Real-Time Data Quality Analysis

Arun Iyengar, Dhaval Patel, Shrey Shrivastava, Nianjun Zhou, Anuradha Bhamidipaty
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598
Email: {aruni, pateldha}@us.ibm.com, shrey@ibm.com, {jzhou, anubham}@us.ibm.com

***Abstract*—Data quality is critically important for big data and machine learning applications. Data quality systems can analyze data sets for quality and detection of potential errors. They can also provide remediation to fix problems encountered in analyzing data sets. This paper discusses key features that of data quality analysis systems. We also present new algorithms for efficiently maintaining updated data quality metrics on changing data sets. Our algorithms consider anomalies in data regions in determining how much different regions of data contribute to overall data metrics. We also make intelligent choices of which data metrics to update and how frequently to do so in order to limit the overhead for data quality metric updates.**

I. Introduction

Data quality is critically important. Incorrect or incomplete values can lead to significant problems in analyzing the data. In order to check for errors and improve data quality, a number of systems have been developed for automating the process of checking for data quality [1], [2], [3], [4], [5]. Data quality checks have also been handled by database systems; such data quality checks can generally only be made on data within the database system. By contrast, the data quality checking systems in this paper are applicable to data in general; the data does not need to reside within a specific database and can be passed as parameters to the functions and methods which perform the data quality checks.

This paper provides a systematic overview of the key features that should be included in systems for analyzing data quality in a wide variety of data analytics and machine learning applications. We also present new methods for efficiently analyzing data sets which are changing over time. In many cases, new data are constantly being streamed in. This requires data quality metrics to be constantly updated as new data are received. We present new techniques for optimizing data quality checks and reducing run-time overhead under these circumstances.

Data quality can mean different things. [6] defines multiple data quality dimensions including accessibility, appropriate amount of data, believability, completeness, concise representation, consistent representation, ease of manipulation, free-of-error, interpretability, objectivity, relevancy, reputation, security, timeliness, and value-added, which is the extent to which data is beneficial and provides advantages from its use.

More recently, [1] defined data quality as having the following dimensions:

- Completeness: the degree to which an entity includes data required to describe a real-world object.
- Consistency: the degree to which a set of semantic rules are violated. There can be intra-relation constraints, as well as inter-relation constraints. An inter-relation constraint in tabular/relational data may involve data from multiple tables.
- Accuracy: correctness of data. There is both syntactic and semantic accuracy.

In this paper, we look at data quality systems with a focus on analyzing real-time data which is constantly changing. The remainder of the paper is structured in the following way. Section II describes key characteristics which we believe are important for state-of-the-art data quality systems. Section III presents new algorithms that we have developed for efficiently analyzing the quality of changing data sets. Finally, Section IV concludes the paper.

II. Important Characteristics of Data Quality Systems

Data quality systems need to deal with large data sets which are constantly changing. The data sets may be used for a variety of different purposes and may be used for machine learning models. In this section, we describe important features in data quality systems.

We define the following features as being essential in a data quality system:

- Having the ability to test data sets for quality using both predefined functions as well as user-defined functions. Predefined functions should exist for common operations such as checking for missing values. Users should be able to define their own functions for operations such as outlier detection.
- The data quality system should be able to associate constraints with a data sets. Data quality checks include determining if the constraints are satisfied. The data quality system should be able to infer some constraints by analyzing data sets. Users can provide other constraints. The data quality system can infer an initial set of constraints, and users can refine the initial set of constraints inferred by the data quality system.
- The data quality system should be able to handle changing data sets and provide analyses of data quality metrics over time over many updates to a data set.
- Data quality systems should provide remediation where possible, to try to fix data quality problems. One example of remediation is imputing missing data values.

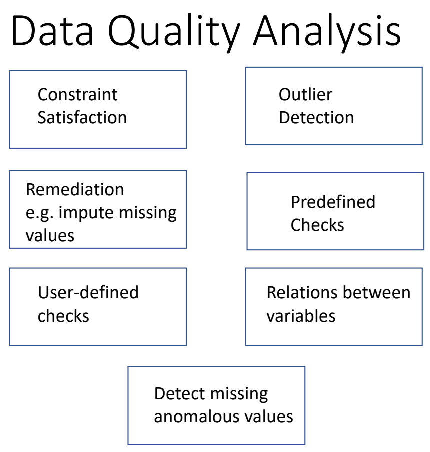

Fig. 1: Common operations in data quality analysis.

- The data quality system needs a good user interface. This includes providing an easy way for users to provide their own data quality checks.
- Run-time performance and scalability are critically important. The data quality system should thus be optimized to efficiently handle many data quality checks over a significant amount of data. Both predefined data checks and user-defined data checks should be properly optimized for performance.
- A data quality system should be customizable so that domain-specific data quality checks can be provided to handle data sets in specific problem domains.
- Data sets are often analyzed using advanced machine learning algorithms. Ideally, the way in which data is analyzed should be considered in determining the types of data quality checks which are most applicable.

Figure 1 depicts common operations performed by data quality systems.

Data quality systems typically have built in checks, such as quantities of missing, unique, distinct, zero, or infinite values in a data set or part of a data set, such as a row or column. There can also be checks for correctness and distributions of data types. It is often desirable to determine statistical properties of data sets, such as means, medians, modes, maximums, and minimums. Determining relationships between different columns of a table, such as correlations, is also of value.

The data quality system can include special checks for certain types of data. For example, there are a number of data quality checks that can be performed specifically for time series data. These include checks on the validity of timestamps associated with a data point. For example, checks can determine if timestamps are missing or out of order. Checks could also determine if there are duplicate timestamps, as well as if timestamps are evenly spaced. The data quality system could also determine if there are gaps between adjacent time stamps which are too long. Checks for sampling frequency are also important.

Data quality checks can take the form of constraints. Constraints are well-known from the database community. SQL defines a number of constraints, such as NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, and others. The types of constraints in relational databases have been used in other data quality systems.

The data quality system should have the ability to automatically determine which constraints are likely to hold for a data set. This can be done by testing a set of constraints on a data set and showing the results to users. Users can then refine the set of constraints that the data quality system came up with by possibly removing some and adding others. The revised list of constraints can then be applied to new data as it is received.

In addition to predefined data quality checks and constraints, a data quality system should provide users with the ability to define their own data quality checks and constraints. That way, powerful data quality checks can be customized to the particular data sets being analyzed. One way to allow users to specify their own data quality checks (including constraints) is to allow them to define their own methods or functions in a programming language such as Python. The data quality system would then invoke the code provided by users to perform the data quality checks.

As an example, outlier detection is an important feature to have in a data quality system. The data quality system can provide basic outlier detection, which would typically involve range checking, such as determining if a numerical value falls within a certain range of values. Since general outlier detection is more complex, users should have the ability to provide their own functions for detecting outliers.

It should be noted that providing a programmatic application programming interface (API) using functions or methods in a language such as Python requires users to have programming skills. An easier to use interface graphical user interface, such as one running in a Web browser, can be provided for basic features. It would not be feasible to provided complicated user-defined data quality checks using this approach. Future data quality systems could provide natural language or speech recognition capabilities which would allow users to control them via text or spoken dialog [7], [8], [9].

Data quality checking systems for machine learning can go significantly beyond functionality typically associated with data quality. One key aspect is remediation. Remediation is concerned with fixing data quality problems. A typical example is handling missing values in data sets. A good data quality system should be able to handle missing values via data imputation. There are a wide variety of data imputation algorithms to choose from. A good data quality checking system should provide a range of different data imputation algorithms. Simple imputation techniques like mean, median, and mode are commonly used. More sophisticated such as multiple imputation using chained equations (MICE) [10] can result in better accuracy but with higher overhead. Other techniques such as neural nets and deep learning have also been used for data imputation [11], [12], [13].

When remediation is performed, there should be ways of determining how good the remediation techniques are. For data imputation, one commonly used method for determining the accuracy of imputation algorithms is to delete known values and see how accurately each imputation algorithm is able to predict the deleted known values. The patterns of missing values can affect the results. It is thus preferable to delete known values in a manner which is similar to the patterns of missing data seen in real data sets. The performance of different imputation algorithms also generally depends on the amount of missing data. In order to accurately compare the performance of imputation algorithms, several runs should be made using different sets of missing data. The overhead for determining the performance of several imputation algorithms can be significant, particularly for large data sets. It should be noted that many data imputation algorithms have parameters. Optimizing the parameters for each algorithm adds further overhead. Empirical evidence can be used to preselect a small number of parameter settings likely to give good performance. The selection process can further be narrowed by making intelligent choices on the type of data imputation algorithm based on the nature of the missing data.

In some cases, the best imputation algorithms can be determined by considering analytics tasks associated with the data. For example, suppose that we are trying to construct the best machine learning model (s) from training data with missing values. In order to construct these models, it may be necessary to impute missing values, as the model construction algorithms may require complete data sets. In this type of scenario, imputation is an essential step in constructing the machine learning models. We can try out different imputation algorithms and pick the one (s) which results in the best fitting models. This is an example of how the way in which data sets are analyzed should be considered in analyzing data quality.

Data quality checks can also include detecting data values which are outliers. Anomaly detection is a key aspect, wherein anomalous patterns in data can be detected. Data quality checks can also include detecting changes in data over time and data drift detection [14], [15], [16].

A. *Imputation*

In our system, the imputation is a crucial component to ensure the final quality of a dataset. There are several requirements for this component. First, the component must provide an imputer algorithm to complete high quality of imputation. Second, we need to have a quick or at least an acceptable response time (run-time) for the imputation task. Third, the quality of the imputation must be persistent. We define persistence as an imputer's performance consistency for the set of datasets sharing similar data properties and similar missing patterns. The persistence requirement comes from the likelihood of repeating the same imputation algorithm for a sequence of imputation tasks for a similar dataset. We define the missing patterns as the numerical representation of missing patterns (such as the missing interval and missing consecutive length distributions).

We support and implement a large number of imputers [17], [18], [19] as the imputer pool in our imputation component and a mechanism to select the best imputer. All the imputers have the same *API* with a dataset requiring imputing and a set of imputation parameters as input. Each of the imputers returns an imputed time series and an object containing results related to the imputation quality. We group those imputers into three categories. Except for the statistical imputers, other imputers utilize the neighborhood (temporal proximity) knowledge around the missing data to estimate missing values. Different imputers have a different neighborhood scope and different algorithms to utilize the neighborhood values for imputing. Considering the potential needs of online and offline imputing, we have some imputers that only use forward information (past) for online imputation. Most of them use both forward and backward neighborhood information to enhance the accuracy of imputing. Such imputers only can be used in offline imputation tasks.

- Statistical Imputers - Using the statistical properties, such as mean, median, and mode values for missing;
- Interpolation-based and Prediction-based Imputers – applying various interpolation or rolling and moving average and interpolation functions to impute the missing;
- ML-based imputers - Applying machine learning, including deep learning for imputing. We implement the ML-based imputers to incorporate the start-of-art imputation algorithms from the latest research papers [20], [21], [22].

Naturally, each imputer of our imputation pool has its pros and cons. For example, the pros and cons come from 1) the ability to handle different missing patterns (such as an independent and identically distributed ($i.i.d$) missing versus consecutive values for time series); the ability to support online imputing (predictive imputation) versus only supporting offline imputation (interpolation); and the ability to handle the noise in the imputation task.

Another necessary functionality of the imputation component is to support the imputer selection for a given dataset. To better serve the selection process, we define multiple metrics for each imputer. We compute the metrics based on multiple iterations of the imputation over the same dataset. We have a more detailed explanation in the next paragraph. The performance metrics are a) mean-square-error (*MSE*), b) mean-absolute-error (*MAE*), c) the *p*-value of the hypothesis test of the imputed and original datasets are statistically identical, and d) Kullback–Leibler (*K-L*)divergence distance of the imputed and original datasets. The run-time imputation time metric is the time required for the imputation. The persistence metric is the consistency of an imputer. We define persistence as the ratio (variance ratio) of the standard deviation of a performance metric (such as *MAE* or *MSE*) over the metric's average.

We support two methods of imputation selection. First, in requiring a quick decision of best imputer selection, we have a static lookup allowing the user to choose the best imputer. We have developed a lookup table which contains the imputation metrics for a given set of benchmark datasets with different missing patterns. Three of the datasets come from R's time series package (imputeTS). The remaining six are publicly available time series datasets. A user selects an imputer based on the imputation requirements (accuracy, run-time response time, and persistence) and the dataset's (and missing pattern's) similarity compared with the benchmark dataset (and missing pattern). To ensure the lookup table's metrics' reliability, we iterate multiple times for a given dataset and a missing pattern to compute the metric's averages and standard deviations. Each iteration generates different missing data points but following the same missing pattern.

In addition to the lookup table, we also provide a more advanced algorithm to select the best imputer, named performance verifiable algorithm or functionality [23]. Different from the static lookup table, the performance verifiable algorithm provides a more reliable performance assessment. We add artificial missing data points and estimate an imputer's performance based on imputation performance on those added points. Our algorithm first analyzes the missing pattern, then creates artificial missing values based on the missing pattern. We calculate the performance by comparing the imputed values and the ground truths for the artificial missing data points. Again, to ensure selection reliability, we have multiple iterations of generating artificial missing data points. We use the performance metric and run-time response time averages as performance, algorithm speed, and variance ratio as the imputer's persistence metric. Such a verifiable performance algorithm is more reliable than a static lookup table to choose the best imputer. However, we still need to pay attention to a possible pitfall. We notice that such estimation might generate system bias to underestimate the imputer's accuracy if we leverage a large size of artificial missing data points. The reason is that the artificial added missing data points disturb the original missing. To minimize the impact of potential systematic bias, we need to control the introduced artificial missing points. The minimal size acceptable is introducing only ONE missing point sampled. However, if we choose a small number of artificial missing data points, we need to use more iterations to ensure estimation reliability.

B. Future Directions

Future research is needed to better integrate data quality analysis with the way in which data sets are analyzed by applications. We gave an example of how this can be done in the context of imputation. Certain metrics might be more important for certain types of data analysis than others, and this needs to be more properly elucidated. That way, the right data quality metrics can be emphasized depending on the specific analytics task. The problem is of increasing importance as the complexity of analysis of the application becomes greater.

Ideally, the data quality analysis should be integrated with the application so that the data quality checks are customized to both the types of analysis being done on the application as well as the specific problem domain.

Another area of research is improving the user interface to data quality systems. Providing a programmatic application programming interface (API) using functions or methods in a language such as Python is a good solution for users with programming skills. For users without programming skills, other user interfaces are needed providing graphical features as well as natural language understanding and possibly speech recognition as well.

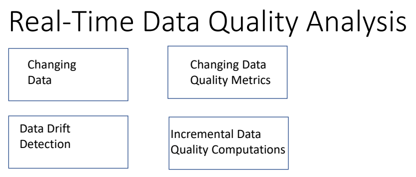
Real-Time Data Quality Analysis
Changing Data
Changing Data Quality Metrics
Data Drift Detection
Incremental Data Quality Computations

Fig. 2: Important features for real-time data quality analysis.

III. Efficient Analysis of Changing Data Sets

Figure 2 depicts important features for real-time data quality analysis. Since data quality systems often perform a significant number of data quality checks over large data sets, computational overhead can be significant. Thus, performance and scalability are quite important. One approach for improving efficiency of data quality computations on changing data sets is to perform incremental computations in which results on previous data sets are used to determine the performance on new data sets. When a previous data set $d1$ is updated with a new data set $d2$, we may be able to avoid performing at least some computations over parts of the data set which have not changed. In order for this approach to work, the data quality system needs to be aware of the previous computations which have been performed on $d1$ as well as the changes between $d1$ and $d2$. The data quality system can then focus on performing computations on the deltas between between $d1$ and $d2$ and combine these new computations with the earlier ones performed on $d1$ to obtain data quality metrics for $d2$.

There are also computations which are common across different data quality metrics. A global optimizer can consider all of the data quality metrics that need to be performed and identify the common computations among the data quality metrics so that the common computations only need to be performed once and the results can be shared across the data quality metrics having common computations.

We provide different types of checks for univariate and multivariate data. For multivariate data, we check for relationships among different variables. This can include determining how different variables are correlated, determining sign relationships between different variables, determining if one variable is (almost always) greater than (or less than) another variable, determining if two variables are equal, etc.

One of the challenges with comparing multivariate data is that the number of comparisons can grow exponentially with the number of variables being compared to each other. Therefore, it is often important to limit the number of comparisons of multiple variables to prevent the overhead from blowing up.

A key issue in data analysis is that the data may be constantly changing. As the data change, the data quality metrics can change as well. As data are updated, data quality metrics need to be updated in an intelligent way. If they are updated too frequently in response to any new data, the overhead for maintaining the metrics might be prohibitively high. If, on the other hand, the metrics are not updated very frequently, the data quality metrics can become outdated and stale. We have developed methods for maintaining data quality metrics which provide a good balance between maintaining updated metrics and not consuming too much overhead.

A key aspect of our data quality system is that we define multiple metrics which are changing over time. As a simple example, consider a metric which determines missing values in a data set, check_na_columns. If no new data are streaming in, then check_na_columns only has to be run on a data set once. If new values are constantly being streamed in, then check_na_columns needs to be run on the new data values. One option is to run check_na_columns on the entire updated data set when new data are received. This results in redundant calculations, because previous checks will have determined proper values which can be re-used. It is thus possible to improve computational efficiency by only determining missing values in newly received data and combining this with output from previous runs of check_na_columns. Similar techniques can be applied to incrementally calculate metrics such as mean, variance, and standard deviation.

It is often important to perform data analysis computations such as regression analysis. There has been past work in developing incremental approaches for performing linear regression and logistic regression [24], [25], [26].

[1] describes a data quality system used by Amazon. The paper describes incremental computations for a number of metrics, including size, compliance, completeness, uniqueness, entropy, mutual information, and predictability.

While our system provides incremental computations for a number of different data quality metrics, it goes beyond past work in defining new metrics for evolving data sets which are changing over time. For changing data sets, it is desirable to define new metrics which are applicable to constantly changing data rather than just using existing metrics. Consider a data quality metric $d1$. This metric could represent a number of possibilities, including missing data, finding low-variance variables, averages, standard deviations, medians, checks for constant values, non-repeating values, repeating values, most occurring values, duplicate values across columns, duplicate rows, etc.

If the data are being streamed in, we may not want to apply these checks monolithically across all existing data values. Instead, we may want to look at these data quality checks applied to certain windows of the data. We thus allow data quality checks to be defined across specific windows of a data set. Here are some of the options that we provide for defining these data quality metrics:

- Apply for a window defined between a start_time and an end_time
- Apply to each batch of new data which is received. An aggregate figure, $d1(batch1)$, is computed for each batch of data batch1. We then plot the values of $d(batch1)$ with $d(batch1)$ on the y-axis and time values on the x-axis.
- Different time periods can be given different weights for calculating data quality metrics. In general, more recent data points can be assigned higher weights than less recent data points for assessing data quality metrics. Exponential weighting is one possibility, although not the only one. Note that each sample can be assigned a different weight based on its time. Another approach is to group samples by time intervals and assign the same weight to a set of samples belonging to the same group.
- In some cases, older values can be ignored entirely. Different algorithms can be applied to determine which older values should be ignored.

We provide data quality metrics which are parameterized by time. The metrics can be calculated and visualized over any range of data points.

Our system gives complete analyses of data sets, including constraints which are applicable to a data set. Examples of constraints include null values not exceeding a threshold, average (standard deviation, variance, median, etc.) falling within a certain range, two columns having a certain mathematical relationship or correlation, etc. With real-time streaming data, applicability of constraints is not a static, fixed property. A constraint may be applicable at one particular time, but not for new data which are being received. We thus provide analysis of constraints across multiple time scales. The applicability of a constraint or set of constraints is thus dynamic and expected to vary over time.

As an example. consider a function:

```
related(feature1, feature2, start_time,
        end_time)
```

This function returns a value between 1 and -1 indicating correlation between feature1 and feature2 for a specific time interval. Our system maintains related values over several different time intervals. We flag time intervals, ti, where related shows anomalous behavior. An example of anomalous behavior would be the value of related changing to values not seen before. Those time intervals are surfaced to the user as anomalous. Anomalous time intervals can be left out or assigned a lower weight in calculating overall quality assessments.

As another example, consider a function:

```
check_na_columns(df, start_time1,
                 end_time)
```

which checks columns for missing values over specific time ranges. This function can be used to detect which parts of the data should be assigned higher weights in calculating data quality metrics. For example, if check_na_columns indicates

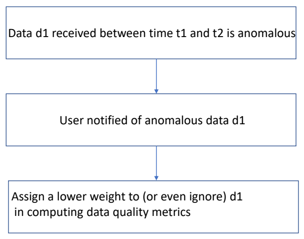
Data d1 received between time t1 and t2 is anomalous
User notified of anomalous data d1
Assign a lower weight to (or even ignore) d1 in computing data quality metrics

Fig. 3: Anomalous regions are assigned lower weights in computing data quality metrics.

an abnormal proportion of missing values in an interval, it may be appropriate to assign less weight to the interval in calculating data quality metrics.

It should be noted that a higher proportion of missing values does not necessarily mean that a time interval should be assigned a lower weight. In many cases, we are looking for an anomalous proportion of null values. That could indicate an abnormally high or low proportion of missing values. If a particular data interval has an unusually low proportion of missing values compared to other intervals, this could indicate an anomaly which would mean that the interval should be assigned a lower weight than other intervals with a proportion of missing values closer to the mean.

In general, we try to identify anomalous regions of data when computing data quality metrics. Those anomalous regions of data are assigned lower weights in computing overall data quality metrics (Figure 3). Anomalous regions can be defined in a number of different ways. For example, an anomalous region could be a data region in which the proportion of missing values exceeds the average proportion of missing values by a threshold. Alternatively, an anomalous data region may comprise a data region in which the proportion of missing values differs from a proportion of null values for other data by a threshold. A wide variety of other criteria can be used for identifying anomalous data regions. For example, an anomalous data region might correspond to a region where a proportion of data values which are outliers exceeds a threshold. In order to determine outliers, ranges of values can be specified. An outlier would fall outside the range of specified values. User-defined functions can also be provided to determine outliers.

An anomalous data region might also correspond to a region having different statistical properties from other data regions. For example, an anomalous data region might have a mean, median, mode, variance, and/or standard deviation which differs from other data by a threshold.

We assign aggregate data quality metrics based on the weights. The aggregate data quality metric would be the weighted sum of data quality metrics for each region, where anomalous regions are assigned lower weights. Default weights can be assigned to regions without anomalies.

Our data quality system may need to maintain state information about previous computations. For example, suppose that it has computed data quality metrics for a data set $d1$. The state information corresponding to these data quality metrics (example: proportion of null values in d1, average, mean, standard deviation for all or part of d1, etc.) is maintained as the system computes data quality metrics for a new version of the data set, $d2$. In general, with an evolving data set, it is desirable to maintain data quality metrics for past versions of the data set.

This state information can be maintained in a file system or database. For situations in which it is not feasible to use a file system or database to maintain state information, the state information can be passed between a client program accessing our data quality system and the data quality system via our API (application programming interface). That way, our system generates the state variables, and once the state variables are created, they are passed between our data quality system and client programs via our API.

There are often trade-offs between efficiency and accuracy of data quality metrics. Achieving the most accurate and up-to-date data quality metrics at all times can have prohibitive overhead. Thus, appropriate trade-offs have to be made in providing reasonable data quality estimates while not using too many computational resources. Intelligent choices can be made in both the frequency for recalculating data quality metrics and selecting the most appropriate data quality metrics to recalculate.

The problem is exacerbated by large data sets, because larger data sets generally consume more computational resources. The number of data quality metrics users are interested in can also affect performance. If a user is interested in tracking a large number of data quality metrics, the overhead will generally be higher than for tracking a smaller number of data quality metrics.

A key element of our approach is to maintain information on performance of different data quality metrics as a function of data size and possibly other characteristics of the data. We maintain historical data on the performance of our data quality metrics. As new data sets are analyzed, we maintain persistent information on performance and other execution characteristics in a history recorder (HR). The HR is analyzed to better understand the performance of our data quality metrics. The HR maintains information on execution of data quality metrics. When a data quality metric function is executed, the HR records information, such as:

- Data quality function name and parameters
- Sizes/dimensions of data sets being analyzed
- CPU time consumed by function execution
- Wall clock time consumed by function execution
- I/O and/or network overhead if significant
- Hardware and software used to execute the function

For tabular data, the HR maintains information such as number of rows, number of columns, as well as information on data types of columns (e.g. numerical, string, categorical, etc.). The HR allows our system to create performance profiles for all of the data quality metrics of interest. For a given data set and data set size, we can thus estimate the overheads for different data quality metrics performed on that data set.

The HR maintains information both from running benchmarks we have developed to test our system as well as from past execution on real workloads from customers and other users.

We also maintain information on how data quality metrics change with changes in the data itself. This information can be used to predict how much data quality metrics would be expected to change in response to new data. Such change predictions can be made using simple calculations or more complex machine learning models.

We focus on limiting invocations of data quality metrics with high overhead. Data quality metrics with lower overhead can be executed more frequently. We also focus on both the rate of change of data and the data quality metrics themselves. If the rate of change is higher, then data quality metrics need to be recalculated more frequently.

As more data are received, we can estimate using simple calculations and predictive models how much data quality metrics are expected to change. It is more important to recalculate data metrics which are expected to change the most.

When we recalculate our data quality metrics, we have updated information on how much they have changed in response to changes in the data. This information can be used to update predictive models on how data quality metrics change with changes in the input data. This way, as our system executes, it gets smarter over time in predicting the behavior of data quality metrics and more accurate in computing performance metrics (with limited computational resources) over time.

Users have the ability to assign an importance score to data quality metrics. A higher importance score indicates that it is more important to have the most up-to-date scores for a data quality metric.

Thus, we have the following criteria for ranking data quality metrics:

- Computational overhead
- Rate of change (as a function of changing input data)
- Importance score provided by users

Our system ranks data quality metrics by these criteria. Based on the rank of a data quality metric, we assign a relative frequency for recalculating each data quality metric. Higher ranked data quality metrics are recalculated more frequently.

Our system has a limited budget for computing data quality metrics. Computational resources are not unbounded. Based on the computational resources available, our system computes data quality metrics based on the relative frequencies assigned in the ranking process.

Note that the relative frequencies are dynamic and may change over time. This could occur, for example, if the rate of change for one or more data quality metrics changes over time.

Figure 4 depicts the algorithm for prioritizing data quality metrics to update. In Step 1, the system maintains past statistics on computational overhead, o, and rate of change, f,

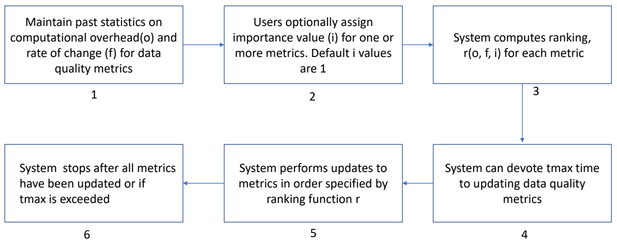
Maintain past statistics on computational overhead(o) and rate of change (f) for data quality metrics
1
Users optionally assign importance value (i) for one or more metrics. Default i values are 1
2
System computes ranking, r(o, f, i) for each metric
3
System can devote tmax time to updating data quality metrics
4
System performs updates to metrics in order specified by ranking function r
5
System stops after all metrics have been updated or if tmax is exceeded
6

Fig. 4: Prioritizing data quality computations.

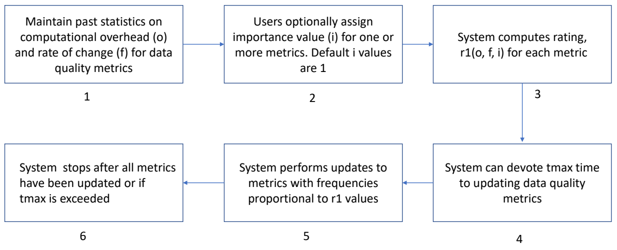
Maintain past statistics on computational overhead (o) and rate of change (f) for data quality metrics
1
Users optionally assign importance value (i) for one or more metrics. Default i values are 1
2
System computes rating, r1(o, f, i) for each metric
3
System can devote tmax time to updating data quality metrics
4
System performs updates to metrics with frequencies proportional to r1 values
5
System stops after all metrics have been updated or if tmax is exceeded
6

Fig. 5: Prioritizing data quality computations while allowing all data quality metrics to be updated.

for different data quality metrics. f represents the magnitude with which a data quality metric changes as new data are received. This step is continuously executing over time.

In Step 2, users optionally assign an importance value, i, for one or more metrics. Default i values are 1, for situations in which users do not provide an importance value.

The system computes ranking $r(o, f, i)$ for each metric in Step 3. A variety of different functions can be used for r. One such example is the following. A rating is assigned using a function:
rating $= a * o + b * f + c * i$ where:

- a is a negative constant, and b, and c are positive constants
- o is overhead for computing the data quality metric
- f is the rate of change of the data quality metric as new data are received
- i is the importance of the data quality metric provided by a user (default value of 1 if not provided)

The ranking function r orders data quality metrics in decreasing order by ratings.

The system can devote tmax time to updating data quality metrics (Step 4). In Step 5, the system performs updates to metrics in an order specified by ranking function r. The system stops performing updates to data quality metrics after all metrics have been updated or if tmax is exceeded (Step 6).

One of the issues with the algorithm in Figure 4 is that if the system always runs out of time to compute data quality metrics (i.e. tmax is always exceeded), low-ranked data quality metrics might never be updated. The algorithm depicted in Figure 5 avoids this problem.

In Step 1, the system maintains past statistics on computational overhead, o, and rate of change, f, for different data quality metrics. f represents the magnitude with which a data quality metric changes as new data are received. This step is continuously executing over time.

In Step 2, users optionally assign an importance value, i, for one or more metrics. Default i values are 1, for situations in which users do not provide an importance value.

The system computes a rating $r1(o, f, i)$ for each metric in Step 3. A variety of different functions can be used for $r1$, including the one defined above for r. $r1$ values represent relative frequencies for which the system should update a data quality metric. All $r1$ values should be positive. In order to ensure that all data quality metrics are computed at least some of the time, even if tmax is always exceeded, the ratio between highest and lowest values should not be too high.

The system can devote tmax time to updating data quality metrics (Step 4). In Step 5, the system performs updates to metrics. Each metric is updated with a frequency proportional to its r1 value. The system stops performing updates to data quality metrics after all metrics have been updated or if tmax is exceeded (Step 6).

IV. Summary and Conclusion

This paper has presented an overview of systems for analyzing data quality. We have described key features in data quality systems and presented new algorithms for efficiently analyzing quality in changing data sets.

Future research is needed in correlating data quality metrics with analyses being performed on the data sets. This is particularly important when the data are being used for training, validation, and test sets in machine learning. Progress in this area would allow considerably better integration of the data quality systems with the data analytics applications and machine learning algorithms being used to analyze the data.

References

[1] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Biessmann, and A. Grafberger, "Automating large-scale data quality verification," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 1781–1794, 2018.

[2] E. Breck, N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data infrastructure for machine learning," in *SysML Conference*, 2018.

[3] S. Shrivastava, D. Patel, A. Bhamidipaty, W. M. Gifford, S. A. Siegel, V. S. Ganapavarapu, and J. R. Kalagnanam, "Dqa: Scalable, automated and interactive data quality advisor," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 2913–2922.

[4] E. Caveness, P. S. GC, Z. Peng, N. Polyzotis, S. Roy, and M. Zinkevich, "Tensorflow data validation: Data analysis and validation in continuous ml pipelines," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 2793–2796.

[5] S. Schelter, S. Grafberger, P. Schmidt, T. Rukat, M. Kiessling, A. Taptunov, F. Biessmann, and D. Lange, "Differential data quality verification on partitioned data," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1940–1945.

[6] L. L. Pipino, Y. W. Lee, and R. Y. Wang, "Data quality assessment," *Communications of the ACM*, vol. 45, no. 4, pp. 211–218, 2002.

[7] E. Paikari and A. Van Der Hoek, "A framework for understanding chatbots and their future," in *2018 IEEE/ACM 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 2018, pp. 13–16.

[8] S. Bieliauskas and A. Schreiber, "A conversational user interface for software visualization," in *2017 ieee working conference on software visualization (vissoft)*. IEEE, 2017, pp. 139–143.

[9] A. Wachtel, J. Klamroth, and W. F. Tichy, "Natural language user interface for software engineering tasks," in *Proceedings of the International Conference on Advances in Computer-Human Interactions (ACHI)*, vol. 10, 2017, pp. 34–39.

[10] J. N. Wulff and L. Ejlskov, "Multiple imputation by chained equations in praxis: Guidelines and review." *Electronic Journal of Business Research Methods*, vol. 15, no. 1, 2017.

[11] Y. Duan, Y. Lv, Y.-L. Liu, and F.-Y. Wang, "An efficient realization of deep learning for traffic data imputation," *Transportation research part C: emerging technologies*, vol. 72, pp. 168–181, 2016.

[12] C. Gautam and V. Ravi, "Data imputation via evolutionary computation, clustering and a neural network," *Neurocomputing*, vol. 156, pp. 134–142, 2015.

[13] C.-Y. Cheng, W.-L. Tseng, C.-F. Chang, C.-H. Chang, and S. S.-F. Gau, "A deep learning approach for missing data imputation of rating scales assessing attention-deficit hyperactivity disorder," *Frontiers in psychiatry*, vol. 11, p. 673, 2020.

[14] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.

[15] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Brazilian symposium on artificial intelligence*. Springer, 2004, pp. 286–295.

[16] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proceedings of the 2007 SIAM international conference on data mining*. SIAM, 2007, pp. 443–448.

[17] A. Gelman and J. Hill, *Missing-data imputation*, ser. Analytical Methods for Social Research. Cambridge University Press, 2006, p. 529–544.

[18] S. Moritz, A. Sardá, T. Bartz-Beielstein, M. Zaefferer, and J. Stork, "Comparison of different methods for univariate time series imputation in r," *arXiv preprint arXiv:1510.03924*, 2015.

[19] S. Moritz and T. Bartz-Beielstein, "imputets: time series missing value imputation in r." *R J.*, vol. 9, no. 1, p. 207, 2017.

[20] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *Scientific reports*, vol. 8, no. 1, pp. 1–12, 2018.

[21] Z. C. Lipton, D. C. Kale, and R. Wetzel, "Modeling missing data in clinical time series with rnns," *arXiv preprint arXiv:1606.04130*, 2016.

[22] Q. Ma, S. Li, L. Shen, J. Wang, J. Wei, Z. Yu, and G. W. Cottrell, "End-to-end incomplete time-series modeling from linear memory of latent variables," *IEEE transactions on cybernetics*, 2019.

[23] N. Zhou, P. Dhaval, A. Iyengar, S. Shrivastava, and A. Bhamidipaty, "A verifiable imputation analysis for univariate time series and enabling package," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020.

[24] A. L. Strehl and M. L. Littman, "Online linear regression and its application to model-based reinforcement learning," in *Advances in Neural Information Processing Systems*, 2008, pp. 1417–1424.

[25] H. B. McMahan and M. Streeter, "Open problem: Better bounds for online logistic regression," in *Conference on Learning Theory*, 2012, pp. 44–1.

[26] A. Dhurandhar and M. Petrik, "Efficient and accurate methods for updating generalized linear models with multiple feature additions," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 2607–2627, 2014.