

Automating Data Quality Validation for Dynamic Data Ingestion

Sergey Redyuk, Zoi Kaoudi, Volker Markl

Technische Universität Berlin

[sergey.redyuk,zoi.kaoudi,volker.markl]@tu-berlin.de

Sebastian Schelter

University of Amsterdam

s.schelter@uva.nl

ABSTRACT

Data quality validation is a crucial step in modern data-driven applications. Errors in the data lead to unexpected behavior of production pipelines and downstream services, such as deployed ML models or search engines. Typically, unforeseen data quality issues are handled via manual and tedious debugging processes in a reactive manner. The problem becomes more challenging in scenarios where large growing datasets have to be periodically ingested into non-relational stores such as data lakes. This is even worse when the characteristics of the data change over time, and domain expertise to define data quality constraints is lacking.

We propose a data-centric approach to automate data quality validation in such scenarios. In contrast to existing solutions, our approach does not require domain experts to define rules and constraints or provide labeled examples, and self-adapts to temporal changes in the data characteristics. We compute a set of descriptive statistics of new data batches to ingest, and use a machine learning-based novelty detection method to monitor data quality and identify deviations from commonly observed data characteristics. We evaluate our approach against several baselines on five real-world datasets, on both real and synthetically generated errors. We show that our approach detects unspecified errors in many cases, outperforms other automated solutions in terms of predictive performance, and reaches the quality of baselines that are hand-tuned using domain expertise.

1 INTRODUCTION

Data-driven decision making is becoming the norm in modern enterprises and organizations, and requires maintaining and regularly updating large datasets, often collected in non-relational stores such as data lakes. A critical step in these scenarios is data quality validation, as the quality of the derived insights and decisions crucially depends on the quality of the collected data [42]. Incorrect or missing data can lead to wrong business decisions and problems in downstream data consumers, such as machine learning (ML) models or search engines [1, 17, 43], and even crash systems, e.g., due to null-pointers originating from missing data. Common sources of errors are bugs in external data sources and data preprocessing code (e.g., when a data engineer accidentally changes a time measurement from seconds to milliseconds in a data-producing pipeline). Such errors often corrupt large parts of the data to ingest and can immediately lead to devastating consequences, e.g., wrong predictions of ML models that consume the data [37]. In this work, we focus on automating the detection of such data quality issues.

We address the following real-world example scenario. Consider a data engineering team at a retail company maintains a search engine for products. To keep the search engine up-to-date,

it deploys a pipeline that regularly ingests and indexes external product data from various heterogeneous sources, such as web crawls, log files, key-value stores, or upstream data pipelines. If a data source introduces errors in the data to ingest, such as missing values or wrong encoding of strings, then the products will not be indexed correctly or, even worse, cause the ingestion process to crash. Such data issues are typically handled reactively: the engineering team discovers data issues via alerts from devops engineers, bug reports, and customer reviews. The data is then manually fixed and back-filled. Handwritten code is added to the data pipeline in retrospect to catch the observed type of errors in the future [43].

In this paper, we propose an approach to automate data quality validation in scenarios where large partitions of a growing dataset have to be regularly ingested into a common data store such as a data lake. While relational databases enforce a schema and integrity constraints for their data [13], many modern applications rely on non-relational data stores. Pipelines that do not specify a particular schema or constraints on the data are often much cheaper to operate in cloud environments (e.g., using S3 as a distributed filesystem for storing the data partitions and Apache Spark for processing them).

In contrast to existing work on fine-grained error detection [1, 17, 27, 29, 36, 47], we focus on scenarios where systems regularly ingest batches of external data, and data errors corrupt a large fraction of the batch [42] (Section 3). In the aforementioned retail example, a few missing product reviews in a partition might not cause issues in the downstream systems, as they are programmed to handle that (e.g., by using missing value imputation strategies). However, an unusually high fraction of missing values in the review description is an indicator of a severe problem in one of the external data sources.

We automate the detection of six types of errors (explicit and implicit missing values, numeric anomalies, typos, swapped fields for numeric and textual attributes) as follows: we leverage previously ingested data batches as “positive” examples of “acceptable” data and use a machine learning approach to identify new batches that significantly deviate from the previously observed data. Specifically, we compute a set of descriptive statistics over the ingested data and train a novelty detection ML model [30, 31] to learn the characteristics of the “acceptable” data. We apply the ML model on new data batches to ingest, in order to identify potentially erroneous data batches that significantly differ from previously observed data (Section 4).

Our approach provides several advantages over existing work. First, it does not require domain experts to design and maintain large numbers of rules [3, 20, 43]. Devising such rules and constraints is a very tedious and expensive process as the datasets found in enterprises are typically large and messy [45], especially if they originate from the integration of different external data sources. Secondly, our approach is computationally efficient as the descriptive statistics we apply can be computed in a single

pass over the data. Our novelty detection model has a low number of parameters to optimize. Finally, our automated approach performs well in cases where the data characteristics change over time, in contrast to rule- and constraint-based approaches [20, 43] that require a manual redefinition of rules and constraints.

We evaluate our approach by comparing its predictive performance to automated and hand-tuned variants of the following state-of-the-art solutions: Tensorflow Data Validation [6], Deequ [43], and statistical testing [32, 41]. Then, we evaluate the sensitivity of our approach towards six types of errors (explicit and implicit missing values, numeric anomalies, typos, swapped fields on numeric and textual attributes) and the predictive performance under various error magnitudes (1, 5, 10, 20, . . . , 80%) in a controlled environment for datasets with synthetically generated errors. Finally, we evaluate the detection quality of our approach over time, as (a) the size of the training set for the novelty detection algorithm grows continuously, and (b) its data characteristics change over time. In summary, we make the following contributions:

- We propose an approach to automate data quality validation for data that is periodically ingested into non-relational stores. In contrast to existing solutions, our approach does not require domain experts to define rules or labeled examples, and self-adapts to temporal changes in the data characteristics (Sections 3 & 4);
- We discuss how to apply our approach efficiently via a novelty-detection ML model trained on data quality metrics of the data (Section 4);
- We evaluate our approach against existing baselines on five real-world datasets with real and synthetically generated errors. We find that our approach detects the unspecified errors in many cases under varying error magnitudes, outperforms other automated solutions in terms of predictive performance, and reaches the ROC AUC score of baselines hand-tuned with domain expertise (Section 5).

2 BACKGROUND

In the context of this work, we understand data quality validation as the process of checking that the input data meet the needs of a data-driven application or its underlying business process, where these specific needs are either formulated explicitly with the data standards and policies or assumed implicitly by the application logic. The concept of data quality is broadly defined as a measure of the fitness of the data to their intended uses and purposes [11]. To identify how well the data fit for the intended purpose, the vast body of knowledge [5] suggests several data quality dimensions, such as data *accuracy* (the degree to which the data correctly represent the real-world entity it models), *completeness* (the degree to which the data contain the necessary attributes to model the entity), *validity* (the degree to which the data are stored or represented in a format that is consistent with the domain of values), and others. In practice, data quality is assessed with a set of quantitative metrics that are associated with the aforementioned data quality dimensions. In this section, we briefly introduce the data quality metrics that we leverage in our approach and the machine learning-related background for novelty detection.

Data quality metrics. We consider several quantitative statistics that can be used to identify data quality issues [18]: (i) *completeness* - the ratio of non-missing values to the number of

records in the data; (ii) *the number of distinct values*; (iii) statistics for numeric data types, such as *maximum*, *mean*, *minimum*, and *standard deviation*, (iv) *the ratio of occurrence for the most frequent value*, etc. These statistics are commonly used in database engines, for data profiling and data quality validation [18] to summarize data of interest and often act as a proxy for the state of data quality. Furthermore, most of the statistics can be cheaply computed in a single scan over the data, except for the number of distinct values and the ratio of the most frequent value, which are typically approximated with the hyperloglog and the count-min sketches respectively [8, 12].

Novelty Detection. Novelty detection is a machine learning technique that aims to identify new patterns and signals that were not present in the training data [30]. It is closely related to anomaly detection as both techniques look for patterns in data that do not conform to the expected behavior [7]. The difference is that anomaly detection assumes that outliers are already present in the training data. In contrast, novelty detection is designed for cases where we only have access to “positive” examples.

Novelty detection is a form of one-class classification [46] (due to the absence of negative examples). Novelty detection algorithms model the data and check whether previously unseen data points resemble the characteristics of the modeled data (i.e., inliers) or deviate from the expected behavior (i.e., outliers). The decision whether or not a new object (i.e., data point) is an outlier against a set of known objects follows the continuity assumption (i.e., two data points that are close in the feature space represent two objects with the resemblance in real life) and usually focuses on distance measures. Common example algorithms in this area are one-class SVMs [44] and isolation forests [26]. For an in-depth overview of the one-class classification problem and novelty detection algorithms, we recommend the reader to refer to Tax [46] and Chandola et al. [7].

3 PROBLEM STATEMENT

In this section, we introduce the problem and its formal definition.

Overview. We address the problem of automating the validation of data quality on dynamic data without relying on domain expertise (e.g., manually specified rules and labeled erroneous data records). As outlined in the running example, we focus on scenarios where data pipelines regularly ingest large batches of potentially erroneous external data and face errors that corrupt a large fraction of the batch.

State-of-the-art solutions in data quality validation typically require domain knowledge to specify explicit rules, constraints, patterns, or labeled examples to verify data quality [6, 18, 20, 43]. They, however, fall short in several cases: (i) incomplete domain knowledge (i.e., when data depict complex processes that even domain experts cannot fully comprehend or when the domain expert is unavailable at the given time), the solutions mentioned above might perform poorly both due to false alarms and missed errors as the specified set of rules or labeled examples are insufficient to capture potential errors; (ii) manual monitoring of data pipelines to detect data quality issues or deployment of staging environments for software testing are often too costly or time-consuming, and are only conducted reactively; (iii) the characteristics of the data might slowly change over time, which implies that manually specified rules have to be constantly adapted and maintained.

These challenges motivate an automatic approach to data quality validation that does not rely on manually specified rules or labeled examples and self-adapts to changes in data characteristics.

Assumptions. For the given use case of the regular ingestion of large batches of a growing dataset, we consider previously observed and successfully ingested data partitions to be of “acceptable” data quality. This assumption is based on our experience with real-world use cases: It is common in production to define principal business and operational performance indicators and monitor them carefully to evaluate business outcomes. For our retail company running example, products that are placed in the wrong category due to various errors lead to negative customer reports or low service ratings, or via incident reporting and tracking systems. This negative feedback serves as a proxy that affects key performance indicators and catches the attention of the responsible staff at some point in time. This, in turn, triggers retrospective analysis. If devastating errors would have occurred in the previously observed data partitions, they would have been detected and fixed after a given time. If errors do not trigger a negative response from the devops engineers or the business after some period of time, we assume that the downstream task is robust to them. Furthermore, existing error detection or data quality validation methods require domain expertise. We focus on real-world scenarios where domain expertise is not available that, in turn, render the majority of data quality monitoring tools inapplicable.

Formal problem statement. Given a structured dataset D of chronologically ordered partitions d_1, \dots, d_{t-1} , each having domain $A = A_1, \dots, A_M$, we have to predict upon the arrival of a new partition d_t whether this partition is of acceptable quality or it is potentially corrupted w.r.t. a set of data quality metrics $Q = Q_1, \dots, Q_G$. We map this problem to a “one-class classification” problem [46] where every partition d_i is represented by a feature vector $\mathbf{x}_{d_i} = (x_1, \dots, x_G) \in \mathbb{R}^G$ of the data quality metrics Q that are computed on every attribute A_i of that partition and a boolean label y_{d_i} , which denotes whether the quality of the batch is acceptable or not. However, we only have access to positive examples during training (hence the term “one class” classification). The classification task is to decide whether a future batch d_t can be considered of acceptable quality (i.e., represents an inlier) or deviates from the state of data quality of the previously observed data batches (i.e., represents an outlier). The main challenge is to model the “acceptable” data in an automated manner, without external specification of the domain or examples of “erroneous” data that have insufficient data quality.

4 APPROACH

Next, we discuss our approach for automating data quality validation of newly observed data batches based on the problem definition we presented in the previous section.

Overview. Figure 1 illustrates our approach: for every observed partition d_1, \dots, d_{t-1} , we model the features \mathbf{x}_{d_i} via a set of descriptive statistics computed from the partition ①. We train a novelty detection model [38] on the resulting feature vectors that learns the characteristics of “acceptable” data ②. In order to check a new data batch d_t , we compute its feature vector \mathbf{x}_t via the chosen descriptive statistics ③. Next, we apply the novelty detection model to label the new batch as acceptable or erroneous based on the learned decision boundaries of the model ④. With every new data partition d_t , we re-train the novelty detection model as the training set grows with t . Our method can be integrated into

data pipelines to raise alerts about potential degradation of data quality automatically. Note that our approach does not rely on domain expertise expressed in the form of rules, constraints, or labeled data. Still, it remains valid in cases where the task definition is relaxed (e.g., domain knowledge is partially available or some error types are expected).

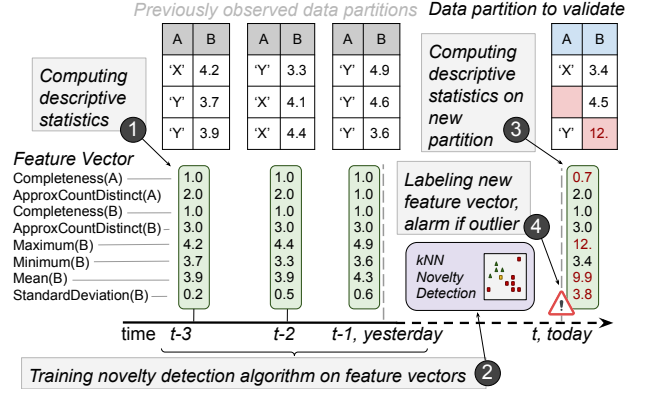


Figure 1: Overview of the approach: for every observed partition (gray tables), we compute a set of descriptive statistics as a feature vector (green, Step 1). We train a novelty detection model that learns the characteristics of acceptable data (Step 2). For the upcoming data partition (blue table), we compute its feature vector (Step 3) and let the model decide whether it is similar to the previously observed data partitions or not (Step 4). In this example, a missing value in column “A” and a numerical outlier in column “B” (red) affect the completeness metric and numeric statistics of the feature vector \mathbf{x}_t . That, in turn, raises an alert.

Descriptive statistics as features. For every attribute A_j of the partition d_i , we compute several quantitative measures that correspond to the underlying data quality metrics (see Section 2):

- *Completeness* - the ratio of not-NULL values;
- *Approximate count of distinctive values* - the hyperloglog [12] approximation of the number of distinctive values;
- *Ratio of the most frequent value* - the count sketch [8] approximation of the number of occurrences for the most frequently repeated value, normalized by the batch size;
- *Maximum, mean, minimum, and standard deviation* for numeric data types;
- *Index of peculiarity* [33] for textual data. Index of peculiarity is based on the bi- and trigram tables of a textual attribute and reflects the likelihood of the hypothesis that trigrams in a given word are produced from the same data source that produced the trigram table. This index is originally applied for detection of typographical errors and facilitates detection of typos in text or a “peculiar” occurrence of symbols in words.

$$I(T) = \frac{1}{2} (\log n(xy) + \log n(yz)) - \log n(xyz) \quad (1)$$

Equation 1 represents the index of peculiarity for a trigram $T = (xyz)$, where $n(\cdot)$ denotes the number of occurrences for a selected bi- or trigram in a textual attribute. Index of peculiarity for a sentence is the root-mean-square aggregation of indices for each trigram that this sentence contains.

Algorithm 1: Pseudocode of our approach.

Input: t , query raw data partition; k , the number of neighbors;
 X , descriptive statistics for previously ingested data partitions;
contamination, the proportion of outliers in X ;
dist, distance measure (e.g., Euclidean, Manhattan);
agg, distance aggregation strategy for k nearest data points.
Output: **label**, query data point t is inlier/outlier

```
1 Initialize array statistics; array distances;  
2   list num_met of metrics for numeric data types;  
3   list gen_met of metrics for other data types.  
4 foreach attribute  $A \in t$  do  
5   metrics = num_met if type( $A$ ) is numeric else gen_met  
6   foreach metric  $\in$  metrics do statistics.append(metric(A))  
7 end  
8 foreach  $x \in X$ , tree = BallTree(X, dist) do  
9   /* .getDist( $x, k$ ) returns distances to  $k$  nearest  
   neighbors of  $x$ ; agg(array) is an aggregation  
   function such as mean, median, or max */  
10  distances.append(agg(tree.getDist( $x, k$ )))  
11 end  
12 /* percentile( $x, q$ ) computes  $q$ -th percentile of  $x$  */  
13 threshold = percentile(distances, (1 - contamination))  
14 /* outlier if aggregated distance from  $t$  to  $k$  nearest  
   neighbors exceeds threshold, else inlier */  
15 return agg(tree.getDist(statistics,  $k$ )) < threshold
```

We concatenate attribute-level statistics into a univariate numeric vector. Depending on the number of attributes and their data types, the feature vector varies in length from one dataset to another, where the length remains constant for partitions of the same dataset. We normalize the resulting feature vectors to a scale of 0 to 1. We chose these statistics based on two criteria: (a) low computational complexity and (b) mapping to the error types that often occur in real-world scenarios [47]. For a particular error type that we investigate, we consider statistics that act as proxies for this error type more descriptive than others in detecting data quality degradation. By a proxy we mean a quantitative measure that is expected to change when a particular error occurs (e.g., numeric outliers are likely to affect the statistical distribution of the attribute [18]). There is no single metric that is more descriptive than others for all the given error types. Preliminary results show that specifying only the descriptive statistics that we expect to be changed when an error occurs increases performance of our approach. This happens because, in low-dimensional feature spaces, data points are more distinct and distance-based methods perform better. However, assuming “zero domain knowledge” and unknown error types, we cannot control the choice of descriptive statistics in practice and, thus, train our approach on all statistics. As discussed in Section 2, most of these statistics can be computed in a single scan over the data. Furthermore, we treat the sequence of feature vectors that we collect over time (i.e., $t_{start}, \dots, t - 1$) as separate data points in the training set. Note that this modeling decision does not preserve the order of these feature vectors.

Choice of the novelty detection algorithm. Given the nature of the challenge at hand, i.e., “zero domain knowledge” or unknown error types, only positive examples are available for training. We thus choose one-class classification algorithms (i.e., novelty detection, see Section 2) as the main candidates for our approach. In this work, we considered several candidates for the novelty detection (ND) algorithm: Angle-based Outlier Detector

(ABOD), Feature Bagging ensemble for the Local Outlier Factor (FBLOF), Histogram-base Outlier Detection (HBOS), Isolation Forest, and the K Nearest Neighbors algorithm with both the maximum and the mean distance aggregation scheme (KNN and Average KNN, respectively) [30, 31]. To choose one particular ND algorithm for our approach, we conduct preliminary experiments on one dataset (Amazon Review, monthly data partition) and three types of errors (explicit and implicit missing values on all attributes, numeric anomalies on the attribute “overall”) with 30% of synthetically introduced errors per data batch, in order to determine which algorithm yields better predictive performance on the one-class classification task (for more details, see Section 5). We deliberately chose one dataset and a subset of error types under investigation to avoid overfitting and the selection bias for the evaluation procedure. Table 1 depicts the predictive performance metrics (ROC AUC score [22]) for all the ND candidates, as well as the break-down of the false positive and false negative results. We report the ROC AUC measure as it takes into account both the type-I and type-II errors. Furthermore, it is insensitive to imbalanced datasets and preferred in practice to other performance metrics such as accuracy or F1 score. In our preliminary experiments, we computed other performance metrics alongside the ROC AUC score. We noticed that, since our evaluation scenario introduces a balanced case where a negative counterpart exists for every positive example, accuracy, F1 and ROC AUC scores report similar values. Based on the preliminary results, we chose the k -Nearest Neighbor algorithm with the mean aggregation scheme [38]. This algorithm consistently outperformed other ND candidates on all three error types and produced no false positive results, meaning that no erroneous data batches were labeled as “acceptable”. The second best-performing candidate is the Angle-Based Outlier Detection method [23] that yielded comparable predictive performance yet took an order of magnitude longer to train the model and infer the labels.

Nearest-neighbor-based novelty detection. For every data point in the feature space, the k -Nearest Neighbor (kNN) algorithm calculates the average distance to its k nearest neighbors and learns a threshold to decide what data points to consider inliers or outliers [2]. The kNN algorithm has a *contamination* hyperparameter that defines a ratio of data points that are assumed to be incorrectly labeled as inliers. Hence they are labeled as outliers in the training data. This scheme internally translates the one-class classification problem into a standard binary classification problem where the examples of both classes are present. The algorithm utilizes the Ball tree[35] space partitioning data structure - a binary tree where each node represents a multi-dimensional hypersphere (i.e., ball) of partitioned data points. This data structure provides properties that are useful for efficient k -nearest neighbor search. All data points in the training set are represented with distances to their k nearest neighbors. Depending on the design decision, these distances are aggregated into a single numeric value with one of the available aggregation strategies (e.g., mean, median, max). These numeric values are used to learn a decision boundary to differentiate inliers and outliers - a data point is considered an outlier if its aggregated distance to k nearest neighbors exceeds the learned threshold. The threshold is defined with the contamination hyperparameter c that is translated into the $(1 - c)$ th percentile of the array of aggregated distance for the whole training set. Figure 1 provides a pseudocode representation of the KNN algorithm.

Table 1: Results of the preliminary experiment on performance evaluation for 7 novelty detection algorithms. Three error types under investigation are explicit and implicit missing values, and numeric anomalies, depicted as “Explicit MV”, “Implicit MV”, and “Anomaly” respectively. We measure predictive performance with the ROC AUC score (AUC), as well as the number of true positive (TP), false positive (FP), false negative (FN), and true negative (TN) results, where FPs are associated with the misclassification rate and FNs - with the false alarm rate.

ND Algorithm	Error type	AUC	TP	FP	FN	TN
One-class SVM	Explicit MV	.9213	178	0	28	150
	Implicit MV	.9213	178	0	28	150
	Anomaly	.9691	178	0	11	167
ABOD	Explicit MV	.9382	178	0	22	156
	Implicit MV	.9382	178	0	22	156
	Anomaly	.9691	178	0	11	167
FBLOF	Explicit MV	.9353	178	0	23	155
	Implicit MV	.9382	178	0	22	156
	Anomaly	.9662	178	0	12	166
HBOS	Explicit MV	.5814	60	118	42	136
	Implicit MV	.5505	60	118	42	136
	Anomaly	.9297	176	2	23	155
Isolation Forest	Explicit MV	.7331	27	151	18	160
	Implicit MV	.5280	27	151	17	161
	Anomaly	.8764	146	32	12	166
KNN	Explicit MV	.9325	178	0	24	154
	Implicit MV	.9325	178	0	24	154
	Anomaly	.9662	178	0	12	166
Average KNN	Explicit MV	.9382	178	0	22	156
	Implicit MV	.9382	178	0	22	156
	Anomaly	.9719	178	0	10	168

Gu et al. [15] present an extensive statistical analysis of nearest neighbor algorithms and report that recent work on this family of methods reaches state-of-the-art performance on novelty detection tasks. Based on the preliminary experiment, we confirm that the kNN novelty detection method performs on par with other approaches or outperformed them, both in terms of the predictive performance and execution time.

Modeling decisions. Next, we discuss several modeling decisions for our kNN-based approach. We choose the Euclidean distance metric as the most commonly used distance measure for the \mathbb{R}^G feature space, and leverage the average distance to k neighbors as an aggregation strategy. Based on preliminary experiments, this decision led to consistently higher predictive performance compared to other settings. Alternative strategies are choosing the largest distance among k neighbors or computing the median. A systematic comparison of kNN algorithms with different distance measures revealed that both the “largest” and the “median” aggregation schemes happen to be less robust than averaging in our setting.

We set the number of neighbors k to aggregate the distance measure to a low factor of five. The variation of this parameter did not lead to significant changes in the predictive performance during the preliminary experiments. The kNN novelty detection algorithm is also parameterized with the contamination parameter [19]. This parameter defines a fraction of data points in the training set to be misclassified as “positive” examples and assumed to be outliers (i.e., false positives). We set the contamination parameter to 1% to keep the ratio of false positives minimal.

Table 2: Characteristics of the datasets. The abbreviations depict, in a direct order, the number of records in the dataset, the number of partitions, the total number of attributes, the average number of records in a data partition, the number of numeric, categorical, and textual attributes. We also report the real-world error types that two datasets with the ground truth, Flights and FBPosts, contain.

Dataset	Flights	FBPosts	Amazon	Retail	Drug
# records	147640	11157	1494070	541909	161297
# part./attr.	31/9	53/14	1665/9	305/8	3579/6
part. size	~2350	~105	~897	~1776	~45
N/C/T	1/4/0	4/3/2	2/1/4	2/5/1	2/2/1
Dataset	Flights		FBPosts		
Errors, %	explicit/implicit missing values, 8-38%		wrong encoding, 16%		
	incomplete datetime format, 95%		syntactic errors and translation, 18%		
	other syntactic/semantic errors, 60%				

We aim to minimize the number of data points in the training set that are considered to be falsely classified as “inliers”. We base this decision on our assumption that all the data partitions are of “acceptable” quality, and no misclassification occurs. Preliminary experiments showed that setting the contamination parameter to 1% leads, on average, to relatively higher predictive performance compared to other values (including 0). Note that automated hyperparameter tuning schemes are challenging in the case of one-class classification problems, as we do not have labels for both of the classes - acceptable and erroneous data.

Application to our example scenario. Based on the running example, imagine the engineering team to apply the proposed approach as a data quality monitoring tool to validate incoming data batches before running data preprocessing and indexing jobs. When a new data batch is examined and no alerts are raised, data pipelines work without any difference and run the downstream preprocessing and indexing job. In case an alert is raised, the team starts a debugging process and applies further error detection and correction strategies. If the method caught the erroneous data batch correctly, the team fixes it and released the quarantined batch back to the pipeline. In the case of false alarms, the data is returned without alterations. The critical point is when the erroneous data batch passes data quality checks and goes further to the downstream pipeline without the errors being fixed (i.e., false positives). In this case, system crashes and degradation in the predictive performance of the underlying ML model might occur.

5 EVALUATION

In this section, we introduce our experimental setup and discuss datasets and metrics for our evaluation. We conduct several experiments. First, we compare the predictive performance of our approach to automated and hand-tuned variants of the following state-of-the-art solutions: Tensorflow Data Validation [6], Deequ [43], and statistical testing [32, 41]. Then, we evaluate the sensitivity of our approach towards six types of errors (explicit and implicit missing values, numeric anomalies, typos, swapped fields on numeric and textual attributes) and the predictive performance under various error magnitudes (1, 5, 10, 20, . . . , 80%)

in a controlled environment for datasets with synthetically generated errors. Finally, we evaluate the detection quality of our approach over time, as (a) the size of the training set for the novelty detection algorithm grows continuously, and (b) its data characteristics change over time.

5.1 Experimental Setup

We evaluate our proposed approach as follows. We experiment with a relational dataset that is partitioned by a chosen temporal attribute (e.g., a creation timestamp for every record). This allows us to simulate our target scenario of the daily ingestion of new data batches in a data pipeline. For every data point that corresponds to a particular day t , we use the previously observed partitions from timestamp 0 to $t - 1$ as training data for our approach. Then, we take both the partition d_t and a corrupted version \hat{d}_t as a counterpart, pass it to our model, and have it predict whether the partition is of acceptable data quality or not. Data partitions of acceptable quality are those that do not affect KPIs and usually depend on the downstream ML task. However, to decouple our experimental evaluation from the underlying ML task, we consider partitions of acceptable data quality the ones that do not contain any errors. We apply standard binary classification metrics such as the area under the ROC curve (ROC AUC score [22]) to evaluate how well the approach performs. We also report confusion matrices to analyze misclassification and false alarm rates.

Datasets. We experiment on five publicly available real-world datasets from different application domains. For two of them, we have access to both the erroneous and the cleaned versions of the data [25]. The other three do not contain any errors, we thus generate the errors synthetically [9, 14, 16]. For details, see Table 2.

Datasets with ground-truth errors. The Flights¹ dataset [25] contains flight status data that is aggregated from 38 different data sources (the airline and the airport websites, third-party web resources). Each record represents a particular flight on a particular day and includes attributes such as the scheduled departure/arrival, the actual departure/arrival, and the departure/arrival gates. FBPosts² is a dataset of crawled Facebook posts for which we have chronological information, as well as the erroneous and the manually cleaned versions of the data (using OpenRefine [24]). The dataset contains information about a sample of posts - their title, content type, text, the week it was written, the domain and the image URL, the number of likes, and the web page it was crawled from. Missing values are the most common error type for this dataset. Both datasets have an attribute that defines the chronological order and enables splitting them into partitions. Two variants of each dataset, the one with errors occurred and the one where the errors are fixed, are provided. We utilize these variants as partitions of acceptable data quality and their corrupted counterparts for our evaluation scenario.

Datasets without ground-truth errors. Amazon Review [16] and the Online Retail³ [9] are two retail datasets. The Amazon Review⁴ dataset contains information about product reviews: their ID, title, category, brand, sales ranking, and related products. The Online Retail dataset contains historical transactional data from a UK-based retailer. It includes the invoice number, customer

ID, country, quantity, description, and the unit price of a product being purchased. The third dataset contains information about Drug Reviews⁵ [14]. It includes the name of a drug, medical conditions this drug has been designed for, ratings and reviews, the review date, and the number of users who considered this review useful. All three datasets have a mix of numeric and categorical attributes. They also contain an attribute that defines chronological order and enables partitioning, but we do not have ground-truth errors available for them.

Synthetic error types. In order to experiment with the datasets that do not provide ground truth, we inject six types of synthetic errors. We choose these types of errors because (a) they are commonly encountered in real-world use cases in industry and mentioned by many practitioners [6, 18] and (b) the majority of them is used as example error types in the research field of error detection [1, 27, 28, 34, 47]. We briefly describe these error types below.

- *Explicit missing values* - empty cells in the data as a result of wrong data collection or integration (e.g., left outer join of two tables) or, simply, an optional field in a web form that was never filled by the end-user and, thus, assigned as NULL while crawling. We remove a fraction of the values of an attribute, replacing them with NULLs;
- *Implicit missing values* - empty cells in the data that are encoded with values of an attribute’s data type that semantically represent a missing value, e.g., a string ‘NONE’ or a numeric value out of the attribute’s domain. In practice, implicit missing values are the result of missing value imputations mechanisms that are implemented in a data pipeline. We replace a fraction of the values of an attribute with ‘NONE’ values for textual fields or encode it as 99999 for numeric fields.
- *Numeric anomalies* - unexpected numeric values as a result of malfunctioning sensors, errors in scaling or type casting (e.g., change of measurement units from centimeters to meters, wrong parsing of *csv* files due to commas as decimal separators, etc.). For continuous numeric attributes, we corrupt a fraction of the values by replacing them with Gaussian noise that is centered at the mean value of the attribute and has a standard deviation that is scaled randomly from the interval of 2 to 5;
- *Swapped numeric fields* - misplacement of numeric values as a result of user mistake or wrong parsing, such as swapping the length and the width values of a retail product. We choose two numeric fields in the dataset and swap a fraction of the values from one attribute to another and vice versa;
- *Swapped textual fields* - analogous to swapped numeric fields on textual attributes, misplacement of textual values as a result of user mistake or wrong parsing, such as swapping the first name and the surname values of in a user registration form. We choose two textual fields in the dataset and swap a fraction of the values from one attribute to another and vice versa;
- *Typos* - unexpected spelling in textual attributes either due to user mistakes or errors in parsing (e.g., wrong encoding). We apply the “butterfinger” strategy that randomly replaces a fraction of letters in textual attributes with other letters that are neighbors on a “qwerty” keyboard layout.

Given the error types and descriptive statistics under investigation, sampling strategy does not have major effects on predictive performance of our approach in most cases. For instance, explicit

¹<http://lunadong.com/fusionDataSets.htm>

²<https://github.com/sergred/automating-data-quality-validation-data>

³<http://archive.ics.uci.edu/ml/datasets/Online+Retail/>

⁴<http://jmcauley.ucsd.edu/data/amazon/>

⁵<https://archive.ics.uci.edu/ml/datasets/Drug+Review+Dataset+%28Druglib.com%29>

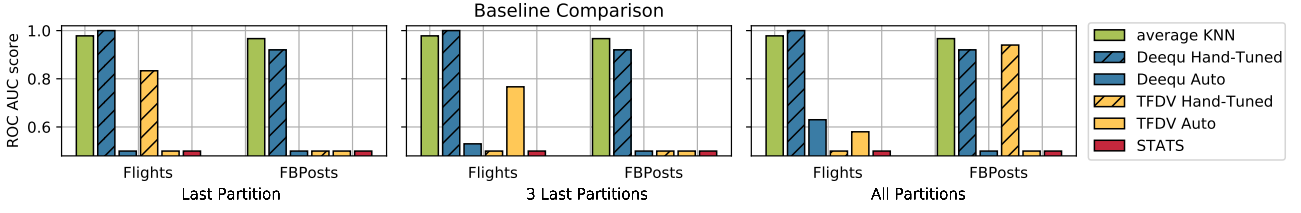


Figure 2: Comparison of the predictive performance of the proposed approach against three baseline solutions: Tensorflow Data Validation, Deequ, and statistical testing. The subplots represent three different training settings where the baselines learn from (a) only one recently observed data partition, (b) a combination of the last three data partitions, and (c) all the observed partitions. The TFDV and Deequ baselines are evaluated in their fully automated variant and a hand-tuned variant applying domain expertise. The bar chart shows that our approach outperforms the other automated baseline solutions and reaches the predictive performance of the hand-tuned baselines. The automated variants of the baselines tend to be conservative and produce false alarms in the majority of cases.

missing values would change the completeness measure, no matter wherein the data partition this error occurs. We use uniform distribution for error generation in the evaluation setup.

Hardware specification. We use an Ubuntu workstation with 8 Intel i7-8550U CPU cores (1.80GHz) and 24Gb RAM. We run all the algorithms with a single process and thread, with an exception of one baseline solution - Deequ library - that is built on top of Spark and runs at scale.

5.2 Comparison to Baselines

In our first experiment, we compare the predictive performance of our proposed approach (“avg. kNN” in Figure 2) to the existing baseline solutions: Tensorflow Data Validation [6], Deequ [43], and statistical testing [32, 41]. The purpose of this experiment is to evaluate whether our automated approach can reach the performance of hand-tuned state-of-the-art solutions.

Baselines. We compare the proposed approach against several existing solutions. As the first baseline, we use univariate *statistical tests* to detect shifts in data distribution between the previously observed data partitions and the current batch as an indicator of errors. We use two tests - the Kolmogorov-Smirnov test to detect shifts in continuous numeric attributes [32], and the Pearson’s Chi-squared test to detect shifts in frequency distribution for categorical values [41]. For every attribute of a data partition, we run one statistical test that gives a p -value as a measure of whether the data values in the current batch come from different data distribution than the values in previously observed data partitions. We choose a test based on the attribute’s data type (numerical or textual data) and compare the outcome to a common threshold of 0.05. Note that we apply Bonferroni correction to account for multiple tests.

We also use the *Tensorflow Data Validation* library [6] (TFDV) to detect data schema violations as an indicator of erroneous partitions. TFDV uses data profiling techniques to model the state of acceptable data quality by inferring their schema - attribute names, data domains, various constraints (e.g., on data distribution, uniqueness, sparsity, etc.). Then, it tests new data against inferred constraints and raises alerts upon schema violation as a signal for potential degradation of data quality. Domain experts use automated schema inference to facilitate data profiling and analysis but they have to hand-tune the schema to keep it up-to-date. In addition to the automated version of TFDV, we apply a hand-tuned version where we define its data schema based on data profiling and manual monitoring of data batches. This

setting aims to compare our approach to a baseline solution that exploits domain expertise.

Lastly, we include the *Amazon Deequ* library [42] and utilize its declarative data quality constraints to validate the data. Similar to the TFDV baseline, we evaluate Deequ in both an automated variant and a hand-tuned variant. In the former, Deequ runs data profiling and constraint suggestion algorithms to generate data unit tests to validate the quality of data partitions. In the latter, we utilize a hand-tuned variant where we manually define the checks to apply based on data profiling and inspection.

Evaluation scenario. For a relational dataset d comprised of chronologically ordered partitions d_1, \dots, d_{t_n} and timestamps t_1, \dots, t_n , we sequentially pick a timestamp t_k within the interval $start < k < n$, where $start$ is a predefined timestamp number to start with and n is the number of available partitions. We select $start$ as 8 in order to limit the minimum size of the training set to 8 data points. We show the partitions $d_{start}, \dots, d_{t_{k-1}}$ as training data to each approach.

For the datasets with the ground truth, we leverage the hand-labeled “dirty” versions $\hat{d}_1, \dots, \hat{d}_{t_n}$ of these partitions for the evaluation. We give both the clean data partition d_t and its corrupted counterpart \hat{d}_t to each approach, and let it decide whether the data batch is of acceptable quality or contains errors. In this experiment, we use only the datasets with available ground truth to compare the predictive performance in real-world cases with unspecified error types, error magnitudes, and real-world temporal changes in data characteristics.

For each approach, we record two predictions at each timestamp t_k in the interval $start < t < n$ - one label for the partition d_t and for the erroneous counterpart \hat{d}_t respectively. We compute the ROC AUC score based on the recorded prediction labels and the ground truth, where d_t has the “inlier” label, and \hat{d}_t has the “outlier” label. We evaluate the automated baseline solutions in three different settings, where the automated inference is based on (a) the last, (b) three last, and (c) all previously observed partitions with no further alteration of the derived rules, constraints, or patterns, to ensure systematic comparison of our approach in a fully automated mode. With the first two settings (one and three data partitions), we evaluate whether using only the most recent data is sufficient for the automated baseline solutions to learn the state of “acceptable” data quality accurately and fast. In contrast, the third setting is applied in order to evaluate the predictive performance of baselines that take the whole training set into account and include “far-in-the-past” data partitions.

For the given experimental scenario and datasets, we spent approximately two hours per dataset for data profiling, manual inspection, and configuration of DeeQu and TFDV via programming interfaces. For DeeQu, we implemented declarative unit tests for data. For TFDV, we adjusted thresholds to allow for particular fractions of previously unseen data and specified data ranges. We must point out, however, that hand-tuning involved analysis of the ground-truth clean data. In this way, we simulated a “domain expert” who knows what errors are expected in the data. In real-world use cases that assume “zero domain knowledge”, the analysis we conducted might be infeasible.

Results. Figure 2 depicts the comparison of the predictive performance of our approach (“Average KNN”, green) against the three baseline solutions: Tensorflow Data Validation (yellow), DeeQu (blue), and statistical testing (red). The bar charts report predictive performance on the `Flights` and the `FBPosts` datasets under three different training settings. The baselines learn from (a) only one recently observed data partition (“Last Partition”, left), (b) a combination of the last three data partitions (“3 Last Partitions”, center), and (c) all the observed partitions (“All Partitions”, right). Tensorflow Data Validation and DeeQu baselines are evaluated in both the fully automated mode and in their hand-tuned variant.

The results indicate that our approach outperforms other automated baseline solutions and reaches the predictive performance of hand-tuned baselines (ROC AUC score of 95%, whereas the hand-tuned DeeQu solution reaches 100% and 92% on the `Flights` and `FBPosts` datasets, respectively). Other automated solutions tend to produce false alarms in the majority of cases. We attribute this to the fact that the automated baseline solutions are “conservative” and strict in terms of their chosen constraints, and thereby produce false alarms in the majority of cases.

Table 3 depicts average execution times for both our approach and the baselines. It shows that, on average, our approach is at least one order of magnitude faster than the baseline solutions. High computational efficiency is associated with the fact that both the descriptive statistics and the KNN algorithm are easy to compute and train. Since the *DeeQu* library is built on top of Spark, this baseline takes more time to check data quality metrics for small datasets due to the large overhead for parallel computation. However, we assume that *DeeQu* might be more efficient on large-scale data, where other baseline solutions would perform reasonably slower.

Discussion. The errors in the dataset are mostly missing values or inconsistencies due to data integration (e.g., different datetime formats for different records). To be precise, 95% of the arrival and departure time information have an inconsistent date-time format, with a large fraction of the data missing. Inconsistencies in the datetime format lead to two problems - either the year is omitted, in which case several data preprocessing techniques replace the missing value with the default year 1970, or the day and month values are swapped as the solution has no means of distinguishing these values. 63% of the arrival and departure gates information is inconsistent in the following ways: (1) presence of explicit and implicit missing values; (2) the missing value encoding differs (e.g., ‘-’, ‘-’, ‘Not provided by airline’); or (3) the information is semantically incomplete (e.g., the ‘Gate 2’ value is replaced with the value ‘Terminal 8, Gate 2’, etc.). Since the cleaned version of the dataset was provided semi-automatically, most of the records which contained missing values were imputed where possible (e.g., by aggregation) or omitted as there

Table 3: Average execution time (in seconds) for baseline comparison. We compare our approach (Avg. KNN) against three baselines (DeeQu, Tensorflow Data Validation, and statistical testing), each of them computed in three modes, where (a) one last, (b) three last, and (c) all previously observed partitions are used for training. The table shows that the average execution time of our approach is one order of magnitude faster than the baselines.

Candidate	Mode	<i>Flights</i> Data	<i>FBPosts</i> Data	<i>Amazon</i> Data
Avg. KNN	-	0.042 +- 0.001	0.006 +- 0.001	0.215 +- 0.087
DeeQu	1 Last	0.322 +- 0.018	0.313 +- 0.020	0.782 +- 0.358
	3 Last	0.381 +- 0.026	0.329 +- 0.022	1.560 +- 0.800
	All	1.115 +- 0.382	0.468 +- 0.084	6.937 +- 5.427
TFDV	1 Last	0.141 +- 0.043	0.036 +- 0.008	6.679 +- 3.380
	3 Last	0.295 +- 0.060	0.058 +- 0.014	7.479 +- 3.753
	All	1.388 +- 0.702	0.126 +- 0.060	14.40 +- 9.940
STATS	1 Last	0.189 +- 0.025	0.160 +- 0.035	11.30 +- 3.575
	3 Last	0.194 +- 0.067	0.189 +- 0.061	20.20 +- 6.613
	All	0.204 +- 0.069	0.379 +- 0.439	105.6 +- 30.80

Table 4: Confusion matrices for the baseline comparison. Analogous to Table 3, we compare our approach against three baselines in three different modes. We evaluate TFDV and DeeQu baselines in their fully automated variant and a hand-tuned variant applying domain expertise.

Candidate	Mode	<i>Flights</i> Data				<i>FBPosts</i> Data			
		TP	FP	FN	TN	TP	FP	FN	TN
Avg. KNN	-	30	0	1	29	52	0	5	47
DeeQu	1 Last	30	0	30	0	50	2	51	1
	3 Last	30	0	28	2	52	0	52	0
	All	30	0	22	8	52	0	52	0
DeeQu Hand-Tuned	1 Last	30	0	0	30	48	4	4	48
	3 Last	30	0	0	30	48	4	4	48
	All	30	0	0	30	48	4	4	48
TFDV	1 Last	0	30	0	30	0	52	0	52
	3 Last	24	6	8	22	0	52	0	52
	All	28	2	23	7	0	52	0	52
TFDV Hand-Tuned	1 Last	21	9	2	28	0	52	0	52
	3 Last	0	30	0	30	0	52	0	52
	All	0	30	0	30	50	2	4	48
STATS	1 Last	0	30	0	30	0	52	0	52
	3 Last	0	30	0	30	0	52	0	52
	All	0	30	0	30	0	52	0	52

were no means to guarantee the correct missing value imputation scheme. 18% of the categorical attribute ‘contenttype’ have implicit missing value ‘nan’ or syntactic mismatch in categories (e.g., a combination of German and English words for ‘article’). 16% of the attribute ‘text’ have the wrong encoding.

Our approach performs well on the given datasets and reaches a ROC AUC score of 95%. Many of the baseline solutions, however, perform on the level of random guessing. Further analysis reveals that these baselines label the majority of the data partitions as erroneous (See Table 4). The reason why the data partitions are labeled as erroneous is due to the conservative default settings of the baseline solutions, as they are primarily designed to strictly detect data quality degradation and have false alarm rates as a secondary concern. Further analysis indicates that TFDV presumably detects errors in attributes where we know for certain

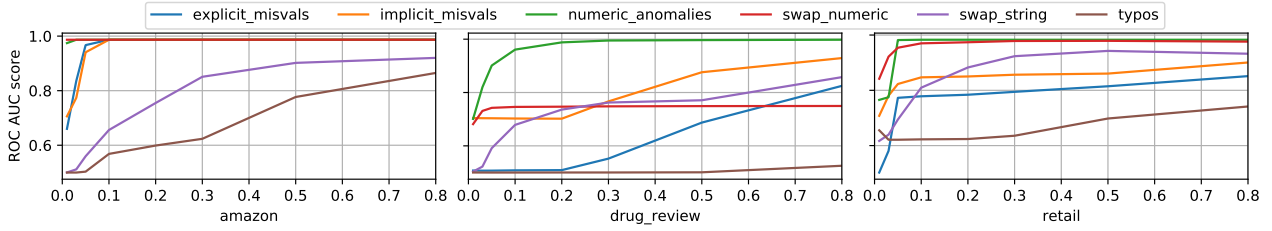


Figure 3: Overview of the predictive performance of our approach on three real-world datasets with synthetically generated errors under varying error magnitude (X-axis, 1 to 80%). We consider six error types: explicit and implicit missing values, numeric anomalies, typos in textual attributes, swapped fields for numeric and textual attributes. We observe two patterns: (a) similar predictive performance regardless of the fraction of errors (flat lines), or (b) gradual growth of the predictive performance towards bigger error magnitude, with the distinctive, more rapid growth for fractions up to 20%.

that there are no errors present. The ‘Source’ and the ‘Flight’ attributes of the *Flights* dataset do not contain errors. However, TFDV detects a violation of data schema as there are previously unseen values in the new batch, so the attribute domain has changed. A similar situation holds for the *FBPosts* dataset, with one additional type of alert - “non-boolean values” (as *FBPosts* contains one boolean attribute).

As for the hand-tuned baselines, DeeQu reaches a perfect ROC AUC score on the *Flights* dataset and 92% on the *FBPosts* with hand-tuned thresholds for the completeness metric. For TFDV, the ROC AUC score ranges from 50 to 82%. The “min domain mass” parameter (i.e., a minimal fraction of data records that have to be included from the inferred data domain) was set to 0 in order to allow for any fraction of previously unseen values in the data partition. Thresholds for the completeness metric were set similarly to the DeeQu baseline. This finding highlights that manual data quality monitoring and hand-tuning of existing solutions with the domain expertise is highly dataset-specific and tedious.

Note that, for Tensorflow Data Validation in several settings, the automated variants perform better than the hand-tuned variant. The reason is that the automated variants are retrained after a new data partition becomes available, whereas the hand-tuned variant is specified once on the initial training set (i.e., t_1 to t_{start}).

5.3 Sensitivity to Different Error Types and Magnitudes

In this experiment, we evaluate whether our approach detects all error types under varying error magnitudes with the similar predictive performance or whether there are error types that are harder to detect than others.

Evaluation scenario. For every dataset d with synthetically generated ground truth, we fix the error type and the error magnitude for generating corrupted data partitions \hat{d}_t . Other than that, the evaluation scenario is identical to the one in Section 5.2.

Results. Figure 3 shows line charts that represent predictive performance of our proposed approach per dataset and error type, where the x-axes of the plot depict the error magnitude. We are interested in the relationship between the predictive performance of our approach and the fraction of errors that are introduced in data partitions. Two distinctive patterns arise in terms of the curve shapes: (a) flat lines represent similar predictive performance regardless of the fraction of errors, whereas (b) the curves with gradual growth towards more significant error magnitudes mean that it is easier to detect degradation in data quality with greater fractions of the data partition being affected.

The latter curves capture rapid increase for smaller fractions of 1 to 20%. The relative difference in predictive performance between the error types varies among the datasets and error magnitudes. Even though the Drug Review and the Online Retail datasets show resemblance in terms of the ROC AUC score, the Amazon dataset exhibits different patterns. For instance, the kNN novelty detection approach shows constant predictive performance rate on Amazon’s numeric anomalies but has a “learning curve” for Drug Review or Online Retail.

Discussion. The figure shows that, in general, the predictive performance differs from one error type to another. We attribute this behavior to two findings from the experiment’s analysis. First, some types of errors are, in fact, easier to recognize than others. That statement holds for the use cases of manual data quality monitoring that are conducted by domain experts. For instance, an explicit missing value (e.g., a NULL value) is reasonably straightforward to detect even when few data records are corrupted. Other error types, such as numeric anomalies, can be detected only in cases where the ranges of acceptable values are available, or the assumption on data distribution exist [18]. Comparing ROC AUC scores between the error types, error magnitudes, and datasets indicates that predictive performance is dataset-specific and likely depends on scales and domains of every data attribute. In the majority of cases, however, missing values and numeric anomalies can be detected relatively reliably and result in high ROC AUC scores.

For every error type that we investigate, there are descriptive statistics that provide better features for classification. For instance, the completeness measure is more descriptive to detect explicit missing values. Data distribution measures (e.g., mean, standard deviation, minimum, maximum) are more descriptive to detect numeric anomalies. However, there is no single metric that is more descriptive than others for all given error types.

Note that our approach often performs reasonably well in cases of small error magnitudes (already at 10%), when introduced errors drastically affect the descriptive statistics of a data partition. Should our approach be insensitive to a specific error distribution (or particular error types), our approach can be extended by adding another descriptive statistic that is sensitive to this error distribution or error type.

Based on Figure 3, typos (brown) appear to be the hardest error type that we consider in this study. We assume that the index of peculiarity for textual attributes is a direct proxy for this error. However, predictive performance on the Drug Review dataset nearly reaches the level of random guessing, whereas on other datasets it exhibits a slow learning curve. Further experimental analysis reveals several differences between textual attributes on

the datasets under investigation. Our approach performs well in cases where attributes have categorical values with rather low cardinality and high repetition of values (e.g., country code). It also performs well on long texts such as reviews and descriptions with high a likelihood of word repetition within the data batch. In this case, a typo that is introduced in one word that repeats itself in the data batch yields high chances for this error to be detected by our approach, as this word becomes “peculiar” in the context of the data batch. On the other hand, typos that are introduced in almost-unique words that belong to a dictionary of a textual attribute would not be detected as this error replaces one unique word to another. For several curves that involved textual attributes, there exists a downward trend at the beginning when the training set is small. It happens due to our design decision to keep a constant contamination parameter (see Section 4, “Modeling Decisions”). In cases of small training sets, the kNN algorithm learns a broad decision boundary that leads to false positive results (i.e., where the majority of data points are considered inliers). Only with the growing training set, the decision boundary becomes smaller and yields more accurate results. One preventative measure is to ensure large initial training sets. When this is not possible, another option is to adaptively select larger contamination parameters for smaller training sets.

We obtain several findings regarding the relationship between the predictive performance of our approach and error magnitude. In general, we note two patterns in the curve. The first case is where the ROC AUC score remains approximately constant across all error magnitudes and does not depend on the fraction of corrupted records in a data partition. This happens in cases where a few erroneous records in a data partition are sufficient to affect descriptive statistics and reliably identify the data partition as erroneous. The second case is where the ROC AUC score increases gradually with the growth of the error fraction. In this case, the reason is that detecting data quality degradation becomes easier when more data in the partition are corrupted. One example is the explicit missing values error type. Note that, for this example, a clean partition d_t might allow for missing values, so that a simple rule of “100% completeness” is not applicable. Thus, the higher the difference between the fraction of missing values in between clean and erroneous data partitions, the higher is the overall ROC AUC score. Note that the shape of the curve and the rate of growth are dataset-specific.

5.4 Sensitivity to a Combination of Errors

We also extend the experiment from Section 5.3 to evaluate the sensitivity of our approach to scenarios where a combination of two different error types occurs in the same data partition.

Evaluation scenario. For every dataset d with synthetically generated ground truth, we fix the error magnitude to 50% for generating corrupted data partitions \hat{d}_t . We choose an attribute A_m of every data partition d_t and apply a pair of error types (if suitable for the attribute’s data type). We use all pairwise combinations of error types under investigation. Other than that, the evaluation scenario is identical to the one in Section 5.2. Note that, as we sample the values-to-corrupt uniformly, there is an overlap in selected cells of a data partition d_t for the first and the second error type of the pair ($\sim 40\%$). For the overlapping values, the second error type overrides the changes made by the first type, resulting in approximate distribution of corrupted values to be 20% of the data partition and 30% respectively. In the case when the union of changes provided by each error type exceeds

50% of the data partition, we uniformly sample changes from the union to ensure total error magnitude of 50%. We compare the predictive performance of our approach to the respective performance when only one of the error types is applied.

Results. For every attribute of every dataset and every applied pair of error types under investigation, we computed three ROC AUC scores: the one where only the first error type is applied to corrupt the data, the one where only the second error type is applied, and one for a combination of applied error types. For all computed scores, we report the mean squared error of 0.028 between the ROC AUC score on a combination of error types and the maximum of ROC AUC scores where only one of the two error types is applied.

Discussion. The results indicate that the predictive performance of our approach in the case when two error types are combined is, on average, close to the performance on a single error type, the “easiest to detect” of the two, taking into account reduced error magnitudes (i.e., when errors corrupt 20-30% of the data partition separately, adding up to a total error magnitude of 50%). We generalize this observation to a combination of more than two error types that corrupt a data partition together.

5.5 Detection Quality over Time

In this experiment, we evaluate the detection quality of our approach over time. The motivation behind this experiment is twofold: (a) the size of the training set for the novelty detection algorithm grows continuously, which might gradually improve predictive performance, and (b) data characteristics are volatile and can change over time, which might lead to the occasional degradation of predictive performance.

Evaluation scenario. For every dataset d with synthetic errors, we fix the error type for generating corrupted data partitions \hat{d}_t . We compute two labels for every daily-ingested data partition, one for the clean variant and one for the corrupted counterpart. When we visualize ROC AUC scores over time, we aggregate these labels on a monthly basis and plot line charts with months as X-axes. Other than that, we leverage a setup that is identical to previous experiments.

Results. Figure 4 depicts the line charts that represent changes in the predictive performance of our approach over time, where the x-axis is the monthly time window (for clarity reasons, it is shown by year in the “Drug Review” graph). Two distinctive patterns arise in terms of the curve shape: (a) flat lines represent approximately constant predictive performance, whereas (b) curves with the gradual increase indicate improvements over time and, respectively, with the growing size of the training set (see Drug Review). The latter examples converge to a stable rate and further resemble the behavior of approximately constant predictive performance.

Discussion. The results indicate how the predictive performance of our approach changes over time, with the corresponding growth of the training set for the novelty detection algorithm to learn from. Similar to the previous section, we see two patterns. First, in most of the cases, the average prediction performance does not change significantly over time. This finding might be counter-intuitive at first, as we usually assume that an ML-based algorithm tends to perform better with more data points to train from. The reason for the approximately constant ROC AUC score is that data points that represent erroneous data partitions are likely to be far from the decision boundaries learned by the kNN

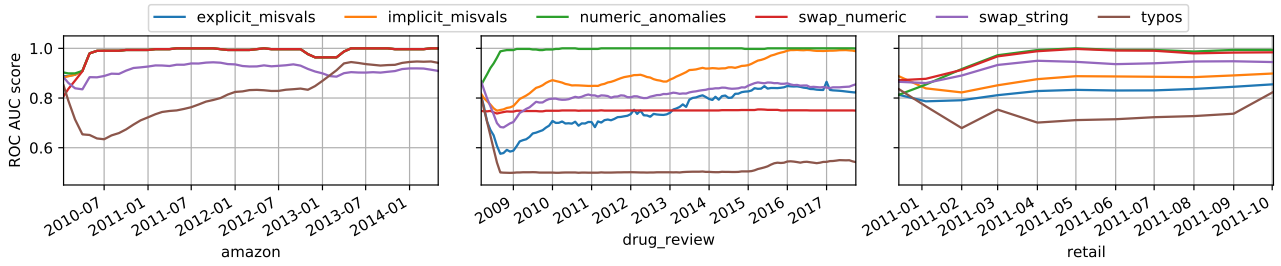


Figure 4: Predictive performance of our approach over time (X-axis). The figure depicts line charts that represent the ROC AUC score for every dataset with the ground truth, per error type over time. Various magnitudes of errors and data attributes are aggregated. The results show that, in the majority of cases, the predictive performance does not improve significantly over time with the growth of the size of the training set. Several cases (Retail, swapped fields and numerical outliers) demonstrate an initial increase of the ROC AUC score, followed by convergence to a stable rate over time. Note that, for the Amazon dataset, line charts that depict missing values, numeric anomalies, and swapped fields error types overlap and are represented by one line (red).

algorithm. These far-off data points (i.e., outliers) are likely to be detected reliably even under the limited size of the training set and, therefore, lead to the stable predictive performance of the kNN novelty detection approach. The second pattern is a gradual increase in the predictive performance in the beginning until we converge to a stable performance rate. Examples are explicit missing values and swapped textual fields for the *Drug Review* dataset. We attribute this pattern to the insufficient size of the training set to learn the decision boundaries that lead to reliable predictions. We assume that the stage of gradual increase in predictive performance corresponds to the “learning process” of the approach to derive accurate decision boundaries with clear benefits of accumulating more data points to the training set. After convergence, re-training of the approach is necessary to self-adapt to temporal changes in data.

The importance of batch frequency. Preliminary experiments show that, when choosing between daily, weekly, and monthly ingestion frequencies, daily ingestion of data led to relatively higher predictive performance. We associate this phenomenon with larger sizes of the training set.

6 RELATED WORK

We distinguish two lines of research that address related data management issues at different angles - *error detection for data cleaning* [1, 27, 36, 40, 47] and *data quality validation* [3, 20, 43].

Error detection for data cleaning. The goal of error detection mechanisms is to find the exact data records and attributes that contain errors. Abedjan et al. [1] consider four different categories of error detection solutions: (a) rule violation, (b) pattern violation, (c) outlier detection, or (d) duplicate conflict resolution based systems. Some of the algorithms require rules or patterns to be specified by the end-user. Outlier detection based methods require clean data to be present in order to “learn” what the inliers are and then decide whether particular records deviate from the expected behavior. Our approach follows similar ideas but constructs feature vectors based on the corresponding data quality metrics that are computed over the data partition instead of relying on the raw data itself. This leads to feature vectors of low dimensionality, fast model training, and guarantees numeric representation of feature vectors. The last category, duplicate conflict resolution systems, handles the specific case of duplicate

entities in the data, and does not cover other types of errors. Compared to the existing error detection algorithms, our approach can detect unspecified error types and does not require domain expertise in terms of rules, patterns, or labeled examples.

Data validation. These methods aim to make a decision whether the data is valid w.r.t. particular assumptions. Tensorflow Data Validation [6] models the state of acceptable data quality with the user-defined data schema - attribute names, data domains, various constraints (e.g., on data distribution, uniqueness, sparsity, etc.). It, then, tests new data against the specified constraints and raises alerts upon schema violation. To assist the end-user, initial data schema can be inferred automatically by analyzing reference data (i.e., an “acceptable” data sample). As stated by the authors, schema refinement by domain experts is required to guarantee the performance of the library, and the schema inference functionality is provided as an aid, not a replacement of the domain expert. Data linter [20], on the contrary, validates data against data lints - deviations from accepted practices of data analysis (analogous to code lints - snippets of code that depict deviation from best practices in software engineering). The lints are predefined by the developers of the tool yet are extensible in case customized practices are in place. Another example is the Deequ library for automating the verification of data quality at scale [42], which proposes unit tests for data - a declarative specification of integrity constraints, such as completeness, consistency, syntactic and semantic accuracy, which the end-user needs to specify. Schelter et al. [42] also introduce functionality for automated constraint suggestion based on data profiles (collected descriptive statistics on data attributes). However, this method requires the presence of reference data - a sample of the data population that is considered to be of acceptable quality and is designed to generate suggestions that are validated by a domain expert. The Metanome platform [36] is a tool for data profiling that incorporates numerous algorithms for the detection of functional, order, or inclusion dependencies, as well as cardinality estimation. This method is not a data validation solution as such, but allows to automatically discover patterns from data that later could be used as rules for data quality. Metanome requires “acceptable” data samples to be present for reliable mining of the data quality patterns. As the main purpose of Metanome is data profiling and not directly data quality validation, this framework requires additional rules for detection of data quality issues and cannot be used directly as a data quality validation tool [10].

To summarize, existing approaches require domain knowledge to define rules, denial constraints, patterns, configuration of error detection solutions [1, 27], integrity constraints, data unit tests [42], error generators [39], data schema [6], or data lints [20]. Automation tools exist for data profiling, constraint suggestion, schema inference, or error detection. These solutions assume a domain expert in the loop. Our approach, in contrast, does not require any domain knowledge specified explicitly for common error types. In contrast to existing solutions, it is inspired by the work of Bleiweiß et al. [4] on exploring changes in dynamic data, Ehrlinger et al. [10] on automating data quality validation, and Ioannou et al. [21] on generating benchmark data. Finally, as our experimental analysis indicates, few automated solutions for data quality validation appear to be particularly “conservative” and produce false alarms in the majority of cases.

7 CONCLUSION & FUTURE WORK

Data quality validation is crucial for large-scale production pipelines. Challenging cases are the ones where domain expertise is incomplete and data changes over time. We showed that collecting simple descriptive statistics over the data and analyzing them with novelty detection methods makes it possible to distinguish critical errors in data. In contrast to existing solutions, our approach does not require domain experts to define rules or labeled examples, and self-adapts to temporal changes in the data characteristics. We evaluated our approach against existing baselines on five real-world datasets with real and synthetically generated errors. We found that our approach detects the unspecified errors in many cases under varying error magnitudes, outperforms other automated solutions in terms of predictive performance, and reaches the ROC AUC score of baselines that are hand-tuned with domain expertise.

In future work, we plan to investigate more exotic types of errors and intend to look deeper into specific types of errors that are hard to capture by common data quality metrics, e.g., errors that are a deterministic function of the inputs (like accidentally changing the encoding). As there exist few real-world datasets that are available for evaluation purposes in data quality validation on dynamic data, we also intend to provide a set of benchmarking datasets. These datasets should contain a wide range of error types and patterns of temporal changes in data characteristics. This will enable research on controlling the false alarm rates for novelty detection algorithms in data quality validation settings.

Acknowledgements. This work was funded by the HEIBRiDS graduate school, with the support of the German Ministry for Education and Research as BIFOLD, BBDC 2 (01IS18025A), BZML (01IS18037A), the Software Campus Program (01IS17052), and Ahold Delhaize. All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

REFERENCES

- [1] Ziawasch Abedjan et al. 2016. Detecting Data Errors: Where Are We and What Needs to Be Done?. In *PVLDB*, Vol. 9.
- [2] Fabrizio Angiulli and Clara Pizzuti. 2002. Fast outlier detection in high dimensional spaces. In *PKDD*.
- [3] Dennis Baylor et al. 2017. TFX: A Tensorflow-based Production-scale Machine Learning Platform. In *KDD*.
- [4] Tobias Bleiweiß et al. 2018. Exploring Change - A New Dimension of Data Analytics. In *PVLDB*, Vol. 12.
- [5] Michael Brackett and Production Susan Earley. 2009. The DAMA Guide to The Data Management Body of Knowledge (DAMA-DMBOK Guide). (2009).
- [6] Eric Breck, Marty Zinkevich, Neoklis Polyzotis, Steven Whang, and Sudip Roy. 2019. Data Validation for Machine Learning. *SysML*.
- [7] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009).
- [8] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding frequent items in data streams. In *ICALP*.
- [9] Daqing Chen, Sai Laing Sain, and Kun Guo. 2012. Data mining for the online retail industry: A case study of RFM model-based customer segmentation using data mining. (2012).
- [10] Lisa Ehrlinger and Wolfram Wöß. 2017. Automated data quality monitoring. In *ICIQ*.
- [11] Martin J Eppler. 2006. *Managing information quality: Increasing the value of information in knowledge-intensive products and processes*. Springer Science & Business Media.
- [12] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *AofA*.
- [13] Hector Garcia-Molina. 2008. *Database systems: the complete book*. Pearson Education India.
- [14] Felix Gräßer, Surya Kallumadi, Hagen Malberg, and Sebastian Zaunseder. 2018. Aspect-Based Sentiment Analysis of Drug Reviews Applying Cross-Domain and Cross-Data Learning. In *DH*.
- [15] Xiaoyi Gu, Leman Akoglu, and Alessandro Rinaldo. 2019. Statistical Analysis of Nearest Neighbor Methods for Anomaly Detection. In *NeurIPS*.
- [16] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*.
- [17] Alireza Heidari, Joshua McGrath, Ihab Ilyas, and Theodoros Rekatsinas. 2019. HoloDetect: Few-Shot Learning for Error Detection. In *CoRR*. arXiv:1904.02285
- [18] Joseph M Hellerstein. 2008. Quantitative data cleaning for large databases. *UNECE*.
- [19] Peter J Huber. 1992. Robust estimation of a location parameter. In *Breakthroughs in statistics*. Springer.
- [20] Nick Hynes, D Sculley, and Michael Terry. 2017. The data linter: Lightweight, automated sanity checking for ml data sets. In *NIPS ML Sys Workshop*.
- [21] Ekaterini Ioannou, Nataliya Rassadko, and Yannis Velegrakis. 2013. On generating benchmark data for entity matching. *Journal on Data Semantics* (2013).
- [22] Jin Huang and C. X. Ling. 2005. Using AUC and accuracy in evaluating learning algorithms. *TKDE* 17, 3 (2005).
- [23] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. 2008. Angle-based outlier detection in high-dimensional data. In *SIGKDD*.
- [24] Tien Fabrianti Kusumasari et al. 2016. Data profiling for data quality improvement with OpenRefine. In *ICITSI*.
- [25] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiye Meng, and Divesh Srivastava. 2012. Truth finding on the deep web: Is the problem solved?. In *PVLDB*, Vol. 6.
- [26] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *ICDM*.
- [27] Mohammad Mahdavi et al. 2019. Raha: A configuration-free error detection system. In *SIGMOD*.
- [28] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning. In *PVLDB*, Vol. 13.
- [29] Zelda Mariet, Rachael Harding, Sam Madden, et al. 2016. Outlier detection in heterogeneous datasets using automatic tuple expansion. (2016).
- [30] Markos Markou and Sameer Singh. 2003. Novelty detection: a review—part 1: statistical approaches. *Signal processing* 83, 12 (2003).
- [31] Markos Markou and Sameer Singh. 2003. Novelty detection: a review—part 2: neural network based approaches. *Signal processing* 83, 12 (2003).
- [32] Frank J Massey Jr. 1951. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association* 46, 253 (1951).
- [33] Robert Morris and Lorinda L Cherry. 1975. Computer detection of typographical errors. *IEEE Transactions on Professional Communication* 1 (1975).
- [34] Felix Neutatz, Mohammad Mahdavi, and Ziawasch Abedjan. 2019. ED2: A Case for Active Learning in Error Detection. In *CIKM*.
- [35] Stephen M Omohundro. 1989. *Five balltree construction algorithms*. International Computer Science Institute Berkeley.
- [36] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. 2015. Data Profiling with Metanome. In *PVLDB*, Vol. 8.
- [37] Neoklis Polyzotis et al. 2017. Data Management Challenges in Production Machine Learning. In *SIGMOD*.
- [38] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. 2000. Efficient algorithms for mining outliers from large data sets. In *SIGMOD*.
- [39] Sergey Redyuk et al. 2019. Learning to Validate the Predictions of Black Box Machine Learning Models on Unseen Data. In *HILDA*.
- [40] Theodoros Rekatsinas, Xu Chu, Ihab Ilyas, and Christopher Ré. 2017. HoloClean: Holistic data repairs with probabilistic inference. In *PVLDB*, Vol. 10.
- [41] Albert Satorra and Pete M Bentler. 1994. Corrections to test statistics and standard errors in covariance structure analysis. (1994).
- [42] Sebastian Schelter et al. 2018. Automating Large-scale Data Quality Verification. In *PVLDB*, Vol. 11.
- [43] Sebastian Schelter et al. 2019. Unit Testing Data with DeeQu. In *SIGMOD*.
- [44] Bernhard Schölkopf et al. 2000. Support vector method for novelty detection. In *NeurIPS*.
- [45] Michael Stonebraker and Ihab Ilyas. 2018. Data Integration: The Current Status and the Way Forward. *IEEE Data Eng. Bull.* 41, 2 (2018).
- [46] David Martinus Johannes Tax. 2001. *One-class classification: Concept learning in the absence of counter-examples*. Ph.D. Dissertation. TU Delft.
- [47] Larisa Visengeriyeva and Ziawasch Abedjan. 2018. Metadata-Driven Error Detection. In *SSDBM*.