation Z with d' attributes, which are non-linear combinations of the input attributes. The decoder decompresses Z into a new representation X' with d attributes that closely matches the original data. X' is called a reconstruction of X. The network is trained to minimize the re-

construction error (Equation 4.1), which is the average squared distance between the original data record and its reconstruction [13].

$$RE = \frac{1}{n}\sum_{i=0}^{n-1}(X'_i - X_i)^2 = \frac{1}{n}\sum_{i=0}^{n-1}\sum_{j=0}^{d-1}(a_i'^j - a_i^j)^2 \tag{4.1}$$

where *n* is the total number of records, *d* is the number of attributes, $X_i = (a_i^0, ..., a_i^{d-1})$ is the i^{th} network input, and $X'_i = (a_i'^0, ..., a_i'^{d-1})$ is the i^{th} network output.

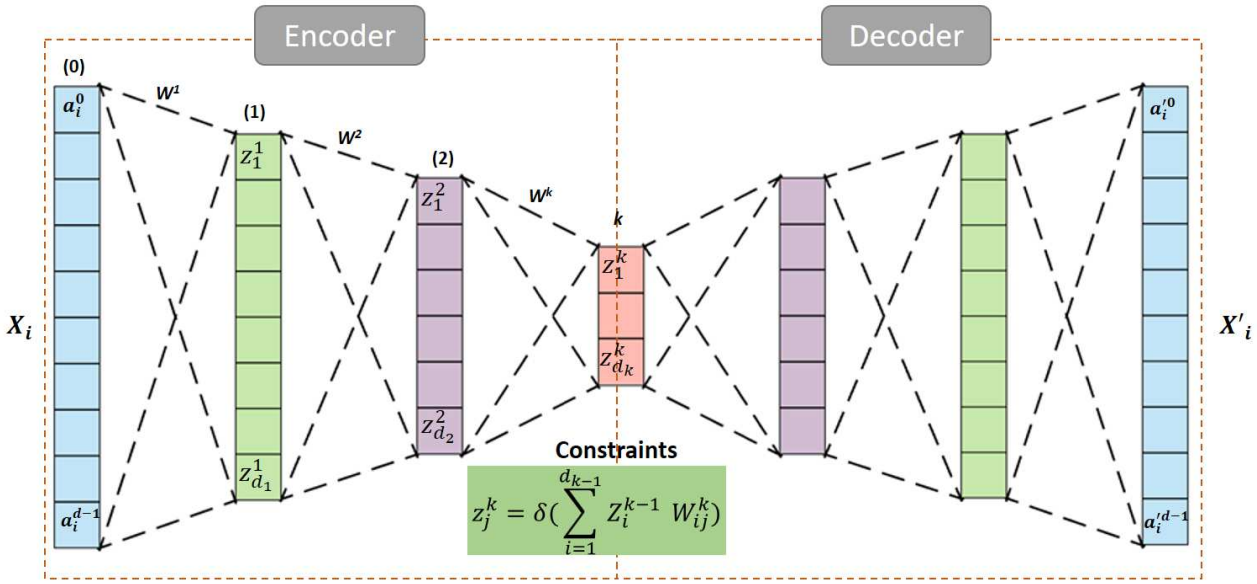


Figure 4.1: Constraints Discovered by Autoencoder

The green box in Figure 4.1 shows the format of the constraints as the outputs of the encoder. The input is a record X_i with d attributes. Each layer generates a new representation as a non-linear weighted combination of the attributes in its previous layer. The last layer of the encoder is a vector of d_k constraints (z_j^k), each of which is a non-linear weighted sum (δ is a non-linear activation function) of the attributes in the previous layers. Multiple hidden layers enable the composition of attributes from lower layers, allowing the possibility for modeling more complex constraints than a similar shallow network [133].

In addition to the existing inputs (i.e., the attributes $a_i^0 \ldots a_i^{d-1}$ for record X_i), ADQuaTe2 also provides the new label (l_i) to the autoencoder. Unlike the other attributes, the labels are not

preprocessed as their values are already appropriately scaled between -1 and 1. Figure 4.2 shows the new attribute and its corresponding output in the autoencoder structure.

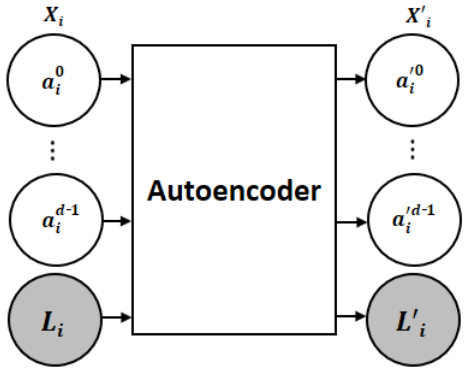


Figure 4.2: Interactive Autoencoder

The autoencoder is trained not only to minimize the difference between the record and its reconstruction (the original reconstruction error), but also to minimize the difference between the record label and the label predicted by the network.

$$RE = \frac{1}{n}\sum_{i=0}^{n-1}((l'_i - l_i)^2 + \sum_{j=0}^{d-1}(a_i'^j - a_i^j)^2) \tag{4.2}$$

where l_i is the label of i^{th} record and l'_i is the label predicted by the network for this input, and N is the number of records.

4.1.3 Anomaly Detection

For each record, ADQuaTe2 calculates the *s*-score based on the autoencoder reconstruction error and the labels obtained using domain expert feedback. Records whose *s*-score is greater than a threshold are flagged as suspicious.

$$s\_score(X) = RE(X) + l(X) \tag{4.3}$$

where $RE(X)$ is the reconstruction error of record X (Equation 4.4) and $l(X)$ is the label assigned to record X after interacting with the domain expert. Since $RE(X)$ is normalized into

the range [0,1] and $l(X) \in \{-1, 0, 0.5, 1\}$, this new definition ensures that the *s*-scores of faulty, valid, and unknown records are in the range [1, 2], [-1, 0], and [0, 1] respectively. Setting the threshold to a value greater than zero ensures that ADQuaTe2 never reports as suspicious any valid record in subsequent executions. Moreover, all the records marked as faulty by the expert in previous executions are reported as suspicious in subsequent executions, but with a higher *s*-score value (in the range [1,2]).

$$RE(X) = Normalized((X'_i - X_i)^2) = Normalized(\sum_{j=0}^{d-1}(a_i^{'j} - a_i^j)^2) \tag{4.4}$$

where *d* is the number of attributes, $X_i = (a_i^0, ..., a_i^{d-1})$ is the i^{th} network input, and $X'_i = (a_i^{'0}, ..., a_i^{'d-1})$ is the i^{th} network output.

It can be too time-consuming for a domain expert to inspect a large number of suspicious records. Thus, ADQuaTe2 instantiates this component by grouping the suspicious records based on their similarity to make the anomaly validation and interpretation feasible. We used a clustering approach called Self Organizing Map (SOM) [106], which preserves the relationships among data attributes in its clusters [134]. Distance-based clustering algorithms, such as *k*-means [64] and *k*-prototypes [135] can also be used for grouping but they do not preserve these relationships [67]. As all data attributes are involved in computing the distance in these clustering techniques, insignificant attributes (i.e., attributes that can be removed without having any impact on the constraint discovery and anomaly detection results) contribute to the clustering result. Therefore, the techniques occlude the relationships of significant attributes in their clusters. We also experienced this issue; the records in a group were similar only with respect to their single attribute values and not based on the relationships among the attribute values.

SOM is a type of unsupervised neural network that produces a low-dimensional representation of the input space. This method converts the complex, nonlinear relationships between data attributes into simple geometric relationships on a low-dimensional map. The dimensionality reduction phase of SOM decreases the impact of insignificant attributes and, as a result, leads to a better clustering based on significant attributes.

The outputs of this component are groups of suspicious records. Tables 4.1 and 4.2 show two groups of suspicious records in the *Drug_exposure* table. Tables 4.3 and 4.4 show two groups of suspicious records in the *Plant_diagnosis* table. Each group contains multiple records and their attribute values.

Table 4.1: Group 1 of Suspicious Records Detected From the *Drug_exposure* Table

ID	Name	Unit	Route	Days Supply	Dose	Quantity	Refills	*s*-score
0	Polyethylene Glycol 3350 17000 MG	Standard acceleration of free fall	Oral	850	66	850	0	0.35
1	Carbamazepine 100 MG	Tablet	Topical	360	200	360	3	0.4

Table 4.2: Group 2 of Suspicious Records Detected from the *Drug_exposure* Table

ID	Name	Unit	Route	Days Supply	Dose	Quantity	Refills	*s*-score
2	Arginine HCL 100 MG/ML	Milliliter	Gastrostomy	50000	191	50000	6	0.7

Table 4.3: Group 1 of Suspicious Records Detected From the *Plant_diagnosis* Table

ID	Host	Diagnosis ID	Genus Confirmation	Diagnosis Needed	Suspected Problem	Sample Category	*s*-score
0	Tomato	Chemical injury	Suspected	Disease ID	None	Vegetables	0.15
1	Silver Maple	Heat stress	Undetermined	Disease ID	None	Woody	0.1

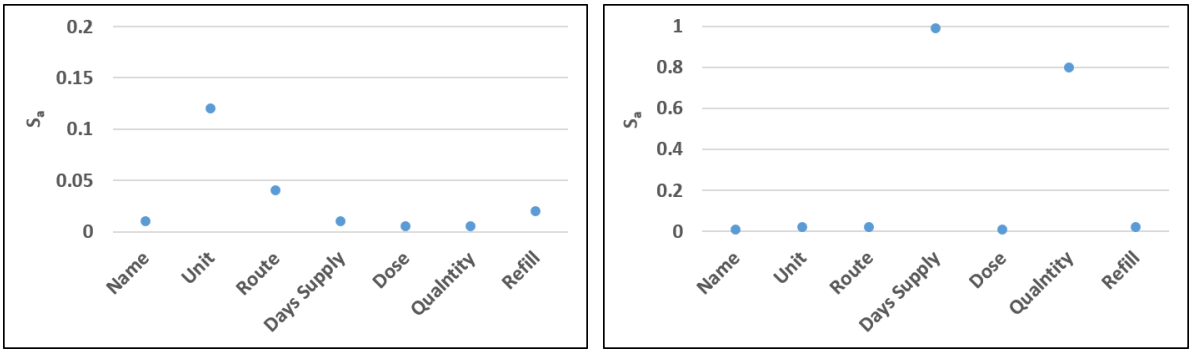
Table 4.4: Group 2 of Suspicious Records Detected From the *Plant_diagnosis* Table

ID	Host	Diagnosis ID	Genus Confirmation	Diagnosis Needed	Suspected Problem	Sample Category	*s*-score
2	Tomato	Fire blight	Not Detected	Disease ID	Fireblight	Vegetables	0.2

4.1.4 Anomaly Interpretation

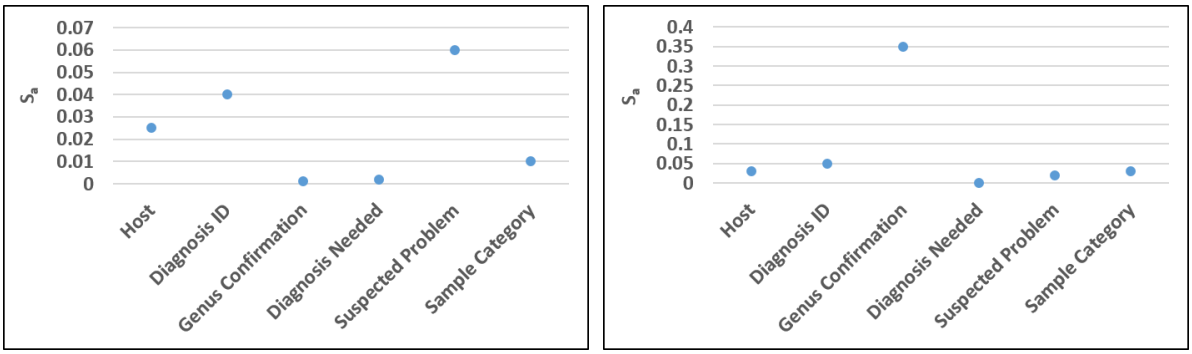
ADQuaTe2 instantiates this component through displaying *s*-score per record in a suspicious group and visualization plots of two types for that group: (1) *s*-score_per_attribute and (2) decision trees.

***s*-score_per_record.** The *s*-score column in Tables 4.1, 4.2, 4.3, and 4.4 shows the value of *s*-score per record (Equation 4.4) for the four suspicious groups in the *Drug_exposure* and *Plant_diagnosis* tables.



(a) Group 1 **(b)** Group 2

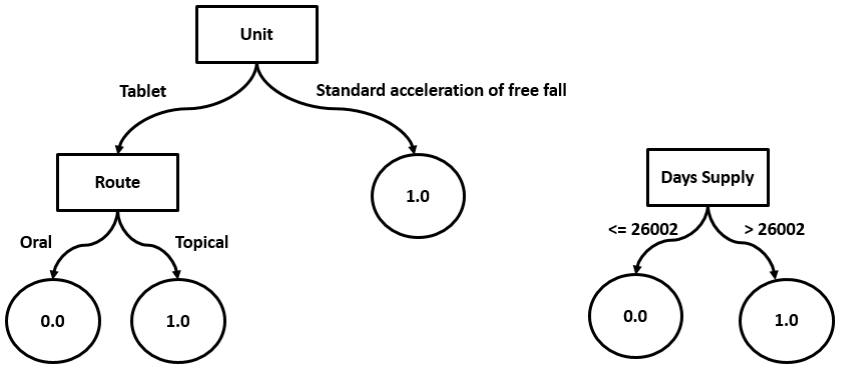
Figure 4.3: *s*-score Per Attribute for *Drug_exposure* Table



(a) Group 1 **(b)** Group 2

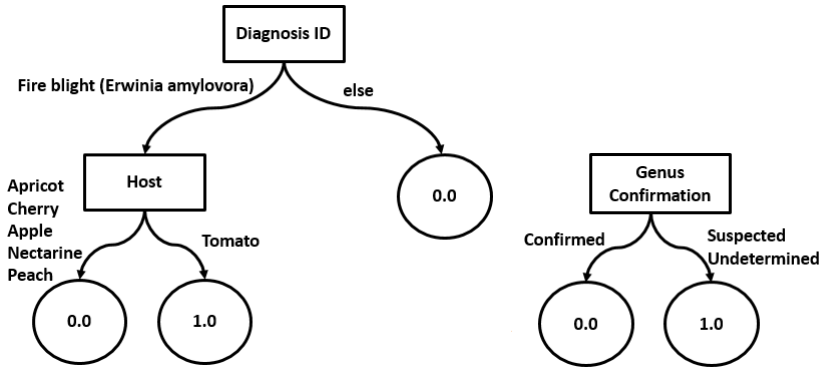
Figure 4.4: *s*-score Per Attribute for *Plant_diagnosis* Table

***s*-scores_per_attribute plot.** For each attribute a in a suspicious group, this instantiation calculates its *s*-score, S_a, where S_a is defined as the average of the *s*-scores of the same attribute, a, for all the records in that group. Figure 4.4 shows these plots for the *Plant_diagnosis* table. Based on these plots, the attributes *Host*, *Diagnosis ID*, and *Suspected Problem* are the major causes of invalidity in Group 1, and the attribute *Genus Confirmation* is the major cause of invalidity in Group 2 of the suspicious records.



(a) Group 1 **(b)** Group 2

Figure 4.5: Decision Trees for *Drug_exposure* Table



(a) Group 1 **(b)** Group 2

Figure 4.6: Decision Trees for *Plant_diagnosis* Table

Decision tree. For each suspicious group, this instantiation process labels the suspicious records in that group as *invalid* and all the non-suspicious records as *valid* and uses this data to train the classifier. Figures 4.5a and 4.5b show the trees generated for the two groups shown in Tables 4.1 and 4.2 for the *Drug_exposure* table. In the decision tree of Figure 4.5a, the *Unit* attribute is selected by the *ID3* algorithm [118] as the root node of the tree. Based on the values of this attribute, two subsets of records are created in the two branches of the tree. The right subset has an entropy equal to 0 as all the records in that subset are invalid (i.e., the label of the records in this path from root to the leaf node is equal to 1.0). The left subset has an entropy value greater than zero and requires more splitting of records based on another attribute. The algorithm selected the *Route* attribute as the decision node for this branch, which results in two subsets, both of which

have entropy equal to zero. The right subset contains records with all invalid values (i.e., records labeled as 1.0) and the left subset include records with all valid values (i.e., records labeled as 0.0). Each branch of the tree from the root to the leaf nodes indicates a specific constraint to be verified.

These constraints are in form of "**If** $Pred$ **then record is** <valid|invalid>" rules, which determine whether a record is valid or invalid based on its attribute values. $Pred$ is in form of "(a_1 ROP_1 v_1) LOP_1 ... LOP_{d-1} (a_d ROP_d v_d)", where a_i is an attribute, v_i is the value of the attribute, ROP_i is a relational operator ($=$, $\geq$, $>$, $<$, $\leq$), and LOP_i is a logical operator (and and or).

The trees in Figures 4.5 are used to generate the following constraints. In Figure 4.5a the path from the root (*Unit*) to the leaf node labeled 0.0 represents the first constraint in the following list, which states that if the drug *Unit* is 'Tablet' and the *Route* is 'Oral' , then the record is valid.

- If *Unit*='Tablet' and *Route*='Topical', then record is invalid.
- If *Unit*='Tablet' and *Route*='Oral', then record is valid.
- If *Unit*='Standard acceleration of free fall', then record is invalid.
- If *Days Supply* $>$ 26002, then record is invalid.
- If *Days Supply* $<=$ 26002, then record is valid.

The trees in Figures 4.6 are used to generate the following constraints. In Figure 4.6a, the path from the root (*Diagnosis ID*) to the leaf node labeled 1.0 represents the first constraint in the following list, which states that if the plant *Diagnosis ID* is 'Fire blight (Erwinia amylovora)' and the *Host* is 'Tomato', then the record is invalid.

- If *Diagnosis ID*='Fire blight'and *Host*='Tomato', then record is invalid.
- If *Diagnosis ID*='Fire blight'and *Host* in ('Apricot', 'Cherry', 'Apple', 'Nectarine', 'Peach'), then record is valid.
- If *Genus Confirmation* in ('Suspected', 'Undetermined'), then record is invalid.
- If *Genus Confirmation*='Confirmed', then record is valid.

4.1.5 Anomaly Inspection

The anomaly inspection component is not customized since ADQuaTe2 uses an approach that is common between non-sequence and sequence data for this component. This approach is described in the framework components (Section 3.2.5).

4.2 Evaluation

Table 4.5: Datasets from Real-world Health and Plant Domains and UCI ML Repository [2]

ID	Name	Domain	#Records	#Attributes	Known Anomalies (%)
1	*Plant_diagnosis*	Plant	313	18	0.00
2	*Measurement* JOIN *Person*	Health	94,165	4	0.02
3	*Drug_exposure* JOIN *Concept*	Health	100,000	20	5.65
4	*Measurement* JOIN *Concept*	Health	100,000	19	4.81
5	*Visit_occurrence* JOIN *Concept*	Health	100,000	9	0.00
6	*Drug_exposure* JOIN *Observation_period* JOIN *Concept*	Health	600,000	7	18.33
7	*Procedure_occurrence* JOIN *Observation_period* JOIN *Concept*	Health	600,000	5	41.00
8	*Observation* JOIN *Observation_period* JOIN *Concept*	Health	1,000,000	7	0.07
9	*Wine*	Liquor	129	13	7.70
10	*Lymphography*	Oncology	148	18	4.05
11	*Glass_identification*	Criminology	214	10	4.20
12	*Vertebral_column*	Biomedical	240	6	12.50
13	*Heart_disease*	Health	267	75	20.60
14	*Ecoli*	Biology	336	8	2.67
15	*Ionosphere*	Radar	351	34	35.89
16	*Breast_cancer*	Oncology	699	10	34.50
17	*Satimage*	Satellite	5803	36	1.20
18	*Satellite*	Satellite	6435	36	32
19	*Shuttle*	Aerospace	49097	9	7.00

We evaluated the constraint discovery, anomaly detection, and anomaly interpretation effectiveness of ADQuaTe2 using real-world data from health and plant domains. We used seven datasets created using multiple table joins in the health data warehouse and one dataset from the plant diag-

nosis database. Rows 1–8 in Table 4.5 show the characteristics of these datasets. We also evaluated the improvements in the accuracy of the approach using datasets with ground truth data from the UCI repository [2]. Rows 9–19 show the characteristics of these datasets. We also demonstrated the parameter tuning effectiveness of ADQuaTe2 using the UCI datasets. Finally, we evaluated the time it takes for the overall approach to execute.

4.2.1 Evaluation Goals

4.2.1.1 Goal 1: Evaluate the effectiveness of the constraint discovery and anomaly detection of ADQuaTe2 using real-world health and plant datasets without incorporating expert feedback.

The success of the approach was measured along two dimensions: the anomaly detection ability and the identification of new constraints that were not previously specified by the experts. The new constraints discovered by the approach should help us detect new anomalies in the data. For this evaluation, we used domain expert feedback to validate the results. However, we did not incorporate this feedback to improve the approach. We evaluated these two components by answering the following questions.

RQ1.a: Can ADQuaTe2 detect the anomalies in real-world health and plant data that were already detected by existing tools that rely on manual specification of constraints by domain experts?

Given E, the set of anomalous records detected by an existing data quality test approach, and A, the set of suspicious records detected by ADQuaTe2, we define the metrics *Previously Detected* (PD), *Suspicious Detected* (SD), and *UnDetected* (UD).

PD: Percentage of anomalous records detected by an existing approach that could also be detected by ADQuaTe2.

$$PD = \frac{|E \cap A|}{|E|} \tag{4.5}$$

SD: Percentage of suspicious records detected by ADQuaTe2 that were not previously detected.

$$SD = \frac{|A - E|}{|A|} \tag{4.6}$$

UD: Percentage of anomalous records detected by an existing approach that could not be detected by ADQuaTe2.

$$UD = \frac{|E - A|}{|E|} \tag{4.7}$$

Table 4.6: Known Anomalies and Suspicious Records in Real-world Health and Plant Datasets Detected by ADQuaTe2

Dataset ID	|E|	|A|	PD	SD	UD
1	0	89	0.00	100.00	0.00
2	19	19	100.00	0.00	0.00
3	5650	6848	98.25	17.50	0.017
4	4810	5070	96.77	8.50	0.032
5	0	5026	0.00	100.00	0.00
6	109980	132174	99.99	16.75	0.01
7	246000	249724	97.21	4.24	2.79
8	700	753	96.14	10.63	3.86

Table 5.4 shows the values of *PD*, *SD*, and *UD* of ADQuaTe2 with respect to the known anomalies in the real-world health and plant datasets. In this table, $|E|$ is the number of known anomalies and $|A|$ is the total number of suspicious records detected by ADQuaTe2. There were no previously known anomalies for dataset *ID*s 1 and 5 and every record reported for these two sets corresponded to a suspicious record not previously detected. ADQuaTe2 detected between 96.14% and 100% of anomalies that were previously detected by Achilles [39] and Murdock [136] testing tools for the health datasets. In the worst case, ADQuaTe2 could not detect 3.86% of anomalies that were previously detected by these tools, which indicates that the autoencoder could not discover all of the associations among the attributes.

RQ1.b: Can ADQuaTe2 detect the anomalies in real-world health and plant data that were missed by domain experts?

In this evaluation, we asked experts to use domain knowledge to validate the detected suspicious records. Given AF, the set of anomalous records that are flagged by the domain expert as actually anomalous, we define the *Newly Detected* (ND) metric to evaluate the constraint discovery effectiveness of ADQuaTe2.

ND: Percentage of suspicious records detected by ADQuaTe2 that are real anomalies not previously detected.

$$ND = \frac{|AF - E|}{|A|} \tag{4.8}$$

Table 4.7 shows the values of *ND* of ADQuaTe2 for the datasets. ADQuaTe2 could detect between 33.33% to 35.63% actual anomalies that were not previously detected. Domain experts interpreted the remaining detected records as *unusual* but possible, *suspicious* if they needed more investigations, and *valid* if they were not actually faulty. It took one hour for the plant domain expert to inspect the 16 groups of 89 reported records. It also took one hour for the health domain expert to inspect 23 groups of 6848 reported records.

Table 4.7: Newly Detected Anomalies by ADQuaTe2 for Plant and Health Datasets

Dataset ID	**Domain**	**ND**	**100-ND**		
			Unusual	Suspicious	Valid
1	Plant	35.63	22.99	13.79	27.59
3	Health	33.33	66.66	0.00	0.00

4.2.1.2 Goal 2: Evaluate the anomaly interpretation effectiveness.

By answering the following questions we demonstrated that ADQuaTe2 can effectively explain the anomalies to the experts.

RQ2: To what extent do the generated visualization plots correctly explain the reason behind the invalidity of anomalies?

We define the *Visualization Efficiency* (VE) metric based on the number of visualization plots to evaluate the anomaly interpretability effectiveness of our approach.

VE: Percentage of plots that could correctly explain the reason behind the invalidity of the suspicious groups.

Table 4.8 shows the visualization efficiency of ADQuaTe2 for two datasets. Between 53.33% to 100.00% of the plots correctly explained the reasons behind invalidity of the records.

Table 4.8: Visualization Efficiency of ADQuaTe2 for Plant and Health Datasets

Dataset ID	**VE**	
	s-score per attribute	Decision tree
1	73.33	53.33
3	100.00	83.33

4.2.1.3 Goal 3: Evaluate the accuracy improvements using UCI datasets.

By answering the following questions we demonstrated that the anomaly detection effectiveness of ADQuaTe2 improves after retraining the machine learning model.

RQ3.a: Does the number of correctly detected anomalies increase after retraining the machine learning model with the help of feedback from the domain expert?

Given E, the set of anomalous records detected by an existing data quality test approach, A the set of anomalous records detected by ADQuaTe2, and AF the set of anomalous records that are flagged by domain expert as actually faulty, we use the *True Positive Rate* (TPR), *Number of Runs* (NR), and *True Positive Growth Rate* ($TPGR$) to answer this question.

TPR: Percentage of actual faulty records that are correctly identified as anomalous.

$$TPR = \frac{|AF|}{|A|} \tag{4.9}$$

NR: Total number of times a domain expert revalidates data until reaching the desired TPR.

TPGR: Percentage change of a TPR variable within the interactive learning period.

$$TPGR = (\frac{TPR_{NR}}{TPR_1})^{\frac{1}{NR}} - 1 \tag{4.10}$$

where TPR_1 is the true positive rate at the first run and TPR_{NR} is the true positive rate at the last run.

RQ3.b: Can the actual faults detected by ADQuaTe2 in the previous runs still be detected after retraining the model?

To answer this question, we use the *False Negative Rate* (FNR) and *False Negative Growth Rate* ($FNGR$) metrics.

FPR: Percentage of undetected anomalies (UD) plus percentage of actual faulty records detected in previous runs that could not be detected in the current run.

$$FNR = \frac{|AF_{old} - AF_{new}|}{|AF_{old}|} + UD \tag{4.11}$$

where AF_{old} is the set of actual faults detected in previous runs and AF_{new} is the one detected in the current run.

FNGR: Percentage change of a FNR variable within the interactive learning period.

$$FNGR = \left(\frac{FNR_{NR}}{FNR_1}\right)^{\frac{1}{NR}} - 1 \tag{4.12}$$

where FNR_1 is the false negative rate at the first run and FNR_{NR} is the false negative rate at the last run.

Figure 4.7 shows how the true positive rate increases over time during the retraining process for the datasets under test. Table 4.9 shows positive values for $TPGR$ for all of these datasets, which demonstrates that the anomaly detection effectiveness of ADQuaTe2 improves after retraining the machine learning model. Figure 4.8 shows how the false negative rate decreases over time during the retraining process for the datasets under test. Table 4.9 shows negative values for $FNGR$ for all of these datasets, which demonstrates that the anomaly detection effectiveness of ADQuaTe2 improves after retraining the machine learning model.

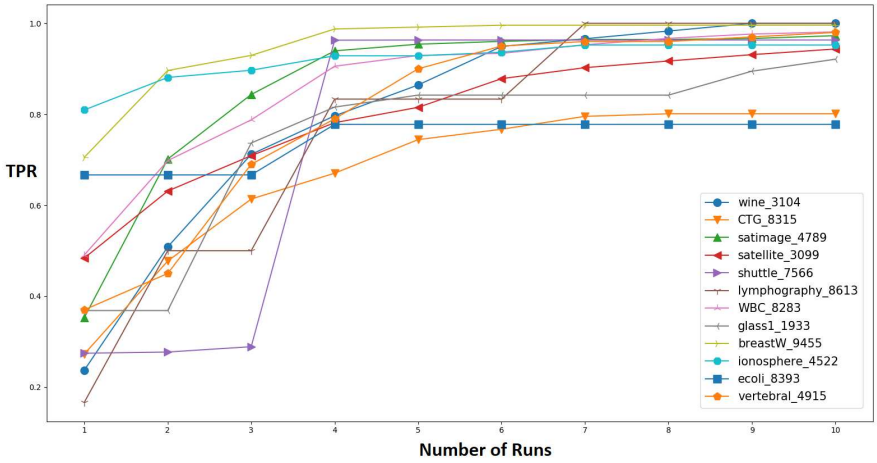


Figure 4.7: Improvement in True Positive Rate for UCI Datasets

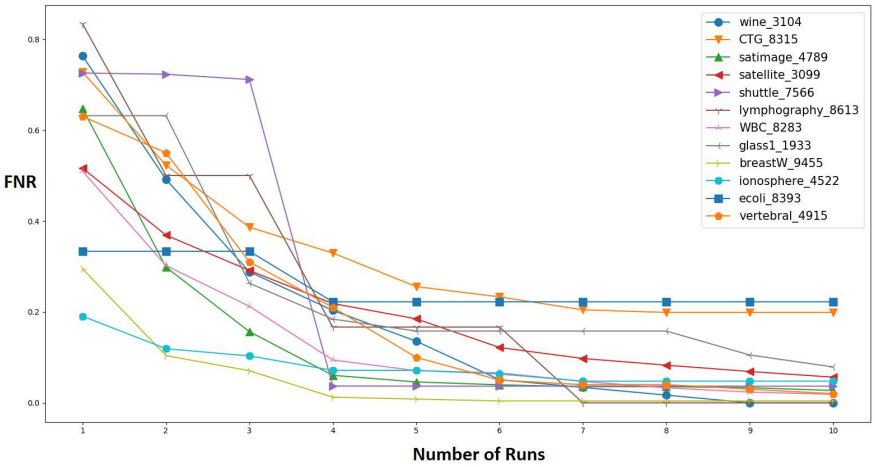


Figure 4.8: Improvement in False Negative Rate for UCI Datasets

Table 4.9: True Positive and False Negative Growth Rate for UCI Datasets for 10 Runs

Dataset ID	TPGR	FNGR
9	0.15	-1
10	0.127	-0.16
11	0.21	-0.17
12	0.13	-0.16
13	0.48	-0.27
14	0.22	-0.12
15	0.038	-0.26
16	0.01	-0.20
17	0.10	-0.27
18	0.069	-0.19
19	0.133	-0.25

Based on Figures 4.7 and 4.8, ADQuaTe2 could detect on average 44% of the known anomalies in the UCL datasets in its first execution. ADQuaTe2 can detect on average 95% of the known anomalies from the UCI ML datasets after retraining the learning model ten times using ground truth knowledge. The TPR and FNR values almost stabilized after four iterations. This shows that the payoff is not worth the cost of running ADQuaTe2 after four iterations for these datasets.

4.2.1.4 Goal 4: Evaluate hyper-parameter tuning.

Table 4.10 shows the hyper-parameters and their range of values that form 300 different models that we used for the grid search approach. There are 30 network architectures (i.e., hidden layers and neurons in each layer) and 10 different values of epochs (i.e., complete passes through the training data) for each architecture. ADQuaTe2 initializes the network hyper-parameters using these values to select the best model.

Table 4.10: Hyper-parameters for Best Model Selection

Hyper-parameter	Minimum	Maximum	Step size
#Hidden layers	1	3	1
#Hidden Nodes	10	100	10
#Epochs	10	100	10

We demonstrated that the original grid search approaches based on RE minimization do not select the most effective model by answering *RQ4.a*. We also demonstrated that the training process must be stopped before the network is overfitted on the training data by answering *RQ4.b*.

RQ4.a: Does the model with the lowest RE maximize TPR?

We demonstrated that a model with the lowest RE has the potential to generate false alarms by calculating RE and TPR for different autoencoder models. Figures 4.9 and 4.10 show examples of RE (blue bars) and TPR (orange bars) for different autoencoder architectures for the *Lymphography* and *Ecoli* datasets. In these figures, the horizontal axis indicates 30 different autoencoder models in form of $[n_1, ..., n_h]$, where n_i is the number of neurons in the i^{th} layer and h is the total number of layers. For example, the first value in the horizontal axis is equal to $[10, 10, 10]$, which

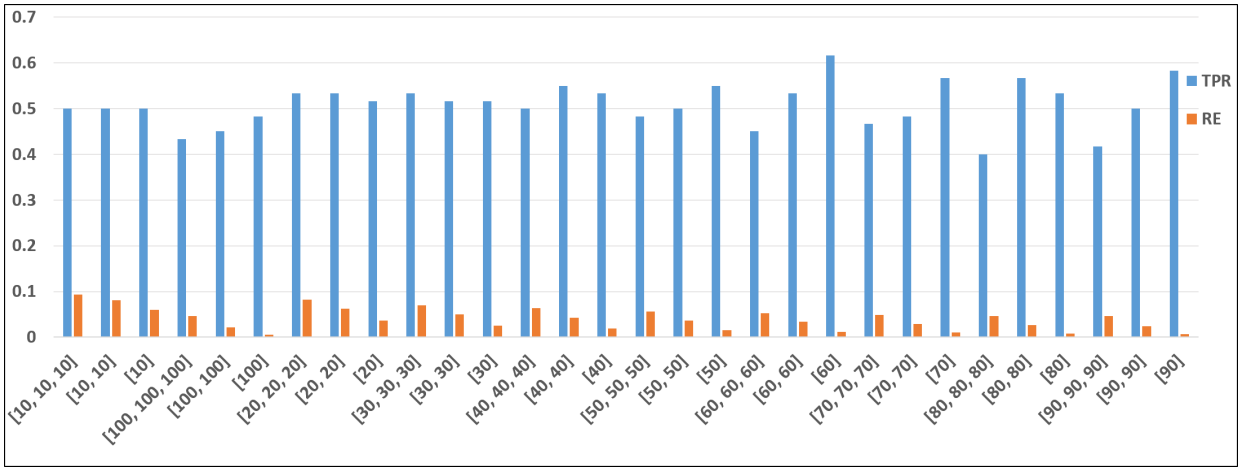


Figure 4.9: RE and TPR per Autoencoder Architecture for Lymphography Dataset

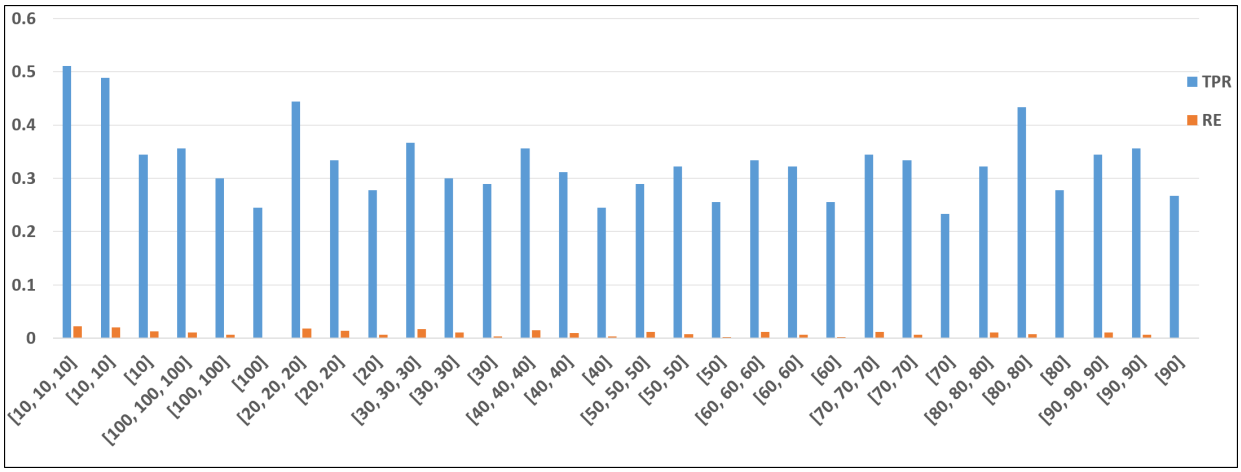


Figure 4.10: RE and TPR per Autoencoder Architecture for Ecoli Dataset

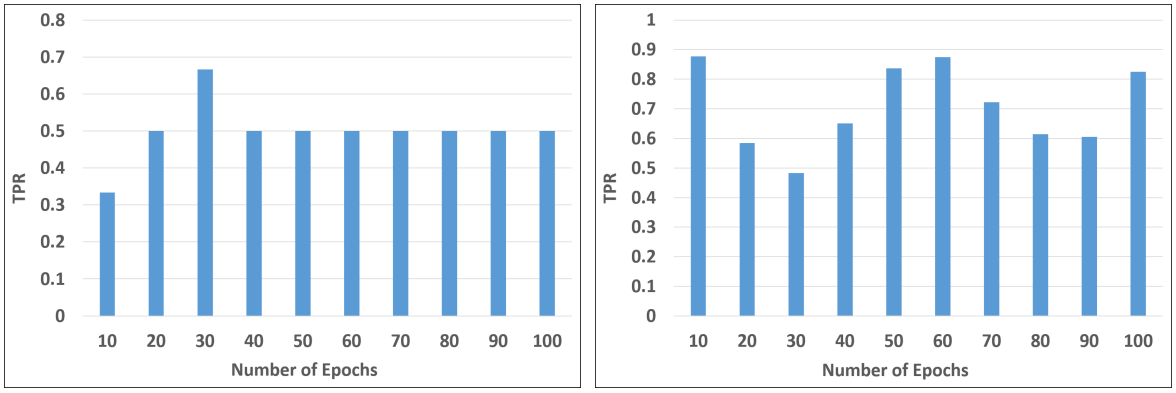
indicates an autoencoder with three hidden layers and 10 neurons in each layer. The vertical bars are values of TPR and RE in the range of $[0, 1]$. The plot for the *Lymphography* dataset shows the values of TPR in the $[0.40,\ 0.62]$ range and the values of RE in the $[0.006,\ 0.09]$ range for the 30 network architectures. The plot for the *Ecoli* dataset shows the values of TPR in the $[0.25,\ 0.51]$ range and the values of RE in the $[0.0, 0.03]$ for the 30 network architectures.

Based on these figures, the maximum value of TPR is not necessarily for the network architecture that minimizes the reconstruction error. The maximum TPR value for the *Lymphography* dataset is for the $[60]$ architecture, while the minimum RE value is for the $[100]$ architecture. Similarly, for the *Ecoli* dataset, the maximum TPR value is for the $[10, 10, 10]$ architecture, while the

minimum RE value is for the $[100]$ architecture. These examples show that the most effective network is not the one with the lowest RE. As a result, the current grid-search approaches for autoencoder model selection do not select a model that maximizes the TPR and have potential to generate false alarms.

RQ4.b: What are the best stopping points for different autoencoder architectures?

We demonstrated that we should use an early stopping technique [16] based on the TPR value for an autoencoder architecture. We calculated TPR for different autoencoder models using the UCI datasets. Figure 4.11 shows examples of the TPR values for the different numbers of epochs used to train the autoencoder network against the *Lymphography* and *Satimage* datasets. The horizontal axis indicates the number of epochs and the vertical bars indicate the TPR values for these numbers of epochs. The plot for the *Lymphography* dataset shows the values of TPR that varies between 0.34 and 0.67. The plot for the *Satimage* dataset shows the values of TPR that varies between 0.48 and 0.88 for different numbers of epochs.



(a) Using Lymphography Dataset **(b)** Using Satimage Dataset

Figure 4.11: True Positive Rate for Different Number of Epochs

From Figure 4.11 we see that the value of the true positive rate is affected by the number of epochs chosen for an autoencoder network. The bars in this figure also show that increasing the number of epochs for a specific network architecture does not necessarily result in improvement in

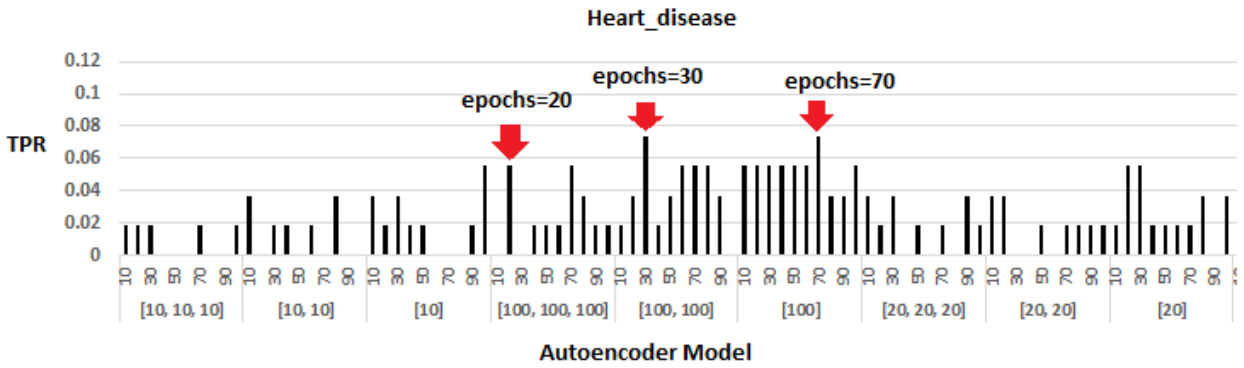


Figure 4.12: Stopping Points for Different Autoencoder Models for Heart_disease Dataset

the anomaly detection effectiveness of the approach (i.e., the value of TPR does not necessarily increase by increasing the number of epochs).

In the proposed early stopping technique, we stop training at the number of epochs after which the TPR value starts decreasing. The vertical bars in Figure 4.12 show the true positive rates for some of the 300 models described in Table 4.10 using the *Heart_disease* dataset. The horizontal axis indicates the autoencoder architectures in the form of $[n_1, ..., n_h]$, where n_i is the number of neurons in the i^{th} layer and h is the total number of layers. This axis also indicates the number of epochs for each architecture. For example, the first part of the horizontal axis shows that there are 10 different values for the number of epochs for an autoencoder in form of $[10, 10, 10]$. The vertical bars indicate the TPR values for these autoencoder models. Figure 4.12 shows that the appropriate stopping points (i.e., the number of epochs that results in the maximum value of TPR) for the $[100]$, $[100, 100]$, and $[100, 100, 100]$ architectures are 70, 30, and 20 respectively.

4.2.1.5 Goal 5: Evaluate the performance of the approach.

We demonstrated that ADQuaTe2 is performance efficient in constraint discovery and anomaly detection by answering the following question.

RQ5: How long does it take to execute ADQuaTe2 against an input dataset?

We measure the Total Time (TT) it takes to perform the automated steps of ADQuaTe2.

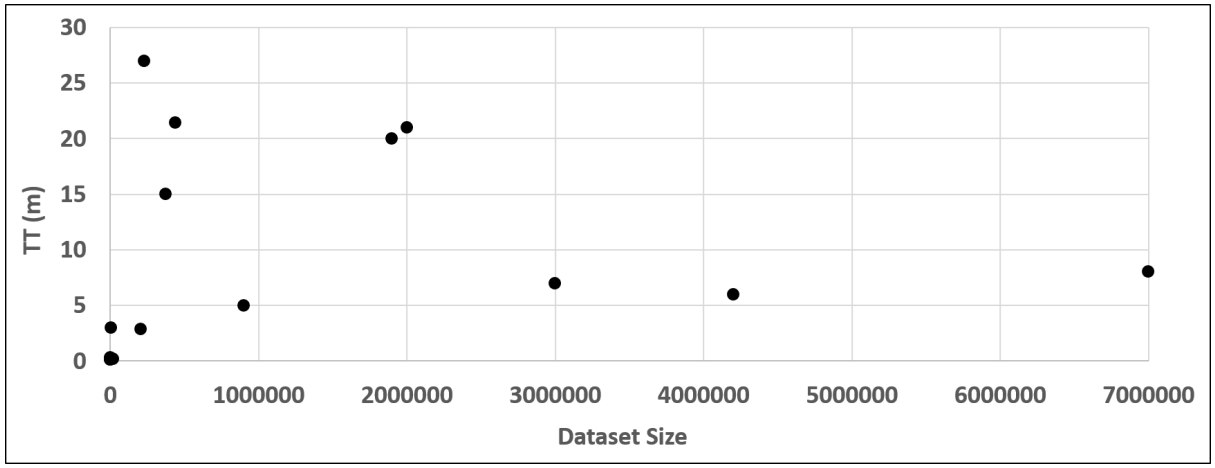


Figure 4.13: Total Time (TT) for Different Dataset Sizes

TT: Time to perform data preparation, constraint discovery, anomaly detection, and anomaly interpretation. The time spent by domain experts is not included because different experts perform the anomaly inspection in different ways, which affects the time measurement.

Figure 4.13 shows values of *TT* for all the datasets under test based on their sizes (equation 4.13) for one execution of ADQuaTe2. It took between 0.138 to 27 minutes to execute the automated steps of ADQuaTe2 for these datasets. As the results show, TT is not necessarily greater for the datasets with larger sizes. This shows that dataset characteristics other than size, such as data types and data sparseness may have also affected the results. The analysis of these other factors on the performance of ADQuaTe2 is the subject of our future work.

$$size = NRe * NAt \tag{4.13}$$

where *NRe* is the number of records and *NAt* is the number of attributes in the input dataset.

4.2.2 Threats to Validity

Internal validity. Anomalies can only be validated by a domain expert at the group level. The expert cannot select individual records in a group that contains both valid and anomalous records. As a result, some records may get incorrect labels while updating the dataset for the retraining phase. However, based on our observations, most of the records in one cluster have similar levels of valid-

ity. There is less than 1% of dissimilar records (i.e., valid in an anomalous group or anomalous in a valid group). This few number of incorrectly labeled records do not affect the overall effectiveness of the retraining process as the autoencoder discovers constraints from the majority of the data in big datasets.

The decision trees describe the constraints violated by each suspicious group. These constraints are expressed in terms of domain attributes and may not represent the constraints discovered by the autoencoder. The constraints discovered by the autoencoder are used to detect the suspicious groups. The reconstruction error of the autoencoder is used to label the data as valid/invalid for the supervised random forest classifier. Thus, there is a connection between the constraints discovered by the autoencoder and the ones described in the decision trees. The decision trees help express constraint violations using domain attributes and make them comprehensible to the domain experts.

External validity. We used four as the number of iterations after which the accuracy of ADQuaTe2 is stabilized. This number was true for the UCI datasets that we used. However, we cannot generalize the number to other datasets. We plan to investigate more datasets in the future.

Construct validity. We evaluated the anomaly detection effectiveness of the approach by measuring the previously detected and newly detected anomalies as true positives and undetected anomalies as false negatives. Using the current definition of undetected anomalies, we cannot measure the false negatives for data sets with no previously known anomalies. Moreover, we cannot ensure that ADQuaTe can detect all the anomalies in such data sets by measuring previously and newly detected anomalies. We evaluated our approach using ground truth data (e.g., from UCI machine learning repository [2]) that contain a set of known anomalies to measure the true positive and false negative rates.

Another threat involves the use of visualization efficiency metric to evaluate the anomaly interpretability effectiveness of ADQuaTe. This measure highly depends on the domain expert and how well s/he can understand the explanations. In this research, we asked experts from plant and health domains to validate the generated visualization plots.

4.3 Summary

We proposed ADQuaTe2 as an instantiation of the ADQuaTe framework for constraint discovery, anomaly detection, and explanation in non-sequence data. Without the intervention of a domain expert, 33.33–35.63% of the suspicious records reported by ADQuaTe2 from two real-world health and plant datasets in the first execution of the tool were actual faults, which were not previously detected. Moreover, ADQuaTe2 could detect on average 98% of the known faults in all the real-world health and plant datasets and 44% of the known faults in the UCL datasets in the first execution. ADQuaTe2 detected on average 95% of the known faults from the UCI ML datasets after retraining the learning model ten times using ground truth knowledge. Between 53.33% to 100.00% of the visualization plots for the plant science and the health datasets correctly explained the reasons behind invalidity of the records.

Chapter 5

IDEAL: An Instantiation of ADQuaTe for Sequence Data

In this chapter, we describe IDEAL [19], which instantiates the ADQuaTe components for sequence data. IDEAL is an automated approach for **I**nteractive **D**etection and **E**xplanation of **A**nomalies using an **L**STM-autoencoder for time-series data. Section 5.1 describes the components of IDEAL and Section 5.2 presents its evaluation. Section 5.3 summarizes the chapter.

5.1 Instantiated Components

In this section, we describe how IDEAL instantiates the data preparation, constraint discovery, anomaly detection, anomaly interpretation, and anomaly inspection components of ADQuaTe for sequence datasets.

5.1.1 Data Preparation

IDEAL instantiates this component for sequence data to transform the data into the right shape for input to the constraint discovery component. IDEAL extracts features as complex dependencies among the input records and attributes, and discovers constraints as complex equations over those features [137]. To transform the input dataset into a form suitable for analysis, we use one-hot encoding [109] and normalization [112] 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 5.3 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 [108]. 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 [107]. 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 [108].

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 [138]. 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.

$$ACF(a,k) = \frac{\sum_{i=1}^{N-k}(d[a](i) - \overline{d[a]})(d[a](i+k) - \overline{d[a]})}{\sum_{i=1}^{N}(d[a](i) - \overline{d[a]})^2} \tag{5.1}$$

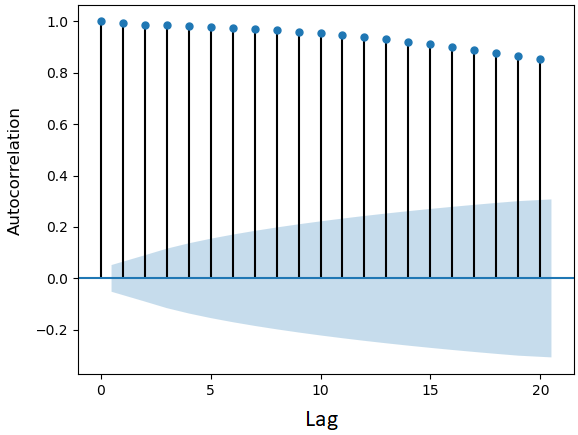


Figure 5.1: ACF for A_4 Attribute in NASA Shuttle for 20 lags

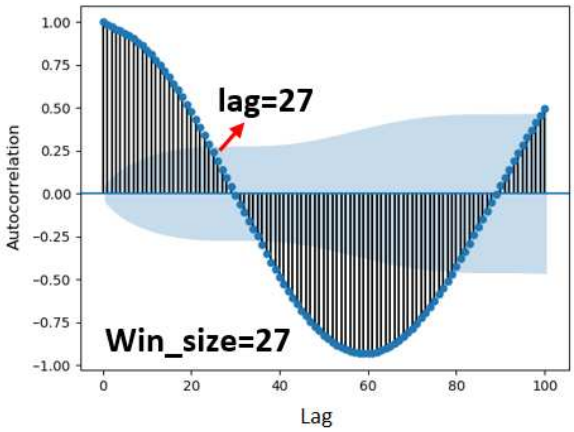


Figure 5.2: Use ACF to Select Window Size

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 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 said to be significantly autocorrelated. The shaded area in Figure 5.1 shows the confidence interval (CI) calculated by Eq. 5.2 for attribute A_4 in the NASA Shuttle dataset.

$$CI = \pm Z_{1-\alpha/2}\sqrt{\frac{1}{N}(1+2\sum_{i=1}^{k}\overline{d[A]}^2)} \tag{5.2}$$

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.

In Figure 5.1, 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. 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 5.2 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.

5.1.2 Constraint Discovery

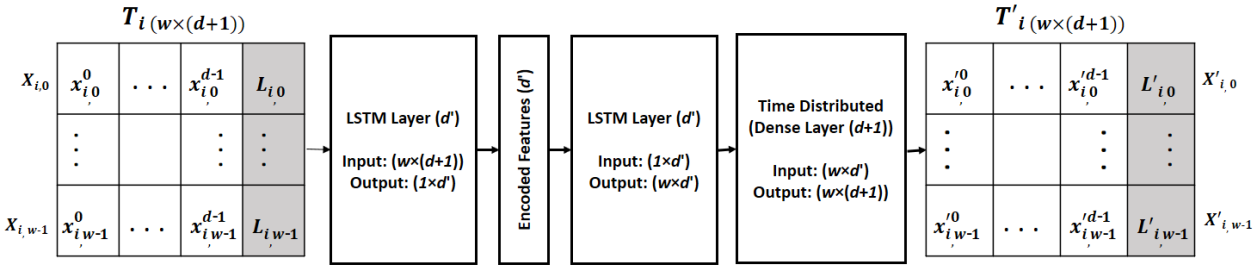


Figure 5.3: Extending LSTM-Autoencoder by Adding a Label Input

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 instantiates this component for sequence data using an LSTM-Autoencoder, which is a sequence-to-sequence modeling technique [137] used to learn time series dependencies.

We extend the LSTM-Autoencoder architecture by adding a label as an additional input to the network structure. The shaded area in Figure 5.3 shows the extension. The input and output are fixed-size time series matrices. $X_{i,j} = [x^0_{i,j}, ..., x^d_{i,j}, L_{i,j}]$ 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 structure is described in Section 2.2. This instantiation redefines 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 5.3 shows the extended reconstruction error.

$$RE = \frac{1}{m}\sum_{i=0}^{m-1}((T'_i - T_i)^2 + (mean(L'_i) - mean(L_i))^2) \tag{5.3}$$

where $mean(L_i)$ is the arithmetic mean of the record labels in time series T_i, $mean(L'_i)$ is the mean of the reconstructed record labels in the reconstructed time series T'_i, and m is the total number of windows.

5.1.3 Anomaly Detection

IDEAL instantiates this component by calculating *s*-score_per_subsequence values to detect suspicious subsequences that violate the constraints discovered by the LSTM-Autoencoder network. Table 5.1 shows an example of a suspicious subequence detected from the NASA Shuttle dataset. The *s*-scores are defined based on the reconstruction error and the record labels. Equation 5.4 shows how to calculate this value.

$$s\_score_i = (\frac{1}{d}\sum_{j=0}^{d-1} s\_score_i^j) + max(L_i) \tag{5.4}$$

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 (Equation 5.5).

Table 5.1: Suspicious Subequence Detected from the NASA Shuttle Dataset

Record Id	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	*s*-score per record
101	0	78	0	24	24	42	55	14	0.0305329
102	0	100	0	34	-23	64	67	4	0.0289624
103	0	79	0	28	10	42	50	8	0.00546589
104	1	83	0	36	0	45	46	2	0.00237685
105	0	77	0	6	-22	40	72	32	0.00771146
106	0	80	5	10	0	43	70	26	0.0037009
107	0	78	0	2	-10	41	75	34	0.00708929
108	0	77	0	8	11	40	69	30	0.0062236
109	0	107	0	30	3	70	76	6	0.0451583
110	0	77	7	20	-6	41	57	16	0.0032065

$$s\_score\_a_i^j = Normalized(\frac{1}{w}\sum_{k=0}^{w-1}(x^j_{(i+k)} - x'^j_{(i+k)})^2) \tag{5.5}$$

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.

In Equation 5.4, $(\frac{1}{d}\sum_{j=0}^{d} s\_score_i^j)$ 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 reported as confirmed anomalous. These subsequences are reported in a separate group from the suspicious subsequences and have higher *s*-score values (in the range [1,2]).

The last column in Table 5.1 shows the *s*-score_per_record values for the suspicious subsequence from the NASA Shuttle Dataset. Equation 5.6 in Section 5.1.4 describes how these values are calculated based on the LSTM-Autoencoder reconstruction error and the record labels. IDEAL highlights the records that are major causes of invalidity in each suspicious sequence. In Table 5.1, the highlited record has the largest value of *s*-score_per_record and is the major cause of the invalidity of this subsequence.

5.1.4 Anomaly Interpretation

IDEAL instantiates this component through displaying s-score per record in a suspicious subsequence and visualization plots of two types for that subsequence: (1) s-score_per_attribute and (2) decision trees.

***s*-score_per_record.** IDEAL uses Equation 5.6 to calculate the invalidity level of the q^{th} record in the subsequence T_i detected as suspicious by our approach. The value of s-score_per_record is calculated based on the LSTM-Autoencoder reconstruction error and the labels obtained using domain expert feedback. 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 5.6 shows how to calculate this value.

$$s\_score\_r_{i,q} = Normalized(\frac{1}{d}\sum_{k=0}^{d-1}(x'^{k}_{i,q} - x^{k}_{i,q})^2 + (L'_{i,q} - L_{i,q})^2) \tag{5.6}$$

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. The anomalous records in each subsequence are identified based on a threshold value calculated using Equation 3.2 described in Section 3.2.5.3.

***s*-scores_per_attribute plot.** For each attribute a_i^j in a suspicious subsequence i, this instantiation calculates its s-score $s\_score\_a_i^j$ based on Equation 5.5. 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. Figure 5.4 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 more than one attributes are compared based on their values of s-score_per_attribute.

Decision tree. For each attribute of the subsequence, the 18 time series features listed in Table 2.4 are extracted using Tsfeatures CRAN library [1]. Next, decision trees are generated through a

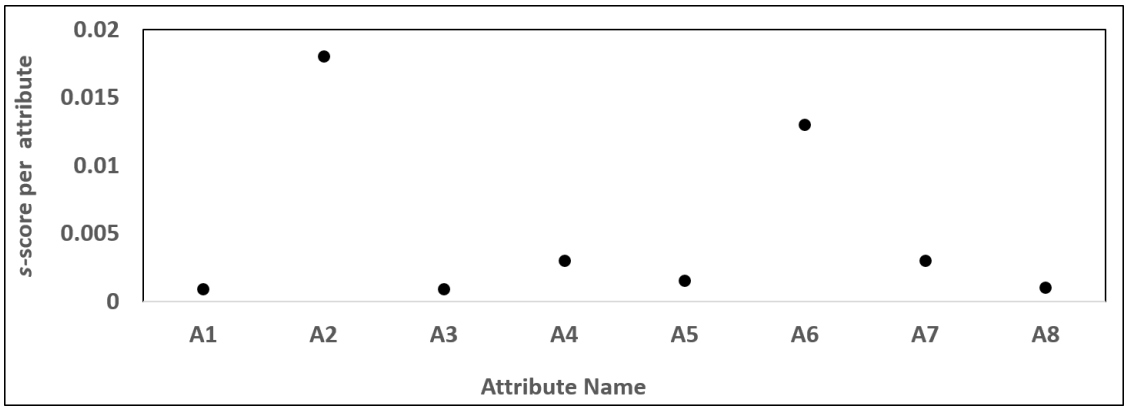
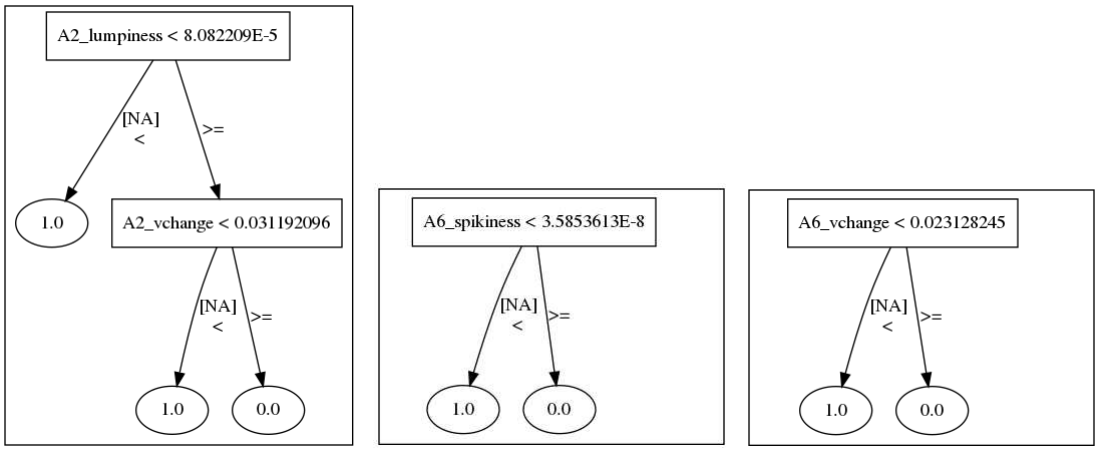


Figure 5.4: *s*-score per Attribute Plot for Suspicious Subsequence Detected from NASA Shuttle Dataset

random forest classifier using the suspicious subsequences labeled as *invalid* and all the non-suspicious sequences labeled as *valid*.

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 5.5: Decision Trees for Suspicious Sequence in NASA Shuttle Dataset

Figure 5.5 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 as follows. These constraints are in form of "**If** $Pred$ **then sequence is** <valid|invalid>" rules, which determine whether a subsequence is valid or invalid based on its features 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).

First Decision Tree

- **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

Second Decision Tree

- **IF** spikiness(A_6) $< 3.58e^{-8}$, **THEN** sequence is invalid

Third Decision Tree

- **IF** vchange(A_6) < 0.02, **THEN** sequence is invalid

The first decision tree shows constraint violations over the lumpiness and vchange features of the time series for the A_2 attribute. Based on this tree, the variance of the variances across multiple blocks in the time series for the A_2 attribute must be greater than or equal to $8.08e^{-5}$. Moreover, the maximum difference in variance between consecutive blocks in time series for the A_2 attribute must be greater than or equal to 0.03.

The second decision tree shows a constraint violation over the spikiness feature of the time series for the A_6 attribute. This feature, which is calculated based on the size and location of the peaks and troughs in time series for the A_6 attribute must be greater than or equal to $3.58e^{-8}$.

The third decision tree shows a constraint violation over the vchange feature of the time series for the A_6 attribute. The maximum difference in variance between consecutive blocks in the time series for the A_6 attribute must be greater than or equal to 0.02.

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.

5.1.5 Anomaly Inspection

The anomaly inspection component is not customized since this approach works as is with both non-sequence and sequence data. This approach is described in the anomaly inspection component of the framework (Section 3.2.5).

5.2 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 Energy datasets. 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 [139], 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.

5.2.1 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 typ-

ical 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 5.2 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 anomalous records from these datasets before applying the mutation operators.

Table 5.2: 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

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 one at a time. We describe these operators below.

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 5.2. For example, the mean value of delivered power to households 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
$r = random(0.05 \times size(D), 0.1 \times size(D))$.

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 $A = R_m[a]$.

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.

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.

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. For example, an attacker may modify the functionality of a sensor to make it produce incorrect values to cause integrity violations.

5.2.2 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 [140] 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 [140].

$$F1 = 2 \times \frac{(precision \times recall)}{(precision + recall)} \tag{5.7}$$

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.

5.2.2.1 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 IDEAL 187, 245, and 8100 seconds to run 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

Table 5.3: $F1_r$ Scores of Different Approaches [3, 4] 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

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. [3] using the same univariate Yahoo synthetic datasets containing known anomalous records. Table 5.3 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 [4] using the same multivariate NASA Shuttle dataset containing known anomalous records. The last five columns of Table 5.3 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?

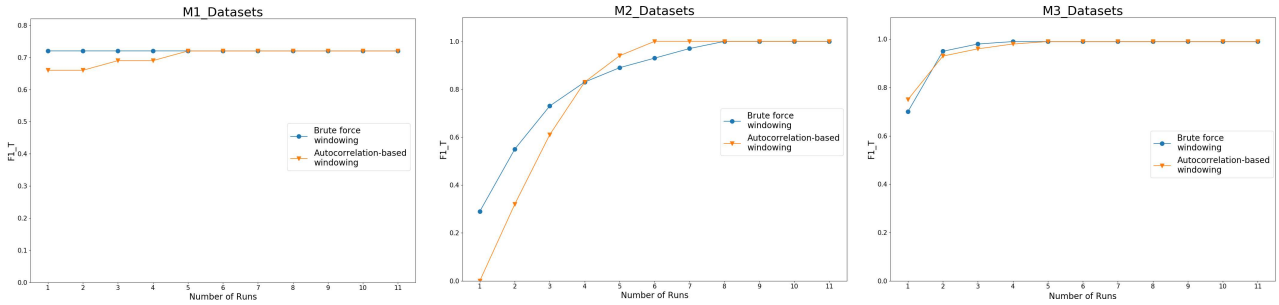
This evaluation was performed using mutated datasets described in Table 5.4. Synthetic_1 and Synthetic_4 are univariate datasets that are mutated using the operators M_1–M_5. Column $|M|$ shows the number of mutants generated from each dataset. Column $|A|$ shows the number of attributes randomly selected from the NASA Shuttle dataset to apply the operators M_1–M_5.

Table 5.4: F1 Score Results for one execution of IDEAL

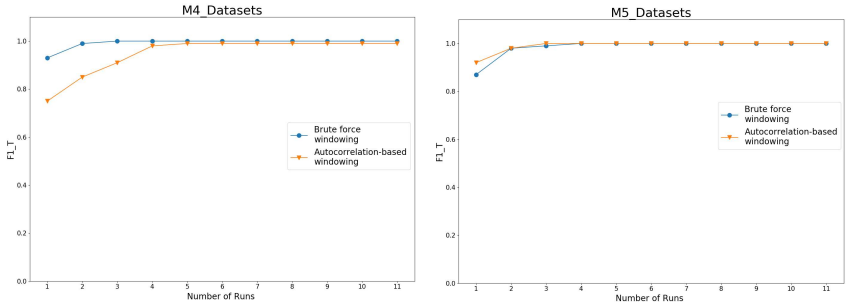
Dataset ID	**Operator**	$\boldsymbol{\|M\|}$	$\boldsymbol{\|A\|}$	$\boldsymbol{F1_t}$	$\boldsymbol{F1_a}$	$\boldsymbol{F1_f}$
Synthetic_1	M_1	2	1	1.0	NA	0.6
Synthetic_1	M_2	99	1	0.8	NA	0.45
Synthetic_1	M_3	179	1	0.6	NA	0.49
Synthetic_1	M_4	263	1	0.98	NA	0.62
Synthetic_1	M_5	88	1	0.78	NA	0.56
Synthetic_4	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

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.

Figure 5.6 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 5.4 that are mutated using



(a) $F1_tGR = 0.001$ For M_1 **(b)** $F1_tGR = 0.142$ For M_2 **(c)** $F1_tGR = 0.033$ For M_3



(d) $F1_tGR = 0.041$ For M_4 **(e)** $F1_tGR = 0.009$ for M_5

Figure 5.6: Average $F1_t$ for Mutated Datasets using Two Types of Windowing

different operators. For example, each data point in the first plot (Figure 5.6 (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?

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.

$$F1_tGR = (\frac{F1_{tNR}}{F1_{t1}})^{\frac{1}{NR}} - 1 \tag{5.8}$$

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 5.6 (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.

5.2.2.2 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 and N_a be the numbers of actual invalid 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 5.4 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 5.2.

5.2.2.3 Goal 3: Performance of constraint discover 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.

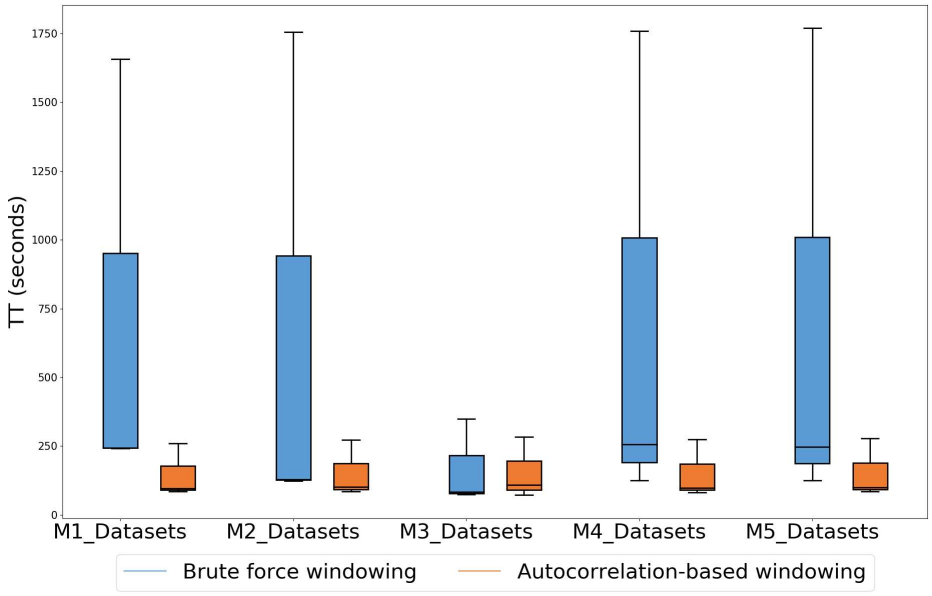


Figure 5.7: TT Box Plots for Datasets Mutated by M_1–M_5

Figure 5.7 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.

5.2.3 Threats to Validity

Internal validity. We used 18 time series features from the TSFreash CRAN library [82] to explain the constraint violations. There are more features than the ones we used in this study to describe a time series. For example, there are 300 features defined in the TSFresh Python library [141]. Using those features may result in reporting other constraint violations than the ones reported by IDEAL. However, many of those features are not easily understandable by humans. Moreover, we used a smaller set of features to decrease the computational complexity of the approach.

The decision trees express constraint violations using domain attributes, which may not represent the constraints discovered by the LSTM-Autoencoder, which were used to detect the suspicious sequences. However, there is a connection between the constraints discovered by the LSTM-Autoencoder and the ones described in the decision trees. The reconstruction error of the LSTM-Autoencoder is used to label the data as valid or invalid for the supervised random forest classifier. The decision trees help express constraint violations using domain attributes and make them comprehensible to the domain experts.

We assume that the mutated datasets have no other faults other than the seeded ones, and that the other datasets only have known anomalies. The existence of other faults could affect the evaluation results (i.e., lowering the calculated accuracy of IDEAL if the tool were to correctly detect the unknown 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. In future, we will use data from other domains to evaluate IDEAL.

5.3 Summary

We instantiated the ADQuaTe framework for sequence data in an LSTM-Autoencoder-based approach called IDEAL that finds and explains anomalies in multivariate time-series data. We proposed an autocorrelation-based windowing technique to automatically identify the input size of the LSTM-Autoencoder network. Decision trees were generated to explain the detected suspicious subsequences and records. Domain expert feedback was 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, NASA Shuttle, and Energy datasets. We demonstrated that the autocorrelation-based splitting of the input data is almost as effective but 3.88 times faster 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. These $F1_t$ values indicate the generation of false alarms by IDEAL as a result of using the unsupervised LSTM-Autoencoder. However, the true positive and false negative rates improved (the growth rate of the $F1_t$ score was positive and 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. We demonstrated that the *s*-score per attribute plots could identify the attributes that were major causes of invalidity in each subsequence with an accuracy between 58 to 100 percent. Moreover, the decision tree plots could identify the constraint violations over features in each subsequence with 28 to 80 percent accuracy.

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".

Chapter 6

Conclusions and Future Work

This research proposes a framework called ADQuaTe that uses unsupervised machine learning techniques to model constraints involving complex relationships among the records and attributes of the data and detects suspicious records or sequences that do not satisfy the discovered constraints. ADQuaTe uses an explainable learning technique to interpret the constraints that are violated by each group or sequence of suspicious records. ADQuaTe works in two modes: (i) with domain expert feedback and (ii) without using domain expert feedback. It can be executed in one iteration in an unsupervised manner without using the feedback loop. Moreover, ADQuaTe allows domain experts to inspect the reported suspicious records. It feeds the constraint discovery and fault detection components with the information received from the experts to minimize false alarms.

ADQuaTe2 is an instantiation of ADQuaTe for non-sequence data using an autoencoder for constraint discovery. We used non-sequence datasets from a health data warehouse from Anschutz Medical School and a plant diagnosis database from Colorado State University to evaluate ADQuaTe2. Our approach discovered new constraints in the attributes of these datasets that were missed by domain experts and detected faults that were not previously detected by the existing tools. We also used non-sequence data with ground truth knowledge from the UCI repository to evaluate the interactive process in ADQuaTe2. We demonstrated that the true positive and false negative rates of detected anomalies improved after incorporating the ground truth knowledge and retraining the learning model. Using the same datasets, we demonstrated that the training process must be stopped at the number of epochs after which the true positive rate starts decreasing for controlling overfitting on the training data.

IDEAL is an instantiation of ADQuaTe for sequence data using an LSTM-Autoencoder for constraint discovery. We demonstrated that IDEAL can detect previously known anomalies in the Energy dataset and also those that we created using mutation analysis on 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. We demonstrated that the true positive and false negative rates improved after incorporating ground truth knowledge about the injected faults. Finally, we demonstrated that the visualization plots could effectively explain the constraints violated by the suspicious subsequences.

In the future, we plan to evaluate ADQuaTe using other types of real-world non-sequence and time-series data. We will improve the constraint discovery effectiveness of ADQuaTe for mixed data types. We will improve the explainability effectiveness of ADQuaTe by directly extracting interpretable information from the complex equations discovered by the autoencoder-based model. We plan to extend ADQuaTe for text data type. We will also extend ADQuaTe to cleanse noise from input data. We are going to instantiate ADQuaTe for streaming data. Finally, we will improve the performance of ADQuaTe to reduce computational cost.

Improve the Constraint Discovery Effectiveness of ADQuaTe for Mixed Data Types. ADQuaTe uses autoencoder-based networks to discover constraints in non-sequence and sequence datasets. The loss function for these networks is called reconstruction error and is defined as the mean square difference between the network input and output. The reconstruction error is the average of the differences between the input attributes and their reconstructions regardless of the attribute types. Using this definition, categorical attributes may carry more weight in the reconstruction error calculation than numeric ones. For example, consider a dataset with one numeric attribute (n) and one categorical attribute (c) with five possible values. The categorical attribute c is converted to five binary attributes ($c_1, ..., c_5$) using the one-hot encoding technique. The reconstruction error is calculated as follows for this dataset.

$$RE = \frac{1}{6}((n' - n)^2 + \sum_{i=1}^{5}(c'_i - c_i)^2) \tag{6.1}$$

where n' is the reconstruction of the numeric attribute n and c'_i is the reconstruction of the categorical attribute c_i for $1 \leq i \leq 5$. This equation shows that the categorical attribute carries five

times more weight in the RE calculation than the numeric attribute. This effect needs to be reduced in the reconstruction error calculation. We propose to evaluate the use of another distance measure used by *k*-prototype clustering techniques [135] to calculate the reconstruction error. Using this definition, the dissimilarity between two values is calculated using the following equation.

$$RE = E(X, X') + C(X, X') \tag{6.2}$$

where X is an input data record, X' is its reconstruction, RE is the dissimilarity measure between X and X', $E(X, X')$ is the Euclidean distance between numeric attributes of X and X', $C(X, X')$ is the number of mismatched categorical attributes between X and X'. For the previous example, $C(X, X')$ is equal to 0 if all the binary attribute values match (i.e., $\forall i \in \{1, ..., 5\}, c_i = c'_i$), and 1 if at least one of the binary attribute values in the input and output (i.e., $\exists i \in \{1, ..., 5\}, c_i != c'_i$) does not match. This definition does not put higher weight on the categorical attributes. We will evaluate the effectiveness of this approach to determine whether the accuracy of constraint discovery and anomaly detection improves by using this new calculation of the reconstruction error.

Improve the Explainability Effectiveness of ADQuaTe. Our current anomaly interpretation approach is based on using an explainable machine learning technique in addition to the autoencoder-based model to add explanations to the constraints discovered by that model. As described in Section 4.2, the constraints expressed in terms of domain attributes may not be similar to the constraints discovered by the autoencoder-based model. We plan to improve the explainability effectiveness of ADQuaTe by directly extracting interpretable information from the complex equations discovered by the autoencoder-based model. Current approaches for the neural network interpretation [142] only extract information about the role of an input attribute in the decision made by the neural network. We also extracted the same information from the autoencoder-based network through identifying the attributes that were major causes of the invalidity of the suspicious records or sequences. We will extend the anomaly interpretation component by extracting other informa-

tion from the trained network, such as the strength of correlations that the network discovers between multiple input attributes. Moreover, we will compare this information with the constraints expressed by the decision trees to demonstrate that the decision trees can correctly describe the constraints discovered by the autoencoder-based model.

Extend ADQuaTe for Textual Data Type. The ADQuaTe framework currently supports categorical and numeric attributes in the input dataset and treats string attributes as categorical. However, text attributes cannot always been described as a set of finite categories. For example, a health dataset may have a *Description* attribute that contains doctor reports after each visit of the patients. This attribute can contain one or more paragraphs of text explaining the patient's status, which cannot be treated as categorical. We plan to extend the data preparation component of ADQuaTe to support textual attributes. We will use Natural Language Processing (NLP) [143] to extract lexical features from the text and use those features as new attributes in the dataset. We will evaluate the effectiveness and efficiency of ADQuaTe using real-world datasets with textual attributes, such as heath data from Anschutz medical campus [22].

Extend ADQuaTe to Denoise Data. Real-world data collected from various sources contains noise (i.e., meaningless data) as a result of hardware failures, programming errors, and gibberish input from measurement tools, such as different types of sensors [144]. It is not trivial to precisely classify an invalid data record as noise or anomaly. Noise is erroneous and perhaps random values in data records, but are not necessarily unusual values [145]. While noise is not interesting for domain experts, anomalies are interesting to investigate using data analysis techniques. Existing noise cleansing techniques use the term weak anomaly (noise) and significant anomaly to distinguish between these two concepts [146]. Statistical and machine learning-based anomaly detection techniques do not perform well in the presence of noise [147], leading to higher probabilities for generating false alarms. We plan to extend ADQuaTe by incorporating an automatic noise cleansing technique [147, 148] in the data preparation component. We will evaluate this extension using both real-world noise as well as random noise that is injected into the input datasets to demonstrate that ADQuaTe can effectively detect significant anomalies in the presence of noise in the data.

Instantiate ADQuaTe for Streaming Data. Today, there is an enormous increase in the generation of streaming time-series data [149]. This data is typically collected from connected real-time data sources, such as environmental sensors [150]. Anomaly detection in streaming data requires processing data in real time and discovering constraints while simultaneously making predictions about anomalous data. Existing anomaly detection approaches for non-streaming data are difficult to execute reliably for streaming data in practice [150]. We plan to extend ADQuaTe to find anomalous records and sequences in multivariate streaming time-series data. We will instantiate the constraint discovery component using a machine learning model, such as Hierarchical Temporal Memory (HTM) [149] that can continuously discover constraints from data. HTM is a time series modeling technique known for its fast and continuous learning, tolerance of noise, and generalization [101]. Moreover, it can deal with changing patterns of data over time. As a result, it is more appropriate than the LSTM-based techniques for modeling streaming time-series data. We will evaluate the effectiveness and efficiency of the approach using real-world data captured online from IoT sensors and from medical records.

Improve ADQuaTe Performance. ADQuaTe uses deep learning along with other learning techniques to effectively detect and explain anomalies in big datasets. This incorporation of several techniques to increase the accuracy of anomaly detection and explanation can lead to increase in computational cost [144], especially for high dimensional datasets with large number of records. We plan to use big data and cloud technologies to reduce the computational cost. We will incorporate parallel and distributed processing to minimize the computational cost. We will use libraries that work in cloud and multi-core environments. We will evaluate the performance of ADQuaTe by processing huge volumes of data in real-time and measuring the reduction in the computational cost.

Bibliography

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

[2] "UCI ML Repository," https://archive.ics.uci.edu/ml/index.php (Accessed 2019-05-14).

[3] Z. Hasani, B. Jakimovski, G. Velinov, and M. Kon-Popovska, "An Adaptive Anomaly Detection Algorithm for Periodic Data Streams", booktitle="Intelligent Data Engineering and Automated Learning," H. Yin, D. Camacho, P. Novais, and A. J. Tallón-Ballesteros, Eds. Springer International Publishing, 2018, pp. 385–397.

[4] S. Shriram and E. Sivasankar, "Anomaly Detection on Shuttle data using Unsupervised Learning Techniques," in *IEEE International Conference on Computational Intelligence and Knowledge Economy*, 2019, pp. 221–225.

[5] A. Singh, N. Thakur, and A. Sharma, "A Review of Supervised Machine Learning Algorithms," in *3rd International Conference on Computing for Sustainable Global Development*, 2016, pp. 1310–1315.

[6] "Isolation Forest," www.pydata.org (Accessed 25-6-2020).

[7] "Understanding LSTM Networks, Recurrent Neural Networks," https://colah.github.io/posts/2015-08-Understanding-LSTMs/ Accessed (21-10-2019).

[8] "Time series decomposition," http://course1.winona.edu/bdeppa/ Accessed (25-10-2019).

[9] J. Bresnick, "Patient Safety Errors are Common with Electronic Health Record Use: Electronic health records are often the culprit in medication errors that negatively impact patient safety," https://healthitanalytics.com/news/patient-safety-errors-are-common-with-electronic-health-record-use (Accessed 2019-08-17).

[10] "Informatica," https://www.informatica.com/ (Accessed 2019-02-12).

[11] C. C. Aggarwal, "An Introduction to Outlier Analysis," in *Outlier Analysis*. Springer International Publishing, 2017, pp. 1–34.

[12] B. N. Saha, N. Ray, and H. Zhang, "Snake Validation: A PCA-based Outlier Detection Method," *IEEE Signal Processing Letters*, vol. 16, no. 6, pp. 549–552, 2009.

[13] C. Zhou and R. C. Paffenroth, "Anomaly Detection with Robust Deep Autoencoders," in *23rd ACM International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 665–674.

[14] B. Kaminski, M. Jakubczyk, and P. Szufel, "A Framework for Sensitivity Analysis of Decision Trees," *Central European Journal of Operations Research*, vol. 26, no. 1, pp. 135–159, 2018.

[15] R. M. Konijn and W. Kowalczyk, "An Interactive Approach to Outlier Detection," in *Rough Set and Knowledge Technology*, J. Yu, S. Greco, P. Lingras, G. Wang, and A. Skowron, Eds. Springer Berlin Heidelberg, 2010, vol. 6401, pp. 379–385.

[16] H. B. Demuth, M. H. Beale, O. De Jess, and M. T. Hagan, *Neural Network Design*, 2nd ed. Martin Hagan, 2014.

[17] H. Homayouni, S. Ghosh, and I. Ray, "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, 2019, pp. 61–68.

[18] H. Homayouni, S. Ghosh, I. Ray, and M. Kahn, "An Interactive Data Quality Test Approach for Constraint Discovery and Fault Detection," in *IEEE Big Data*, Los Angeles, CA, 2019, pp. 200–205.

[19] H. Homayouni, S. Ghosh, I. Ray, S. Gondalia, J. Duggan, and M. Kahn, "An Autocorrelation-based LSTM-Autoencoder for Anomaly Detection on Time-Series Data," *Submitted as a full paper to IEEE Big Data*, 2020.

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

[21] "ADQuaTe," https://github.com/hajarhomayouni/DataQualityTestTool (Accessed 2020-08-11).

[22] "HDC," http://www.ucdenver.edu/about/departments/healthdatacompass/ (Accessed 2020-07-07).

[23] "CSU Plant Diagnostic Clinic," https://plantclinic.agsci.colostate.edu/ (Accessed 2020-05-07).

[24] "Yahoo Server Traffic: A Benchmark Dataset for Time Series Anomaly Detection," Mar. 2020. [Online]. Available: https://yahooresearch.tumblr.com/post/114590420346/a-benchmark-dataset-for-time-series-anomaly

[25] "NASA Shuttle from UCI ML Repository," Mar. 2020. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Statlog+(Shuttle)

[26] "Energy Data," https://energy.colostate.edu/ Accessed (24-6-2020).

[27] T.-K. Huang and J. Schneider, "Learning Hidden Markov Models from Non-Sequence Data via Tensor Decomposition," in *26th International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2013, p. 333–341.

[28] ——, "Learning Auto-Regressive Models from Sequence and Non-Sequence Data," in *24th International Conference on Neural Information Processing Systems*, 2011, p. 1548–1556.

[29] "Breast Cancer Wisconsin (Diagnostic) Data Set," https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic) (Accessed 2020-08-11).

[30] "Glass Identification Data Set," https://archive.ics.uci.edu/ml/datasets/glass+identification (Accessed 2020-08-11).

[31] M. Golfarelli and S. Rizzi, "Data Warehouse Testing: A Prototype-based Methodology," *Information and Software Technology*, vol. 53, no. 11, pp. 1183–1198, 2011.

[32] J. Gao, C. Xie, and C. Tao, "Big Data Validation and Quality Assurance – Issuses, Challenges, and Needs," in *IEEE Symposium on Service-Oriented System Engineering*, 2016, pp. 433–441.

[33] S. Dakrory, T. Mahmoud, and A. Ali, "Automated ETL Testing on the Data Quality of a Data Warehouse," *International Journal of Computer Applications*, vol. 131, no. 16, pp. 0975–8887, 2015.

[34] M. G. Kahn, T. J. Callahan, J. Barnard, A. E. Bauck, J. Brown, B. N. Davidson, H. Estiri, C. Goerg, E. Holve, S. G. Johnson, S.-T. Liaw, M. Hamilton-Lopez, D. Meeker, T. C. Ong, P. Ryan, N. Shang, N. G. Weiskopf, C. Weng, M. N. Zozus, and L. Schilling, "A Harmonized Data Quality Assessment Terminology and Framework for the Secondary Use of Electronic Health Record Data," *EGEMS (Washington, DC)*, vol. 4, no. 1, p. 1244, 2016.

[35] M. A. Weiss, *Data Structures & Algorithm Analysis in C++*, 4th ed. Pearson, 2013.

[36] H. Homayouni, S. Ghosh, and I. Ray, "Data Warehouse Testing." Advances in Computers, 2019, vol. 112, pp. 223 – 273.

[37] "Quality Assurance (QA) by the National Weather Service (NWS)," http://www.nws.noaa.gov/directives/ (Accessed 2018-03-05).

[38] "i-tree Eco," https://www.itreetools.org/ (Accessed 2018-02-11).

[39] "Achilles," https://github.com/OHDSI/Achilles (Accessed 2019-02-12).

[40] G. Hripcsak, J. D. Duke, N. H. Shah, C. G. Reich, V. Huser, M. J. Schuemie, M. A. Suchard, R. W. Park, I. C. K. Wong, P. R. Rijnbeek, J. van der Lei, N. Pratt, G. N. Norén, Y.-C. Li, P. E. Stang, D. Madigan, and P. B. Ryan, "Observational Health Data Sciences and Informatics (OHDSI): Opportunities for Observational Researchers," *Studies in Health Technology and Informatics*, vol. 216, pp. 574–578, 2015.

[41] "PEDSnet," https://pedsnet.org/ (Accessed 2017-05-11).

[42] "Pcori," https://www.pcori.org/ (Accessed 2019-05-15).

[43] D. Loshin, "Rule-based Data Quality," in *11th ACM International Conference on Information and Knowledge Management*, 2002, pp. 614–616.

[44] V. J. Hodge and J. Austin, "A Survey of Outlier Detection Methodologies," *Artificial Intelligence Review*, vol. 22, no. 2, pp. 85–126, 2004.

[45] Y. Chang, W. Li, and Z. Yang, "Network Intrusion Detection Based on Random Forest and Support Vector Machine," in *IEEE International Conference on Computational Science and Engineering (CSE)*, vol. 1, 2017, pp. 635–638.

[46] J. Zhang, M. Zulkernine, and A. Haque, "Random-forests-based network intrusion detection systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 5, pp. 649–659, 2008.

[47] J. Zhang and M. Zulkernine, "A Hybrid Network Intrusion Detection Technique Using Random Forests," in *1st International Conference on Availability, Reliability and Security (ARES'06)*, 2006, pp. 262–269.

[48] 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.

[49] S. B. Kotsiantis, "Decision Trees: a Recent Overview," *Artificial Intelligence Review*, vol. 39, no. 4, pp. 261–283, 2013.

[50] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely Randomized Trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, 2006.

[51] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 3146–3154.

[52] M. Swarnkar and N. Hubballi, "OCPAD: One Class Naive Bayes Classifier for Payload based Anomaly Detection," *Expert Systems with Applications*, vol. 64, pp. 330–339, 2016.

[53] P. Lam, L. Wang, H. Y. T. Ngan, N. H. C. Yung, and A. G. O. Yeh. (2017) Outlier Detection in Large-Scale Traffic Data by Naive Bayes Method and Gaussian Mixture Model Method.

[54] R. Zhen, Y. Jin, Q. Hu, Z. Shao, and N. Nikitakos, "Maritime Anomaly Detection within Coastal Waters Based on Vessel Trajectory Clustering and Naïve Bayes Classifier," *The Journal of Navigation*, vol. 70, no. 3, pp. 648–670, 2017.

[55] J. Thongkam, G. Xu, Y. Zhang, and F. Huang, "Support Vector Machine for Outlier Detection in Breast Cancer Survivability Prediction," in *Advanced Web and Network Technologies, and Applications*, Y. Ishikawa, J. He, G. Xu, Y. Shi, G. Huang, C. Pang, Q. Zhang, and G. Wang, Eds. Springer Berlin Heidelberg, 2008, pp. 99–109.

[56] G. C. Cawley and N. L. Talbot, "On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation," *J. Mach. Learn. Res.*, vol. 11, pp. 2079–2107, 2010.

[57] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion Detection by Machine Learning: A Review," *Expert Systems with Applications*, vol. 36, no. 10, pp. 11 994–12 000, 2009.

[58] M. Markou and S. Singh, "Novelty Detection: a Review—part 1: Statistical Approaches," *Signal Processing*, vol. 83, no. 12, pp. 2481–2497, 2003.

[59] A. Daneshpazhouh and A. Sami, "Semi-supervised Outlier Detection with Only Positive and Unlabeled Data based on Fuzzy Clustering," *Information and Knowledge Technology*, pp. 344–348, 2013.

[60] S. Agrawal and J. Agrawal, "Survey on Anomaly Detection using Data Mining Techniques," *Procedia Computer Science*, vol. 60, pp. 708–713, 2015.

[61] H. Lukashevich, S. Nowak, and P. Dunker, "Using One-class SVM Outliers Detection for Verification of Collaboratively Tagged Image Training Sets," in *IEEE International Conference on Multimedia and Expo*, 2009, pp. 682–685.

[62] R. Domingues, M. Filippone, P. Michiardi, and J. Zouaoui, "A Comparative Evaluation of Outlier Detection Algorithms: Experiments and Analyses," *Pattern Recognition*, vol. 74, pp. 406–421, 2018.

[63] C. C. Aggarwal, "Outlier Analysis," in *Data Mining: The Textbook*. Springer International Publishing, 2015, pp. 237–263.

[64] G. Gan and M. Kwok-Po Ng, "k-means Clustering with Outlier Removal," *Pattern Recognition Letters*, vol. 90, pp. 8–14, 2017.

[65] M. Ahmed and A. Naser, "A Novel Approach for Outlier Detection and Clustering Improvement." IEEE, 2013-06, pp. 577–582.

[66] F. Cao, J. Liang, and L. Bai, "A New Initialization Method for Categorical Data Clustering," *Expert System Applications*, vol. 36, no. 7, pp. 10 223–10 228, 2009.

[67] G. Gan, C. Ma, and J. Wu, *Data Clustering: Theory, Algorithms, and Applications*. Society for Industrial and Applied Mathematics, 2007.

[68] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying Density-Based Local Outliers," in *ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 2000, p. 93–104.

[69] Z. Cheng, C. Zou, and J. Dong, "Outlier Detection Using Isolation Forest and Local Outlier Factor," in *Conference on Research in Adaptive and Convergent Systems*. Association for Computing Machinery, 2019, p. 161–168.

[70] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in *8th IEEE International Conference on Data Mining*. IEEE Computer Society, 2008, pp. 413–22.

[71] R. Thomas and J. Judith, "Voting-based ensemble of unsupervised outlier detectors," in *Advances in Communication Systems and Networks*. Springer, 2020, pp. 501–511.

[72] D. Wilks, "The Multivariate Normal (MVN) Distribution," in *Statistical Methods in the Atmospheric Sciences*, ser. International Geophysics, D. S. Wilks, Ed. Academic Press, 2011, vol. 100, pp. 491–518.

[73] H. Bulut, "Mahalanobis Distance based on Minimum Regularized Covariance Determinant Estimators for High Dimensional Data," *Communications in Statistics - Theory and Methods*, vol. 0, no. 0, pp. 1–11, 2020.

[74] Y. Bengio, A. Courville, and P. Vincent, "Representation Learning: A Review and New Perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[75] "Informatica," https://www.informatica.com/ (Accessed 2017-04-14).

[76] G. Dong and J. Pei, *Sequence Data Mining*. Springer Science & Business Media, 2007, vol. 33.

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

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

[79] 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," vol. 33, pp. 1409–1416, 2019.

[80] "El Nino Data Set," https://archive.ics.uci.edu/ml/datasets/El+Nino (Accessed 2020-08-11).

[81] Q. Wen, J. Gao, X. Song, L. Sun, H. Xu, and S. Zhu, "RobustSTL: A Robust Seasonal-Trend Decomposition Algorithm for Long Time Series," *CoRR*, vol. abs/1812.01767, 2018. [Online]. Available: http://arxiv.org/abs/1812.01767

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

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

[84] 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.

[85] 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.

[86] 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.

[87] "Autoregression Models for Time Series Forecasting with Python," https://machinelearningmastery.com/ Accessed (23-10-2019).

[88] G. A. F. Seber and A. J. Lee, *Linear Regression Analysis*, 2nd ed. Wiley, 2003.

[89] I. J. Myung, "Tutorial on Maximum Likelihood Estimation," *Journal of Mathematical Psychology*, vol. 47, no. 1, pp. 90–100, 2003.

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

[91] A. Hatua, T. T. Nguyen, and A. H. Sung, "Information Diffusion on Twitter: Pattern Recognition and Prediction of Volume, Sentiment, and Influence," in *Proceedings of the Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*. Association for Computing Machinery, 2017, p. 157–167.

[92] D. F. Findley, D. P. Lytras, and T. S. McElroy, "Detecting Seasonality in Seasonally Adjusted Monthly Time Series," *Statistics*, vol. 3, 2017.

[93] N. Tomar, D. Patel, and A. Jain, "Air Quality Index Forecasting using Auto-regression Models," in *IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*, 2020, pp. 1–5.

[94] S. Dhamodharavadhani and R. Rathipriya, "Region-Wise Rainfall Prediction Using MapReduce-Based Exponential Smoothing Techniques," in *Advances in Big Data and Cloud Computing*. Springer, 2019, pp. 229–239.

[95] O. Kramer, *Genetic Algorithm Essentials*. Springer, 2017, vol. 679.

[96] M. R. Mohammadi, S. A. Sadrossadat, M. G. Mortazavi, and B. Nouri, "A Brief Review Over Neural Network Modeling Techniques," in *IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)*, 2017, pp. 54–57.

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

[98] 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.

[99] F. A. Gers, D. Eck, and J. Schmidhuber, "Applying LSTM to Time Series Predictable through Time-Window Approaches," in *Artificial Neural Networks — ICANN 2001*, G. Dorffner, H. Bischof, and K. Hornik, Eds. Springer, 2001, pp. 669–676.

[100] N. I. Sapankevych and R. Sankar, "Time Series Prediction Using Support Vector Machines: A Survey," *IEEE Computational Intelligence Magazine*, vol. 4, no. 2, pp. 24–38, May 2009.

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

[102] R. J. Hyndman, E. Wang, and N. Laptev, "Large-scale Unusual Time Series Detection," in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, 2015-11, pp. 1616–1619.

[103] N. Laptev, S. Amizadeh, and I. Flint, "Generic and Scalable Framework for Automated Time-series Anomaly Detection," in *21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1939–1947.

[104] "How to Decompose Time Series Data into Trend and Seasonality," https://machinelearningmastery.com/decompose-time-series-data-trend-seasonality/ Accessed (25-10-2019).

[105] I. Méndez-Jiménez and M. Cárdenas-Montes, "Time Series Decomposition for Improving the Forecasting Performance of Convolutional Neural Networks," in *Advances in Artificial Intelligence*. Springer International Publishing, 2018, pp. 87–97.

[106] T. Kohonen, "The Self-Organizing Map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

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

[108] 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 *2019 Prognostics and System Health Management Conference*, 2019, pp. 286–290.

[109] W. Zhang, T. Du, and J. Wang, "Deep Learning over Multi-field Categorical Data," in *Advances in Information Retrieval*, ser. Lecture Notes in Computer Science, N. Ferro, F. Crestani, M.-F. Moens, J. Mothe, F. Silvestri, G. M. Di Nunzio, C. Hauff, and G. Silvello, Eds. Springer International Publishing, 2016, pp. 45–57.

[110] "Scaling," https://scikit-learn.org/stable/modules/preprocessing.html (Accessed 2019-03-23).

[111] "One-hot Encoding," https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html (Accessed 2019-06-24).

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

[113] W. Lu, Y. Cheng, C. Xiao, S. Chang, S. Huang, B. Liang, and T. Huang, "Unsupervised Sequential Outlier Detection with Deep Architectures," *IEEE Transactions on Image Processing*, vol. 26, no. 9, pp. 4321–4330, 2017.

[114] G. Williams, R. Baxter, H. He, S. Hawkins, and L. Gu, "A Comparative Study of RNN for Outlier Detection in Data Mining," in *IEEE International Conference on Data Mining*, 2002, pp. 709–712.

[115] W. Zhang and X. Tan, "Combining Outlier Detection and Reconstruction Error Minimization for Label Noise Reduction," in *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE, 2019, pp. 1–4.

[116] C. Zhou and R. C. Paffenroth, "Anomaly Detection with Robust Deep Autoencoders," in *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 665–674.

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

[118] Chen Jin, Luo De-lin, and Mu Fen-xiang, "An Improved ID3 Decision Tree Algorithm," in *4th International Conference on Computer Science Education*, 2009, pp. 127–130.

[119] N. Bhargava, S. Dayma, A. Kumar, and P. Singh, "An Approach for Classification Using Simple CART Algorithm in WEKA," in *International Conference on Intelligent Systems and Control (ISCO)*, 2017, pp. 212–216.

[120] "H2o Random Forest," http://h2o-release.s3.amazonaws.com/h2o/master/1752/docs-website/datascience/rf.html, Accessed (23-6-2020).

[121] "Outlier Detection Datasets," http://odds.cs.stonybrook.edu/ (Accessed 2019-08-17).

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

[123] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for Hyper-parameter Optimization," in *Proceedings of the 24th International Conference on Neural Information Processing Systems*, USA, 2011, pp. 2546–2554.

[124] M. Kampffmeyer, S. Løkse, F. M. Bianchi, R. Jenssen, and L. Livi, "Deep kernelized autoencoders," in *Image Analysis*, P. Sharma and F. M. Bianchi, Eds. Springer International Publishing, 2017.

[125] S. Narayan and G. Tagliarini, "An Analysis of Underfitting in MLP Networks," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, 2005, pp. 984–988.

[126] "H2O," https://www.h2o.ai/products/h2o/ (Accessed 2019-02-25).

[127] "Tensorflow," https://www.tensorflow.org/ (Accessed 2019-05-13).

[128] "Scikit Learn," https://scikit-learn.org/ (Accessed 2019-05-13).

[129] "Pandas," https://pandas.pydata.org/ (Accessed 2020-05-11).

[130] "Statistics," https://docs.python.org/3/library/statistics.html (Accessed 2020-05-11).

[131] "Graphviz," https://pypi.org/project/graphviz/ (Accessed 2020-05-11).

[132] G. Zhong, L.-N. Wang, X. Ling, and J. Dong, "An Overview on Data Representation Learning: From Traditional Feature Learning to Recent Deep Learning," *Journal of Finance and Data Science*, vol. 2, no. 4, pp. 265–278, 2016.

[133] Y. Bengio, "Learning Deep Architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[134] V. Cherkassky and F. M. Mulier, *Learning from Data: Concepts, Theory, and Methods*, 2nd ed. Wiley-IEEE Press, 2007.

[135] R. Madhuri, M. R. Murty, J. V. R. Murthy, P. V. G. D. P. Reddy, and S. C. Satapathy, "Cluster Analysis on Different Data Sets Using K-Modes and K-Prototype Algorithms," in *ICT and Critical Infrastructure: Proceedings of the 48th Annual Convention of Computer Society of India- Vol II*. Springer International Publishing, 2014, pp. 137–144.

[136] "Murdock," https://murdock-study.com/ (Accessed 2019-06-24).

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

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

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

[140] 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 *33rd Annual ACM Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 2015, p. 347–356.

[141] "Time Series Features," https://tsfresh.readthedocs.io/en/latest/ Accessed (28-10-2019).

[142] F. Doshi Velez and B. Kim, "Considerations for Evaluation and Generalization in Interpretable Machine Learning," in *Explainable and Interpretable Models in Computer Vision and Machine Learning*, H. J. Escalante, S. Escalera, I. Guyon, X. Baro, Y. Gucluturk, U. Guclu, and M. van Gerven, Eds. Springer International Publishing, 2018, pp. 3–17.

[143] J. F. Allen, "Natural Language Processing," in *Encyclopedia of Computer Science*. Chichester, UK: John Wiley and Sons Ltd., 2003, pp. 1218–1222.

[144] R. A. A. Habeeb, F. Nasaruddin, A. Gani, I. A. T. Hashem, E. Ahmed, and M. Imran, "Real-time Big Data Processing for Anomaly Detection: A Survey," *International Journal of Information Management*, vol. 45, pp. 289 – 307, 2019.

[145] P.-N. Tan, M. Steinbach, A. Karpatne, and V. Kumar, *Introduction to Data Mining*, 2nd ed. Pearson, 2018.

[146] B. Olivieri, M. Endler, I. Vasconcelos, R. O. Vasconcelos, and M. C. Júnior, "Smart Driving Behavior Analysis Based on Online Outlier Detection: Insights from a Controlled Case Study," pp. 73–78, 2017.

[147] Y. Liu, T. Dillon, W. Yu, W. Rahayu, and F. Mostafa, "Noise Removal in the Presence of Significant Anomalies for Industrial IoT Sensor Data in Manufacturing," *IEEE Internet of Things Journal*, pp. 1–1, 2020.

[148] R. Deb and A. W.-C. Liew, "Noisy Values Detection and Correction of Traffic Accident Data," *Information Sciences*, vol. 476, pp. 132 – 146, 2019.

[149] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised Real-time Anomaly Detection for Streaming Data," *Neurocomputing*, vol. 262, pp. 134 – 147, 2017.

[150] D. J. Hill and B. S. Minsker, "Anomaly Detection in Streaming Environmental Sensor Data: A Data-driven Modeling Approach," *Environmental Modelling & Software*, vol. 25, no. 9, pp. 1014 – 1022, 2010.