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 [25] 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 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. There is no order assumed for the non-sequence data records by existing data analysis approaches [26].

A non-sequence dataset can have a single attribute (d=1) or multiple attributes (d>1). For example, a one-attribute breast cancer dataset may contain values of *tumor size* for different patients. A multiple-attribute *glass identification* data 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: (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 [27], Gao et al. [28], Darkory et al. [29], and Kahn et al. [30].

- **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* [28] and *data conformance* [30] 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 [31]. 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* [28, 29] and *plausibility* [30]. 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', 'F' or 'U' 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* [27] and *logical constraint consistency* [32]. 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) [33] and the US Forest Service's i-Tree Eco [34] are examples of approaches that rely on manual identification of the constraints for weather and climate data. Achilles [35], proposed by the Observational Health Data Sciences and Informatics (OHDSI) [36] community, and PEDSnet [37], proposed by the Patient-Centered Outcomes Research Institute (PCORI) [38], 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 [34]. 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.

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

GuardianIQ [39] 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 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 [39] 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 [40]. 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 [41–43]. 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 [44]. One can analyze the tree to determine the constraints that are violated by each invalid record. However, these trees are prone to overfitting [45]. Random Forest [46] and Gradient Boosting [47] 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 [48, 49]. 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 [50], 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) [51]. 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.

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 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

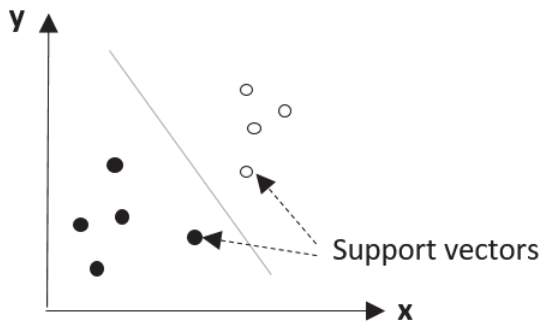

Figure 2.1: SVM Divides Valid/Invalid Records by a Linear Hyperplane [5]

the kernelling approach, the data continues to be mapped into higher attribute dimensions until a hyperplane can be formed to divide it.

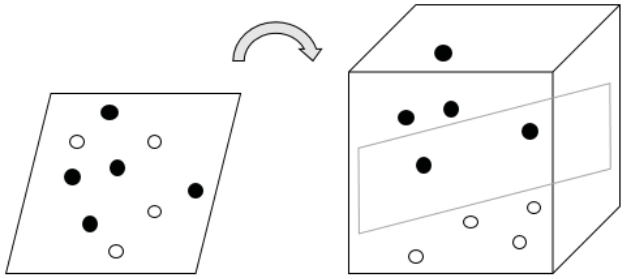

Figure 2.2: SVM Maps Records into a Higher Dimension [5]

SVM is applicable to both Linearly Separable and Non-linearly Separable labeled data records. However, the kernelling approach is sensitive to overfitting [52], 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) [53, 54]. 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 [55]. 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) [56, 57]. 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 hyperplane 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 [58] 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 [59, 60]. 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 [61]. Distance-based clustering algorithms, such as *K*-prototypes [62] cannot derive the complex non-linear relationships that exist among attributes of the data in their clusters [63]. 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 [64]). 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 identified 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 a right 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 do not determine the violated constraints.

Isolation Forest (IF [65]). 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 anomalies. This technique is faster than distance-based techniques, such as clustering and LOF since it does not depend on computationally expensive operations like distance or density calculation [66]. The partitioning process is based on single attribute values and not based on the relationships among the values. Moreover, This technique do not determine the violated constraints.

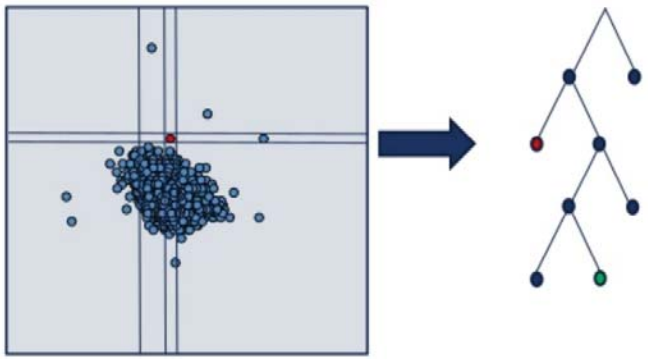

Figure 2.3: Isolation Forest for Anomaly Detection [6]

Elliptic Envelope (EE [67]). Elliptic Envelope is an unsupervised technique that fits a high dimensional Gaussian distribution [68] with possible covariances between attribute dimensions to the input dataset. The technique identifies as anomalous those records that stand far enough from the fit shape. This technique draws an ellipse around the data records, classifying any record inside the ellipse as valid and any record outside the ellipse as anomalous. The EE technique uses the FAST-Minimum Covariance Determinate based on Mahalanobis distance [69] 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 [70]. Representation learning is an unsupervised learning technique that investigates associations in 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. Moreover, the representations investigated by these methods are not human interpretable.

2.1.4 Summary

ML-based approaches are classified as supervised, semi-supervised, and unsupervised. Supervised techniques require manual labeling of 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. 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. Moreover, the techniques have potential to generate false alarms, which can make analysis overwhelming for a data analyst [12].

Recently, efforts have been devoted to explaining the outcomes of machine learning techniques. Researchers at Google Brain have developed a system called Testing with Concept Activation Vectors (TCAV) [71] as a translator for humans that allows a user to ask a black box machine learning technique to what extent a specific attribute has played into its decision. This approach can only indicate the attributes that contribute to the classification of a record as valid and invalid. However, they do not list the constraints violated by that record.

Table 2.3 summarizes and describes existing data quality test approaches in terms of their applicability and the steps they perform.

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 [27] Dakrory et al. [29] Gao et al. [28]		✓	✓			
Kahn et al. [30]	✓		✓			
QA [33] i-Tree Eco [34] Achilles [35] PEDSnet [37]	✓		✓		✓	✓
GuardianIQ [39]		✓	✓		✓	
Informatica [72]		✓		✓	✓	
ML-based: Classification Tree [41–43] Naive Bayesian [48, 49] Support Vector Machines [51] Artificial Neural Network [53, 54] One-Class Support Vector Machine [56, 57] Clustering [59, 60] Representation Learning [12] Local Outlier Factor [64] Isolation Forest [65] Elliptic Envelope [67]		✓		✓	✓	
TCAV [71]						✓

2.2 Data Quality Test Approaches for Sequence Data

Sequence data, also known as time-series data [73], is a set of time-ordered records [74]. 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 [74–76].

A time series T is a sequence of d-dimensional records [74] 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 [74] 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) [75]. A univariate time series has one time-dependent attribute. For example, a univariate time series can consist of daily temperatures recorded sequentially over 24-hour increments. A multivariate time series is used to simultaneously capture the dynamic nature of multiple attributes. For example, a multivariate time series from a climate data store can consist of precipitation, wind speed, snow depth, and temperature data.

Table 2.4: Time Series Features [1]

Feature	**Description**
F_1: Mean	Mean value
F_2: Variance	Variance value
F_3: Lumpiness	Variance of the variances across multiple blocks
F_4: Lshift	Maximum difference in mean between consecutive blocks
F_5: Vchange	Maximum difference in variance between consecutive blocks
F_6: Linearity	Strength of linearity
F_7: Curvature	Strength of curvature
F_8: Spikiness	Strength of spikiness based on the size and location of the peaks and troughs
F_9: Season	Strength of seasonality based on a robust STL [77] decomposition
F_{10}: Peak	Strength of peaks
F_{11}: Trough	Strength of trough
F_{12}: BurstinessFF	Ratio between the variance and the mean (Fano Factor)
F_{13}: Minimum	Minimum value
F_{14}: Maximum	Maximum value
F_{15}: Rmeaniqmean	Ratio between interquartile mean and the arithmetic mean
F_{16}: Moment3	Third moment
F_{17}: Highlowmu	Ratio between the means of data that is below and upper the global mean
F_{18}: Trend	Strength of trend based on robust STL decomposition

The research literature [1, 78] uses various features that describe the relationships among the time-series records and attributes. Trend and seasonality [79] are the most commonly used features. Trend is defined as the general tendency of a time series to increment, decrement, or stabilize over

time [79]. 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 [79]. 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 can violate the constraints over time-series features as *anomalous records* and *anomalous sequences*.

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

Anomalous sequences. Given a set of subsequences $T = \{T_0, ..., T_{m-1}\}$ 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 from non-sequence data, such as Support Vector Machine (SVM) [80], Local Outlier Factor (LOF) [64], Isolation Forest (IF) [65], and Elliptic Envelope (EE) [67] 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 [81].

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.

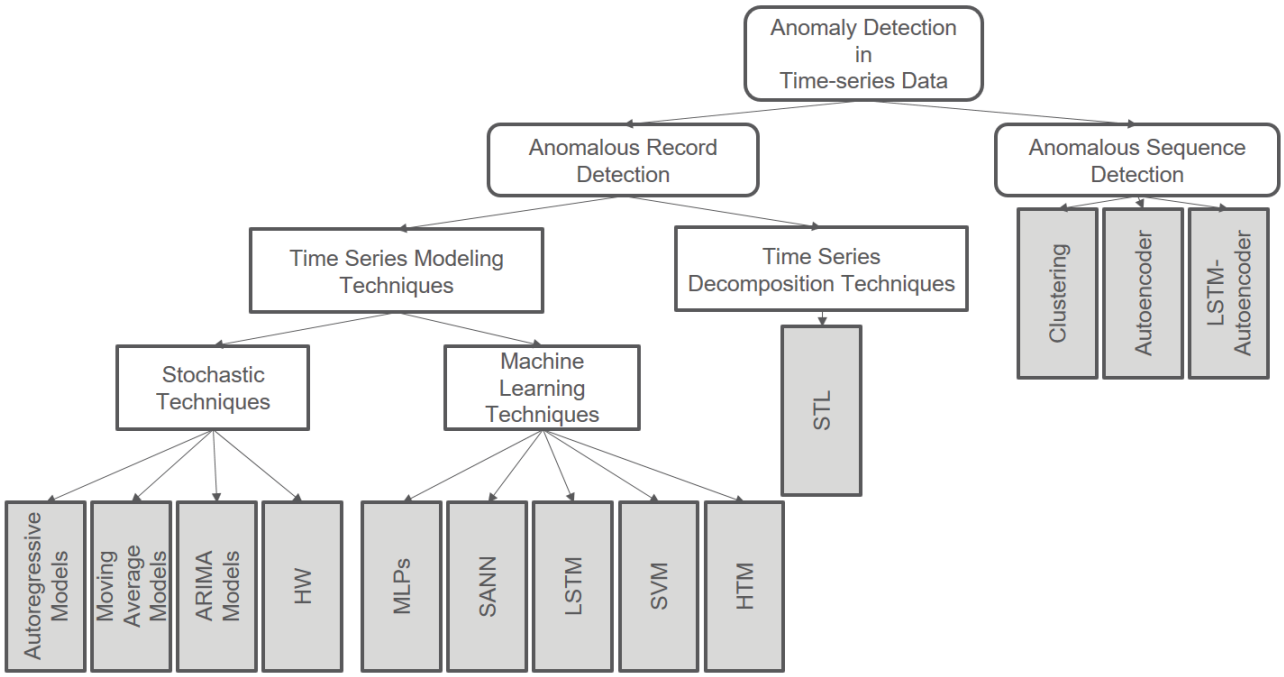

Figure 2.4: Classification Framework for Anomaly Detection Approaches for Sequence Data

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 [79]. Examples of these techniques are *Autoregressive (AR)*, *Moving Average (MA)*, and *Autoregressive Integrated Moving Average (ARIMA)* and *Holt-Winters (HW)* models.

Autoregressive (AR) models [82]. 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 that 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 [83]. 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 [84], likelihood estimators [85], and Yule-Walker equations [79] are typical stochastic techniques used to estimate this model parameters.

The AR model is only appropriate for modeling univariate stationary time-series data [79]. Moreover, it does not consider the non-linear associations between the data records in a time series.

Moving Average (MA) models [86]. 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 [79]. 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 [79].

Autoregressive Integrated Moving Average (ARIMA) models [82]. 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 [87], logging [88], and deflating [89]. 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 [79].

Holt-Winters (HW [3]). An HW models three features of a time series: (1) the mean value, (2) trend, and (3) seasonality. This technique uses exponential smoothing [90] to model the time series.

Exponential smoothing assigns to past records exponentially decreasing weights over time. The objective is to decrease the weight the HW puts on older data records. Three types of exponential smoothing (i.e., triple exponential smoothing) are performed for the three features of the time series. The model requires multiple hyper-parameters: one for each smoothing, the length of a season, and the number of periods in a season. Hasani et. al [3] enhanced this technique (HW-GA) using a Genetic Algorithm [91] 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) [92]. This technique is a type of Artificial Neural Network (ANN) [93], which supports non-linear modeling, with no assumption about the statistical distribution of the data [79]. 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 [93].

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

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 Networ (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 [79].

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

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) [94]. 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 [94]. 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.

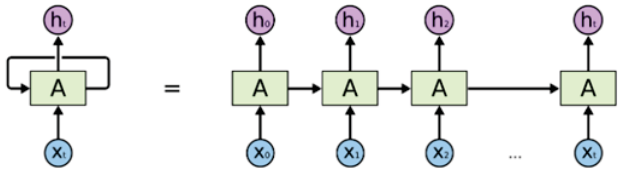

Figure 2.5: An Unrolled RNN [7]

The original RNNs can only learn short-term dependencies among data records by using the recurrent feedback connections [75]. LSTMs extend RNNs by using specialized gates and memory cells in their neuron structure to learn long-term dependencies.

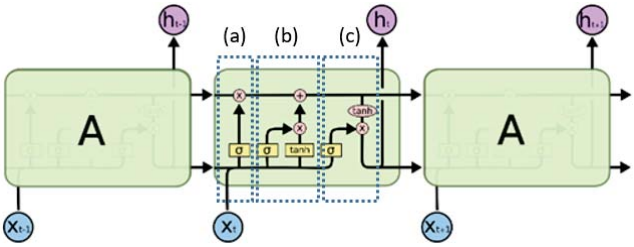

Figure 2.6: LSTM Structure [7]

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.

The 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 to real-world applications. Moreover, LSTMs have ability to learn seasonality [95]. However, the trained network is a complex equation over the attributes of the data records, which is not human interpretable.

Support Vector Machine (SVM [79]). 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 [96], 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 [79]. Moreover, the trained model is not human interpretable.

Hierarchical Temporal Memory (HTM [97]). This is an unsupervised technique that continuously models the 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. By matching stored sequences in the hierarchy with the input, the HTM predicts 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) [98,99]. 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 residuals 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) [100].

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/decreasing frequency/amplitude over time [100].

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.

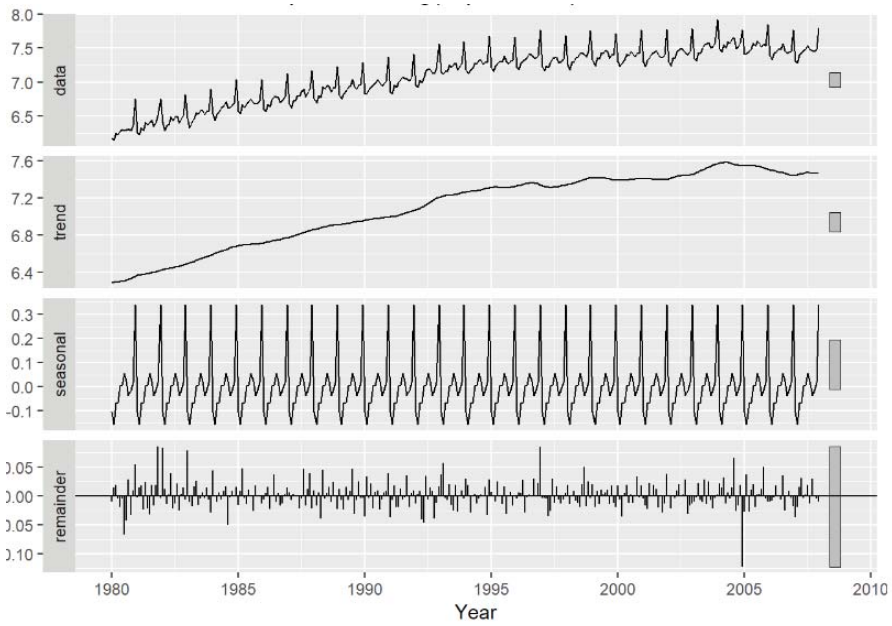

Figure 2.7: STL Decomposition of Liquor Sales Data [8]

Seasonal-Trend decomposition using LOESS (STL) [101]. 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-comlex 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 the real-world datasets are messy and noisy [100]. 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 [99]. These techniques extract subsequence features, such as trend and seasonality. Table 2.4 shows the time series features from the TSFeatures CRAN library [78]. Next, an unsupervised clustering technique, such as K-means [60] and Self-Organizing Map (SOM) [102] 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 [63]. Moreover, these techniques only detect anomalous sequences without determining the records/attributes that are the major causes of invalidity in each sequence.

Autoencoder [74]. 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 com-

presses 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 subsequence, (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 [74]. 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.

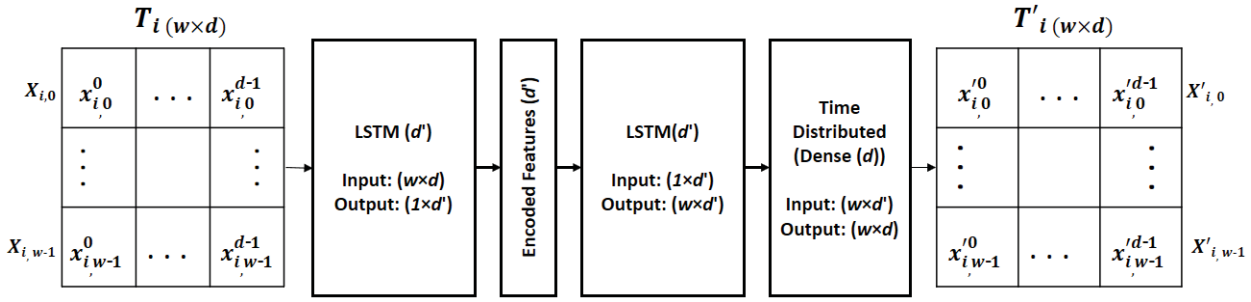

Figure 2.8: An LSTM-Autoencoder Network

Figure 5.3 shows the LSTM-Autoencoder architecture. The input and output are fixed-size time series matrices. $X_{i,j} = [x^0_{i,j}, ..., x^{d-1}_{i,j}]$ 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 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. We have identified the following open problems in testing the time-series data.

Table 2.5: Data Quality Test Approaches for Sequence Data

Approach	**Time Series Type**	**Anomaly Type**	**Modeling Non-linearity**	**Modeling Seasonal-ity**	**Modeling Long-term Dependen-cies**
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	✓		
STL	Univariate	Records	✓	✓	
Clustering	Univariate	Sequences	✓	✓	
Autoencoder	Multivariate	Sequences	✓		
LSTM-Autoencoder	Multivariate	Sequences	✓	✓	✓

Inapplicability to real-world time series. The stochastic time series analysis approaches only focus on univariate time-series data. Moreover, they assume that the time series is linear and follows 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 [103] or by using an exhaustive brute-force approach [104]. 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.

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, and S. Gondalia, "IDEAL: An Approach for Interactive Detection and Explanation of Anomalies using Autocorrelation-based LSTM-Autoencoder for Time-Series Data," *Submitting 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] "HDC," http://www.ucdenver.edu/about/departments/healthdatacompass/ (Accessed 2020-07-07).

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

[23] "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

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

[25] 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.

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

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

[28] 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.

[29] 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.

[30] 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.

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

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

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

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

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

[36] 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.

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

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

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

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

[41] 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.

[42] 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.

[43] 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.

[44] 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.

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

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

[47] 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 Pro-*

cessing 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

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

[55] 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.

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

[57] 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.

[58] 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.

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

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

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

[62] 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.

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

[64] 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.

[65] 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.

[66] 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.

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

[68] 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.

[69] 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.

[70] 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.

[71] 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.

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

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

[74] 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.

[75] 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.

[76] 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.

[77] 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

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

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

[80] 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.

[81] 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.

[82] 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.

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

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

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

[86] 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.

[87] 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.

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

[89] 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.

[90] 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.

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

[92] 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.

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

[94] 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.

[95] 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.

[96] 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.

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

[98] 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.

[99] 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.

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

[101] 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.

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

[103] 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.

[104] 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.

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

[106] 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.

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

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

[109] 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.

[110] 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.

[111] 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.

[112] 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.

[113] 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.

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

[115] 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.

[116] 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.

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

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

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

[120] 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.

[121] 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.

[122] 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.

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

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

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

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

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

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

[129] 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.

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

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

[132] 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.

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

[134] 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.

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

[136] 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.

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

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

[139] 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.

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

[141] 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.

[142] 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.

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

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

[145] 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.