An Interactive Data Quality Test Approach for Constraint Discovery and Fault Detection

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

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

***Abstract*—Data quality tests validate heterogeneous data to detect violations of syntactic and semantic constraints. The specification of these constraints can be incomplete because domain experts typically specify them in an ad hoc manner. Existing automated test approaches can generate false alarms and do not explain the constraint violations while reporting faulty data records. In previous work, we proposed ADQuaTe, which is an automated data quality test approach that uses an unsupervised deep learning technique (1) to discover constraints from big datasets that may have been missed by experts, and (2) to label as suspicious those records that violate the constraints. These records are grouped and explanations for constraint violations are presented to domain experts who determine whether or not the groups are actually faulty. This paper presents ADQuaTe2, which extends ADQuaTe to use an interactive learning technique that incorporates expert feedback to retrain the learning model and improve the accuracy of constraint discovery and fault detection. We evaluate the effectiveness of the approach on real-world datasets from a health data warehouse and a plant diagnosis database. We also use datasets with known faults from the UCI repository to evaluate the improvement in the accuracy of the approach after incorporating ground truth knowledge.**

***Index Terms*—Big Data; Data quality tests; Explainable learning; Interactive learning; Unsupervised learning;**

I. Introduction

Enterprises use databases, data warehouses, and big data appliances to store, manage, and query data for making critical decisions. Records get corrupted because of how data is collected, transformed, and managed, and also because of malicious activities. Inaccurate data leads to incorrect decisions. Thus, rigorous data quality testing approaches are required to ensure that the data is syntactically and semantically correct.

Syntactic constraint validation checks for the conformance of an attribute with the structural specifications in a data model. For example, in a health data store, *patient_age* must take numeric values. Semantic constraint validation checks for the conformance of the 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., $pregnancy\_status = true \rightarrow patient\_gender = female$).

Data quality tests rely on the specification of constraints, which are typically defined by domain experts but often in an ad hoc manner based on their knowledge of the application domain and the needs of the stakeholders. Specifications can be incomplete. For example, the constraint that restricts the values for the day's supply of a drug may be missing. Consequently, a data record in a health data store may contain an incorrect value for that drug. Incorrect values in attributes pertaining to medications can have disastrous consequences if the data is used for patient treatment and in medical research [1].

Tools that automatically generate syntactic constraints exist, but they only check for trivial ones, such as the not-null check [2]. Existing machine learning-based approaches automatically discover non-trivial semantic constraints from the data and report the faulty records as outliers [3]. However, these approaches do not explain which constraints are violated by these records. Consequently, domain experts have to examine a large number of outliers to decide whether or not they are actually faulty and to determine the reason behind the invalidity of the records.

In previous work we proposed a data quality test approach called ADQuaTe [4] that addressed the above issues. 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 that violate the constraints as suspicious, and explains the violations. ADQuaTe uses an unsupervised deep learning technique called autoencoder [5] to discover the constraints associated with the unlabeled records (i.e., records whose validity is not known in advance). We used an autoencoder because its deep architecture can model constraints involving both linear and non-linear relationships among data attributes. Moreover, the unsupervised technique removes the need for labels. Big datasets are usually not labeled and even if labels were available, they would be based only on the specified constraints. Unlike typical data quality testing approaches that mark records as valid/invalid, the fault classification in ADQuaTe is non-binary; each record is assigned a continuous suspiciousness score (s-score) between zero and one. Records that do not conform to the discovered constraints (i.e., records whose s-score is greater than a threshold) are flagged as suspicious. To reduce the time needed to inspect a large number of suspicious records, ADQuaTe uses a Self Organizing Map (SOM) [6] clustering technique to identify a small number of record groups such that the records in each group are likely to violate the same constraints. The contribution of each attribute to the overall suspiciousness score of each group is calculated. Attributes with higher scores in each group are used to generate a decision tree using an explainable machine learning technique called Random Forest classifier [7] to identify the constraints violated by that group.

Previously, we evaluated ADQuaTe on real-world applications and demonstrated that the approach can uncover previously detected as well as new faults in the data. However,

978-1-7281-0858-2/19/$31.00 ©2019 IEEE

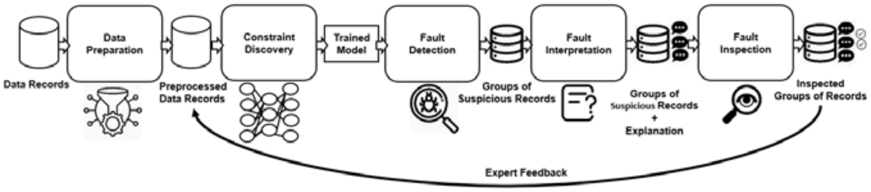


Fig. 1: ADQuaTe2 Approach

ADQuaTe also incorrectly reported as suspicious a number of valid data records and could not detect some previously known faults. These false positives and negatives occurred because ADQuaTe uses an unsupervised technique that has the potential to learn incorrect constraints pertaining to the invalid data. The resulting false alarms can make fault inspection overwhelming for domain experts [8].

In this work, we extend ADQuaTe to minimize false alarms through an interactive learning process [9]. The new approach, ADQuaTe2, allows domain experts to inspect the suspicious groups using a web-based interface and flag as faulty those groups of records that are actually faulty. The feedback is incorporated to label the training data records as faulty or valid. The accuracy of constraint discovery is improved by adding the record label as a new attribute to the training dataset. We extend the autoencoder model by redefining the error function [5] of the autoencoder network based on the record labels. We also extend the autoencoder parameter initialization [10] for the retraining phase by using the parameter values learned in the previous execution of the approach. We redefine the *s*-score calculation based on the record labels. The objective is to ensure that no valid records are reported as suspicious in the retraining phase. Furthermore, the approach tunes the threshold value to reduce false alarms over time.

We evaluated ADQuaTe2 using datasets from a health data warehouse and a plant diagnosis database. We demonstrated that our approach can discover new constraints that were missed by domain experts and detect new faults in these datasets. We also evaluated the improvements in the accuracy of ADQuaTe2 using datasets with some ground truth data (i.e., a set of known faults) from the UCI repository [11]. 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.

II. Proposed Approach

Figure 1 shows an overview of the proposed approach. The input is in the form of data records and the output consists of a report showing groups of suspicious records accompanied with an explanation of the violated constraints. There are five components, namely, data preparation, constraint discovery, fault detection, fault interpretation, and fault inspection. The first four components were proposed in our earlier work [4] and are briefly described in Sections II-A to II-D. Fault inspection is a new component that we added to take feedback from the domain expert and is described in Section II-E. Sections II-F to II-H describe how the constraint discovery and fault detection components are extended in ADQuaTe2.

A. Data Preparation

This component prepares the raw data by transforming it into a form suitable for analysis. Typical machine learning algorithms cannot be directly applied to certain data types, such as categorical. Numeric datatypes can have different ranges. The data attributes need to be preprocessed based on their type and values. We used the one-hot encoding [12] method for preprocessing the categorical attributes and the standardization [13] method for the numeric attributes.

B. Constraint Discovery

This component obtains a trained model that best represents the different types of constraints in the unlabeled data. We use an unsupervised deep neural network called autoencoder [5] that has been demonstrated to be effective for attribute representation learning [14]. 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 normalized average squared distance between the original data and its reconstruction [5]. The constraints represented in the trained model are in the form of complex equations that formulate the associations among data attributes. However, these constraints are not human-interpretable. Further steps are required to explain the identified constraints to the domain experts.

C. Fault Detection

This component detects suspicious records that do not conform to the constraints represented by the trained model. Each record is assigned a suspiciousness score (*s*-score), which is equal to the reconstruction error of the record. Records whose *s*-score is greater than a threshold are flagged as suspicious.

It may be too time-consuming for a domain expert to inspect a large number of suspicious records. Thus, we group the suspicious records based on their similarity into a small number of groups. We use a clustering approach called Self Organizing Map (SOM) [6], which preserves the relationships among attributes [15] in its clusters. The outputs of this component are groups of suspicious records. To inspect the groups, a domain expert needs additional information about the reason behind the invalidity of each group.

D. Fault Interpretation

This component helps a domain expert interpret each suspicious group by generating visualization plots of two types, namely, *s*-score per attribute and decision tree. The trained autoencoder model calculates the *s*-score per attribute. The higher the value of *s*-score, the more likely is the attribute to contribute to the invalidity of the group. For each group we plot the *s*-score values for all the attributes in the group.

We use a *decision tree* [16] based technique called random forest [7] classifier to determine the constraints that are violated by each group of suspicious records. In this decision tree 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 path described by the attribute values from the root to that leaf node.

A domain expert decides whether or not a suspicious group is actually faulty by analyzing the constraints represented in the decision trees and by inspecting the values of the attributes that are major contributors to the *s*-scores.

E. Fault Inspection

This component takes domain expert feedback through a web-based user interface that uses check boxes for the expert to flag as faulty the groups that are actually faulty. ADQuaTe2 labels the training data records using this feedback. The labels are used by the extended constraint discovery and fault detection components to improve the accuracy of the approach.

F. Update Training Dataset for Retraining

We add a new 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 expert feedback. Records marked suspicious by the fault detection component are labeled 0.5, out of which those marked as actually faulty by the domain expert are labeled 1 and those not marked by the domain expert are labeled -1. Records that are not reported by ADQuaTe as suspicious remain 0. The updated dataset is used to retrain the autoencoder.

G. Extension to Constraint Discovery

We extend this component by (1) redefining the reconstruction error of autoencoder based on the label value, and (2) initializing the network parameters for the retraining phase.

1) Redefine reconstruction error based on label value: In addition to the existing inputs (i.e., the attributes $a_{i1} \ldots a_{id}$ for record x_i), we also provide the new label (l_i) to the autoencoder. Unlike the other attributes, the labels are not preprocessed. Figure 2 shows the new attribute and its corresponding output in the interactive autoencoder structure.

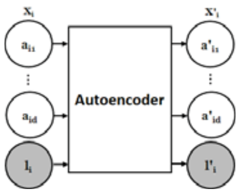


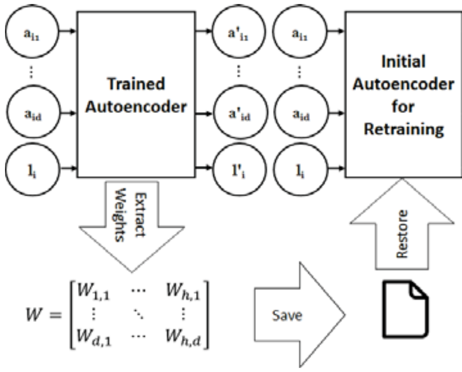
Fig. 2: Interactive Autoencoder

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.

$$RE = \frac{1}{N}\sum_{i=1}^{N}((l'_i - l_i)^2 + \sum_{j=1}^{d}(a'_{ij} - a_{ij})^2) \tag{1}$$

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.

2) Initialize autoencoder parameters for retraining: The original autoencoder uses randomly initialized network parameters, such as weights [10]. 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 accuracy of the network is at least as much as the accuracy of the previous trained network. Figure 3 shows how the network weights are extracted and restored for the retraining phase. After the network is trained using the input dataset, the network parameters are extracted and saved in an intermediate file. These parameters are restored to initialize the autoencoder for the retraining phase.



a_i1
a_id
l_i
Trained Autoencoder
a'_i1
a'_id
l'_i
a_i1
a_id
l_i
Initial Autoencoder for Retraining
a'_i1
a'_id
l'_i
Extract Weights
Restore
$W = \begin{bmatrix} W_{1,1} & \cdots & W_{h,1} \\ \vdots & \ddots & \vdots \\ W_{d,1} & \cdots & W_{h,d} \end{bmatrix}$
Save

Fig. 3: Initialize Autoencoder for Retraining

H. Extension to Fault Detection

We extend this component by (1) redefining *s*-score based on the label values and (2) tuning the threshold value.

1) Redefine s-score based on record labels: As described in Section II-C, the *s*-score of a data record is equal to the reconstruction error of that record. ADQuaTe2 calculates the *s*-score based the reconstruction error and the labels obtained using domain expert feedback (equation 2). The objective is to ensure that all the records detected in the retraining phase are either faulty (i.e., records that were flagged by the expert in previous executions) or unknown (i.e., records that have not been reported by ADQuaTe2 in previous executions).

$$s\_score(x) = RE(x) + l(x) \tag{2}$$

where $RE(x)$ is the reconstruction error of record x 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 will be in the range [1, 2], [-1, 0], and [0, 1] respectively. By setting the threshold to a value greater than zero, we ensure that ADQuaTe2 will not report as suspicious any valid record (i.e., records that ADQuaTE2 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 suspicious in subsequent executions, but with a higher s-score value (in the range [1,2]).

2) 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 $b \leq c$, there is no overlap between the s-scores of faulty and valid records. Equation 3 describes how ADQuaTe2 tunes *T* for the retraining phase to make sure that no valid records are displayed as suspicious in the next iteration.

$$T = \begin{cases} min(c,\ P) & if\ P > 0, \\ min(c,\ p(s\text{-}scores, 90)) & otherwise \end{cases} \tag{3}$$

where P is the percentage of previously known faults for the input dataset and p is the percentile function, which 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 faults in the dataset, ADQuaTe2 detects at least $P\%$ of records as suspicious to ensure that all the previously detected faults are reported in the current iteration. We set the threshold to 10% for datasets with no known faults because the average percentage of known faults in the datasets used by this study as well as 26 other datasets [17] from the UCI repository is equal to 10%. Domain experts can change this value based on the knowledge of the validity of their datasets.

III. Evaluation

In our earlier work, we evaluated the constraint discovery, fault detection, and fault interpretation effectiveness of our approach using real-world data from health and plant domains. We used seven datasets created using multiple table joins in a health data warehouse [18] and one dataset from a plant diagnosis database [19]. Rows 1 to 8 in Table I show the characteristics of these datasets. We demonstrated that ADQuaTe could detect between 96.14% and 100% of faults that were previously detected by Achilles [20] and Murdock [21] tools for the health datasets. In the worst case, ADQuaTe could not detect 3.86% of faults that were previously detected by the existing tools. This indicates that the autoencoder could not discover all of the associations among the attributes. ADQuaTe could detect between 33.33% to 35.63% actual faults that were not previously detected. 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. Between 53.33% to 100.00% of the visualization plots correctly explained the reasons behind invalidity of the records.

In this paper, we evaluate the improvements in the accuracy of the approach using datasets with ground truth data from the UCI repository [11]. Rows 9 to 15 in Table I shows the characteristics of these datasets. We also evaluate the time it takes for the overall approach for all datasets.

A. Goal 1: Evaluate the accuracy improvements.

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

RQ1.a: Does the number of correctly detected faults increase after retraining the machine learning model with the help of feedback from domain expert?

Given *E*, the set of faulty records detected by an existing data quality test approach, *A* the set of faulty records detected by ADQuaTe2, and *AF* the set of faulty 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 faulty.

$$TPR = \frac{|AF|}{|A|} \tag{4}$$

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.

$$TPGR = (\frac{TPR_{NR}}{TPR_1})^{\frac{1}{NR}} - 1 \tag{5}$$

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

RQ1.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 faults (UD) plus percentage of actual faulty records detected in previous runs that could not be detected in the current run.

$$FNR = \frac{|AF_{old} - AF_{new}|}{|AF_{old}|} + \ UD \tag{6}$$

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.

$$FNGR = \left(\frac{FNR_{NR}}{FNR_1}\right)^{\frac{1}{NR}} - 1 \tag{7}$$

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

We used the UCI ML datasets with a set of previously known faults. We implemented a script that automatically executes ADQuaTe2 against an input dataset and detects suspicious records. The script updates the values of the label attribute from 0.5 to 1 for the suspicious records that are

TABLE I: Datasets from Real-world Applications and UCI ML Repository [11]

ID	Name	Domain	#Records	#Attributes	Known Faulty Records (%)
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	*Lymphography*	Oncology	148	18	4.05
10	*Glass_identification*	Criminology	214	10	4.20
11	*Vertebral_column*	Biomedical	240	6	12.50
12	*Heart_disease*	Health	267	75	20.60
13	*Ecoli*	Biology	336	8	2.67
14	*Ionosphere*	Radar	351	34	35.89
15	*Breast_cancer*	Oncology	699	10	34.50

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 fault detection component (Section II-E) and measures the accuracy of ADQuaTe2 based on the metrics described in this section. The whole process is performed 10 times.

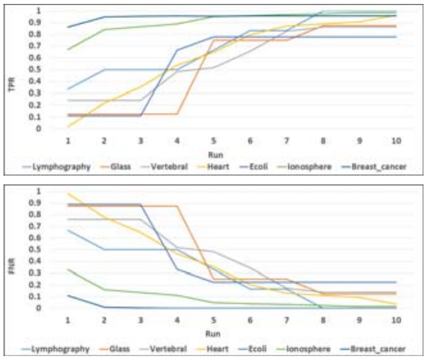


Fig. 4: Improvement in TPR and FNR for UCI Datasets

Figure 4 shows how the true positive rate increases and false negative rate decreases over time during the retraining process for the datasets under test. Table II shows positive values for TPGR for all of these datasets, which demonstrates that the fault detection effectiveness of ADQuaTe2 improves after retraining the machine learning model. Table II shows negative values for FNGR for all of these datasets, which demonstrates that the fault detection effectiveness of ADQuaTe2 improves after retraining the machine learning model.

TABLE II: True Positive and False Negative Growth Rate for UCI Datasets for 10 Runs

Dataset ID	TPGR	FNGR
9	0.127	-0.16
10	0.21	-0.17
11	0.13	-0.16
12	0.48	-0.27
13	0.22	-0.12
14	0.038	-0.26
15	0.01	-0.20

Based on Figure 4, the TPR and FNR are almost stabilized after eight iterations. This shows that the payoff is not worth the cost of running ADQuaTe2 after eight iterations. This number was applicable to the UCI datasets that we used but cannot be generalized to other datasets. Domain experts can change the number of iterations based on their observation on the improvement of the accuracy for their datasets.

B. Goal 2: Evaluate the performance of overall approach.

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

RQ2: How long does it take to execute ADQuaTe2 against an input dataset?

TT: The total time taken to perform the automated steps of data preparation, constraint discovery, fault detection, and fault interpretation. Time spent by domain experts is not included because different experts inspect faults in different ways.

Table III shows values of *TT* for all the datasets under test for one execution of ADQuaTe2. The datasets in this table are sorted based on their $size = NRe * NAt$, where *NRe* is the number of records and *NAt* is the number of attributes. It took between 0.138 to 21 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 size. This shows that dataset characteristics other than size, such as data types and sparseness may have also affected the results. The analysis of other effective factors on the performance of ADQuaTe2 is the subject of our future work.

TABLE III: Total Time (TT) for All Datasets

Dataset ID	Size	TT (min)
11	1,440	0.146
10	2,140	0.138
9	2,664	0.160
13	2,688	0.160
1	5,634	3.000
15	6,990	0.210
14	11.934	0.333
12	20,025	0.208
2	376,660	15.000
5	900,000	5.000
4	1,900,000	20.000
3	2,000,000	21.000
7	3,000,000	7.000
6	4,200,000	6.000
8	7,000,000	8.000

IV. Related Work

Data management systems provide syntactic constraint violation detection at the schema design level. Association Rule Mining [22] investigates semantic constraints from data using association rules that describe co-occurrence of attribute values. However, this is an inefficient approach as it makes one pass through the dataset for every combination of attribute values. Moreover, too many different association rules can be derived from even a tiny dataset, most of which are non-interesting to domain experts [23].

Machine learning-based approaches can detect the outliers [3] that violate semantic constraints in the data. Depending on the availability of labeled data, these techniques are classified as supervised (e.g., Naive Bayesian [24], Support Vector Machine (SVM) [25], and Artificial Neural Network (ANN) [26]), unsupervised (e.g., OC-SVM [27]), and semi-supervised (e.g., clustering and Representational Learning (RL) [8]). Supervised techniques require domain experts to manually label training data, which is not scalable for big datasets. The training phase is restricted to a set of labeled data that are biased towards the domain expert's knowledge. The Naive Bayesian approach assumes a strong independence between the record attributes and cannot discover constraints that involve relationships among multiple attributes. SVM trains a hyperplane in the attribute space that best divides a labeled dataset into valid/invalid classes but the trained hyperplane is an equation over data attributes that is not human interpretable. ANN trains a network of information processing units that best classifies the records as valid/invalid but the network is not human interpretable. Semi-supervised techniques require providing a clean dataset for the training phase. These techniques are also biased towards the definition of valid records by the experts. The hyperplane trained by an OC-SVM captures regions where the probability density of the valid data lives, but is not human interpretable. Distance-based clustering algorithms cannot derive relationships among attributes in their clusters [28]. Moreover, the clusters do not explain what constraints were violated. The representations used in Representational Learning also do not explain how the records violate constraints and the representations themselves are not human interpretable.

V. Conclusions

We extended our previous approach by incorporating the expert feedback to minimize the false alarms and generate more precise results. ADQuaTe2 allows experts to inspect the reported records and feeds the constraint discovery and fault detection components with the information received from the expert. Our approach discovered new constraints in the attributes of health and plant data that were missed by domain experts and detected faults that were not previously detected by the existing tools. The true positive and false negative rates improved after incorporating the ground truth knowledge and retraining the learning model. We will extend ADQuaTe2 to support constraint discovery over multiple records using time series analysis techniques.

Acknowledgment

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.

References

[1] "Patient Safety Errors are Common with Electronic Health Record Use," https://healthitanalytics.com/news/patient-safety-errors-are-common-with-electronic-health-record-use (Accessed 2019-08-17).

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

[3] C. C. Aggarwal, "An introduction to outlier analysis," in *Outlier Analysis*, 2017, pp. 1–34.

[4] H. Homayouni, S. Ghosh, and I. Ray, "ADQuaTe: An Automated Data Quality Test Approach for Constraint Discovery and Fault Detection," in *20th IEEE IRI*, 2019, pp. 61–68.

[5] C. Zhou and R. C. Paffenroth, "Anomaly Detection with Robust Deep Autoencoders," in *23rd ACM KDD*, 2017, pp. 665–674.

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

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

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

[9] R. M. Konijn and W. Kowalczyk, "An Interactive Approach to Outlier Detection," in *5th RSKT*, 2010, vol. 6401, pp. 379–385.

[10] C. M. Bishop and P. o. N. C. C. M. Bishop, *Neural Networks for Pattern Recognition*. Clarendon Press, 1995.

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

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

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

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

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

[16] 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.

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

[18] "HDC," http://www.ucdenver.edu/about/departments/healthdatacompass/ (Accessed 2019-02-12).

[19] "CSU Plant Diagnostic Clinic," https://plantclinic.agsci.colostate.edu/ (Accessed 2019-03-26).

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

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

[22] R. Agrawal, T. Imieliński, and A. Swami, "Mining Association Rules Between Sets of Items in Large Databases," *SIGMOD Rec.*, vol. 22, no. 2, pp. 207–216, 1993.

[23] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, "Chapter 3 - output: Knowledge representation," in *Data Mining*, 2017, pp. 67–89.

[24] 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.

[25] C. Cortes and V. Vapnik, "Support-vector Networks," *ML*, vol. 20, no. 3, pp. 273–297, 1995.

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

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

[28] G. Gan, C. Ma, and J. Wu, *Data Clustering: Theory, Algorithms, and Applications*. SIAM, 2007.