Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning

Mohammad Mahdavi
Technische Universität Berlin
mahdavilahijani@tu-berlin.de

Ziawasch Abedjan
Technische Universität Berlin
abedjan@tu-berlin.de

ABSTRACT

Traditional error correction solutions leverage handmaid rules or master data to find the correct values. Both are often amiss in real-world scenarios. Therefore, it is desirable to additionally learn corrections from a limited number of example repairs. To effectively generalize example repairs, it is necessary to capture the entire context of each erroneous value. A context comprises the value itself, the co-occurring values inside the same tuple, and all values that define the attribute type. Typically, an error corrector based on any of these context information undergoes an individual process of operations that is not always easy to integrate with other types of error correctors. In this paper, we present a new error correction system, Baran, which provides a unifying abstraction for integrating multiple error corrector models that can be pretrained and updated in the same way. Because of the holistic nature of our approach, we generate more correction candidates than state of the art and, because of the underlying context-aware data representation, we achieve high precision. We show that, by pretraining our models based on Wikipedia revisions, our system can further improve its overall precision and recall. In our experiments, Baran significantly outperforms state-of-the-art error correction systems in terms of effectiveness and human involvement requiring only 20 labeled tuples.

PVLDB Reference Format:
Mohammad Mahdavi and Ziawasch Abedjan. Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning. *PVLDB*, 13(11): 1948-1961, 2020.
DOI: https://doi.org/10.14778/3407790.3407801

1. INTRODUCTION

Data cleaning is one of the most important but time-consuming tasks for data scientists [14]. The data cleaning task consists of two major steps: (1) *error detection* and (2) *error correction* (i.e., data repairing). The goal of error detection is to identify wrong data values [3]. The goal of error correction is to fix these wrong values to correct values [37].

Proceedings of the VLDB Endowment, Vol. 13, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3407790.3407801

Traditional data cleaning systems follow the *preconfiguration paradigm*, where the user has to provide upfront a correct and complete set of rules and parameters, such as functional dependencies and desired data patterns [3, 37, 11, 13, 48]. For most non-expert users, this is a major impediment as they need to know both the dataset and the data cleaning system upfront to be able to configure the systems properly [3, 27, 48]. Recently, we have promoted the *configuration-free paradigm*, where user supervision is in the form of annotating a few data values [27]. In fact, the user only provides a few examples of data errors and their correction and the system learns to generalize the error detection/correction operation. This paradigm is more suitable for three scenarios: (1) the dataset is novel and data constraints are not available, (2) users are domain experts who are not adept at generating the right rules/parameters for complex data cleaning systems, or (3) they prefer to additionally annotate a few data values to improve the performance of traditional data cleaning systems.

Due to the promises of our configuration-free error detection system, Raha [27], here, we study the application of the same intuition for the subsequent task of error correction. Configuration-free error correction is much more challenging than its pendant for error detection. Error detection is a binary classification task with two possible classes for each value inside a dataset, namely clean or dirty. However, the class space for error correction is infinite as, theoretically, any possible string could be a correction for an erroneous value. Thus, there is always a high chance of choosing a wrong correction from this infinite search space, which affects the precision in error correction. To tackle this issue, previous approaches confine the space of possible corrections by only considering the values that reside inside the dataset itself [48, 37] or trusted external sources [12]. These pruning strategies limit the error correction recall as the actual correction may not exist in the provided resources.

Our approach. We propose a new configuration-free error correction system, Baran, that achieves both high precision and recall. For this purpose, Baran combines so-called error corrector models that capture the context information surrounding a data error. Given a dirty dataset with already detected data errors, Baran fixes data errors with a novel two-step formulation of the error correction task. First, each error corrector model generates an initial set of potential corrections for each detected data error. This step particularly increases the achievable recall bound of error correction. Then, Baran ensembles the output of these models into one final correction for each data error in a semi-supervised

manner. In fact, Baran iteratively asks the user to label a tuple with data errors from the dataset and leverage the provided user corrections to update the corrector models, incrementally. Training one classifier per column, Baran identifies the most accurate correction among all the proposed correction candidates for each data error. This step particularly preserves the high precision of error correction.

To design a complete set of error corrector models, we need to leverage all *data error contexts*. In principle, to fix a data error in a given dataset, we can leverage three data error contexts:

1. **Value of data errors.** Some data errors can be fixed by only taking the erroneous value itself into consideration. In this case, we just need to transform the erroneous data value into the correct value. For example, an erroneous date value "16/11/1990" can be fixed by transforming it into the correct format, i.e., "16.11.1990".
2. **Vicinity of data errors.** The correction of some data errors requires information on their vicinity, i.e., information about other clean data values in the same data row. For example, we cannot fix the erroneous value "Paris" in column *Capital* with the data value itself as "Paris" is not an erroneous value on its own. But, if we check its clean neighboring data values in the same tuple and observe "Germany" in column *Country*, then we can fix "Paris" to "Berlin" to make it consistent with its vicinity.
3. **Domain of data errors.** Fixing some data errors needs domain information, i.e., information about other clean data values inside the same column. For example, to fix an outlier temperature value, we can use other clean values inside the same column *Temperature* to impute the correct value.

As the examples above show, each data error context can be leveraged in an entirely different way to generate a correction candidate. These correction procedures are not easy to integrate although they can independently lead to the same correction candidate.

Another benefit of our two-step task formulation is that error corrector models can be pretrained for expanding the space of possible corrections without requiring the user to do it manually. In fact, Baran can also leverage transfer learning, which is the act of gaining knowledge from one task and then applying this knowledge to a different but related task [31]. A natural fit for transfer learning is when training data is scarce and expensive in one domain while it is widely available in a similar domain. In this case, the learning model can be pretrained on the related domain and then be fine-tuned on the current dataset [15]. The same situation holds in our example-driven error correction task. In addition to the clean values of the current dataset, we need further correction candidates to improve the achievable recall bound. These correction candidates have to be either provided by the user or learned from external sources. While user labeling is expensive on the current dataset, value-based corrections are widely available in external sources, such as the Wikipedia page revision history. Therefore, we can extract value updates from external sources that serve as additional correction examples to pretrain the error corrector models. The pretraining aims at learning corrections for common typos and mistakes, such as the wrong usage of "Holland" instead of "Netherlands". The pretrained models can then be fine-tuned on the dataset at hand with the help of user labels. Thus, Baran can also learn dataset-specific preferences and select the best correction among all correction candidates. We leverage the Wikipedia page revision history as a rich source that contains terabytes of human-committed revisions. In practice, any other source of data updates can be leveraged the same way.

Challenges. To design such an error correction system, we addressed the following questions:

- How can we design uniformly and incrementally updatable error corrector models that can propose various correction candidates for different data errors?
- How can we formulate a semi-supervised classification task for error correction that effectively and efficiently identifies the actual correction among many candidates?
- How can we extract value-based corrections from publicly available revision data histories and pretrain the error corrector models with them?

Contributions. To address these challenges, we make the following contributions:

- We propose a new configuration-free error correction system, named Baran, that fixes data errors without any user-given rules or parameters (Section 3).
- We propose simple, general, and incrementally updatable error corrector models (Section 4). These models leverage the value, the vicinity, and the domain contexts of data errors to propose correction candidates for any data error.
- We design a novel binary classification task that benefits from a dense feature representation and a tuple sampling approach that selects the most informative tuples for generalizing user corrections (Section 4). The task effectively and efficiently ensembles the output of the error corrector models into one final correction for each data error.
- We propose an approach to segment and align the Wikipedia page revision history to collect additional training data for pretraining value-based models (Section 5).
- We conduct extensive experiments to evaluate our system in terms of effectiveness, efficiency, and human involvement (Section 6). As our experiments show, Baran significantly outperforms 4 recent error correction systems on 7 well-known datasets.

2. FOUNDATIONS

We first formally define the problem statement and then review state-of-the-art solutions and their limitations.

2.1 Problem Statement

The error correction problem is the task of replacing detected data errors with the corresponding correct values. Let $d = \{t_1, t_2, ..., t_{|d|}\}$ be a relational dataset of size $|d|$, where each t_i denotes a tuple. Let $A = \{a_1, a_2, ..., a_{|A|}\}$ be the schema of dataset d, with $|A|$ attributes. We denote $d[i, j]$ as the data value in tuple t_i and attribute a_j. Let d^* be the ground truth of the same dataset with only clean values.

A data error is a data value inside a dataset that deviates from the unavailable ground truth. We consider both major *syntactic* and *semantic* data error categories [27] and their subsequent data error types, such as missing values, typos, formatting issues, and violated attribute dependencies [36]. Let $E = \{d[i, j] \mid d[i, j] \neq d^*[i, j]\}$ be the set of detected data errors, as the output of an upstream error detection step. The quality of the detected data errors generally can impact the downstream error correction step. Similar to prior work, we discuss our approach based on the assumption that the

Table 1: A dirty dataset d and its cleaned version d^*.

ID	Name	Address
1	H	5th Str
2	Hana	-
3	Gandom	7th Street
4	Chris	9th Str

ID	Name	Address
1	Hana	5th Street
2	Hana	5th Street
3	Gandom	7th Street
4	Christopher	9th Street

error detection step was correct and complete [37, 48]. In our experiments (Section 6.6), we also evaluate the performance of our system on top of Raha [27], which delivers imperfect error detection results.

Given as input a dirty dataset d, the set of detected data errors E, and a labeling budget θ_{Labels} to annotate tuples, the goal is to fix as many detected data errors as possible.

2.2 State of the Art

State-of-the-art error correction systems follow the preconfiguration paradigm. They need the user to preconfigure the system with integrity rules (e.g., Holistic [11]), statistical parameters (e.g., SCARE [48]), or both (e.g., HoloClean [37]). Their idea is to minimally swap the values in the dataset with respect to a cost function, such as the number of value changes, until the dataset becomes consistent with respect to the integrity rules and statistical likelihoods. In the lack of redundancy in data and a correct and complete set of predefined user-given rules and parameters to define consistency, these systems struggle to achieve both high precision and recall across many datasets.

Example 1 (Limitations of the existing systems). Table 1 shows a dirty dataset with its already detected data errors marked in red. The goal is to fix these data errors and generate the depicted cleaned dataset d^*. The aforementioned systems would not be able to effectively fix these data errors because of the mentioned limitations. First, because the dataset does not have a high degree of redundancy, these systems cannot find the correction of the data error "9th Str", as the correct value "9th Street" does not appear anywhere in the dataset. Second, there are no general integrity rules that we can define on this dataset. The functional dependency *Name* $\rightarrow$ *Address* is still not enough to fix the data error "5th Str" because tuples 1 and 2 do not have the same value in column *Name*. □

3. BARAN OVERVIEW

Figure 1 illustrates the workflow of Baran. Given a dirty dataset with marked data errors and an optional revision dataset, Baran fixes data errors with the help of user feedback and returns a cleaned version of the dataset. The workflow consists of an online phase and an optional offline phase.

3.1 Online Phase

The online phase aims to fine-tune the error corrector models on the current dataset and ensemble them to fix data errors. If the error corrector models have been already pretrained, Baran incrementally updates them. Otherwise, Baran trains the error corrector models from scratch on the current dataset. Baran conducts the following steps in each iteration of the online phase.

Steps 1 and 2: Sampling and labeling a tuple. In each iteration, Baran samples one tuple to be labeled by the user. Thus, the total number of iterations is bound by the user labeling budget θ_{Labels}. Then, Baran asks the user to fix the marked data errors in the sampled tuple. To optimally leverage the limited number of user labels, the set of sampled tuples should cover as many data error types as possible inside the dataset. We detail these steps in Section 4.2.1.

Step 3: Fine-tuning error corrector models. Baran updates error corrector models based on the user-corrected data errors. In Section 4.1, we define a unified model for all different error corrector models so that we can incrementally update all of them in the same way with every new user-corrected data error.

Steps 4 and 5: Generating correction candidates and features. Each error corrector model proposes various correction candidates for each data error. We need to represent the fitness of each particular correction candidate for each particular data error. As detailed in Section 4.2.2, Baran generates a feature vector that represents the mutual fitness of each pair of a data error and a correction candidate.

Steps 6 and 7: Training and applying classifiers. Since the user fixes data errors in the sampled tuples, we have the actual correction of a small subset of data errors. Thus, we can design a binary classification task, where a classifier decides whether a correction candidate is the actual correction of a data error or not. In Section 4.2.2, we discuss how Baran trains a binary classifier per column based on the feature vectors and the user labels. Baran applies the trained classifiers to predict the final correction for the rest of the data errors.

3.2 Offline Phase

In the optional offline phase, Baran pretrains value-based error corrector models by processing any external source that provides value-based corrections.

Step 1: Extracting training data. An external dataset with value-based corrections may not necessarily be fully structured. That is why we need to first extract value-based corrections from non/semi-structured external sources, as discussed in Section 5.

Step 2: Pretraining error corrector models. Baran pretrains value-based error corrector models based on the extracted training data. We detail this step in Section 5.

4. THE ERROR CORRECTION ENGINE

We first introduce our error corrector models and how they can be fine-tuned. Then, we explain our approach to ensemble the output of these models and fix data errors.

4.1 Error Corrector Models

An error corrector model is any algorithm that can propose a correction to a data error based on a logic that uses an error context. The set of error corrector models should be ideally complete and contain correct and automated correctors. In fact, their combination should be able to fix all data errors (completeness) accurately (correctness) without any user involvement (automation). Choosing error corrector models under these three requirements simultaneously is not trivial because there is typically a trade-off among them. For example, a data scientist may assess data errors of a dataset and then write a script to transform wrongly formatted date values "dd/mm/yyyy" to the format "dd.mm.yyyy". Although this error corrector model is accurate, it involves the user (not automated) and may not fix other potential

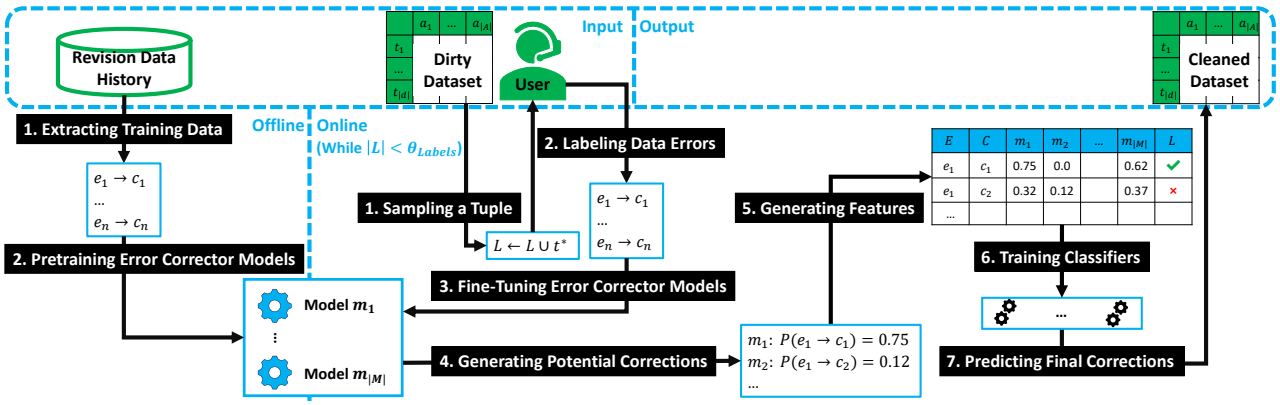


Figure 1: The workflow of Baran.

data errors (not complete). Therefore, we have to handle the natural trade-off of correctness, completeness, and automation in designing error corrector models.

We address this trade-off by first ignoring the precision of the error corrector models. Our error corrector models are designed to propose as many potential corrections as possible in the first place. This way, we increase the achievable recall bound by automatically generating a set of correction candidates. Many of these correction candidates might be irrelevant. However, we will avoid false positives later as well, when we train classifiers on top of these automatically proposed correction candidates.

To this end, we design a set of error corrector models, each of which leverages one context of a data error in the form of a heuristic. To keep the error corrector models simple, general, and incrementally updatable, we define an error corrector model m formally as the conditional probability

$$P(c|e_m) = \frac{\text{count}(c|e_m)}{\text{count}(e_m)}, \quad (1)$$

where e_m is a context of the data error e that the model m uses; $\text{count}(e_m)$ is the number of times that the data error context e_m is observed; and $\text{count}(c|e_m)$ is the number of times that the context e_m is leveraged to fix the data error e to the correction c. The intuition is that the more often a data error context e_m is leveraged to fix a data error e to a correction c, the more it is likely that c will be the actual correction of data error e. Note that we can incrementally update this model by storing $\text{count}(c|e_m)$ and $\text{count}(e_m)$ for each pair of data error context e_m and correction c. This definition of error corrector models is abstract and needs to be implemented for each type of models accordingly.

While we propose a default and general set of error corrector models, which are applicable on a wide range of datasets, the set of models can be extended with optional custom error corrector models provided by the user. In particular, the user can optionally implement data constraints in the form of error corrector models and incorporate them into Baran. Since Baran considers the error corrector models as black boxes, it generalizes the previous error correction aggregators [35, 37].

4.1.1 Value-Based Models

Value-based error corrector models learn to fix data errors e using only the erroneous value itself [19]. Here, data error e and its context $e_{val} = d[i, j]$ are identical. Whenever an erroneous value e_{val} is corrected to a value e^*_{val}, Baran updates the value-based error corrector models by encoding the erroneous value and its correction operation.

Erroneous value encoding. Abstracting erroneous values is an essential technique for training value-based error corrector models. There are different levels of abstraction to encode data values, such as abstracting the character category or string length. We leverage two simple and general encoders. Our first encoder is the *identity encoder*, which encodes the value with its original characters. This encoding is suitable for fixing semantic data errors. For example, to fix the wrong country name "Holland" to "Netherlands", our value-based error corrector models need to see the exact wrong value "Holland". Our second encoder is the *Unicode encoder*, which encodes each character of a data value with its equivalent Unicode category [47]. For example, an uppercase character will be replaced with its category symbol "<Lu>" and a number will be replaced with its category symbol "<Nd>". This encoding enables the value-based models to learn syntactic error corrections faster by generalizing the syntax of data errors.

Example 2 (Erroneous value encoding). Considering the erroneous value e_{val} = "16/11/1990" and the corrected value e^*_{val} = "16.11.1990", we have two methods to encode this erroneous value. The identity encoder encodes the erroneous value with its original characters: Whenever an erroneous value is equal to "16/11/1990", a potential correction operation could be to replace "/" with ".". On the other hand, the Unicode encoder encodes this erroneous value by abstracting its characters to their Unicode category: Whenever an erroneous value is in the format "<Nd><Nd><Po><Nd><Nd><Po><Nd><Nd><Nd><Nd>", a potential correction operation could be to replace "/" with ".". While the first encoding is very accurate, the second one improves the overall recall. □

Correction operation encoding. After encoding the erroneous value, we need to encode the required correction operations. In general, there are four kinds of correction operators that can be applied on any erroneous value.

1. *Remover operator.* This operator removes substrings. For example, if we want to fix the erroneous value "U.S." to the value "US", the remover operator can remove ".".
2. *Adder operator.* The adder operator adds substrings. For example, if we want to fix the erroneous value "US" to the value "U.S.", the adder operator can add ".".

3. *Replacer operator.* The replacer operator both removes and adds substrings simultaneously. For example, if we want to fix the erroneous value "16/11/1990" to the value "16.11.1990", the replacer operator can remove "/" and, instead of it, add ".".
4. *Swapper operator.* The swapper operator substitutes the entire erroneous value with another value. While the previous value-based operators are suitable for syntactic errors, the swapper operator is useful for fixing semantic data errors. For example, if we want to fix the erroneous value "Holland" to the value "Netherlands", the swapper operator can substitute these values.

We train value-based models, each of which learns to perform one of these correction operations on an encoded erroneous value. With this set of operators, we can generate any value-based correction. Given a user-corrected data error, Baran calculates the difference of the erroneous value e_{val} and the corrected value e^*_{val} on the character level, according to the diff checking technique [22], and extracts operators.

Example 3 (Diff checking). Considering the erroneous value e_{val} = "Chris Edward NoLan" and the user-corrected value e^*_{val} = "Christopher Nolan" as the source and target character sequences, respectively, the output of the diff checker algorithm is as follows:

$$\text{diff}(e_{val}, e^*_{val}) = \begin{cases} \texttt{Add ``topher'' after ``Chris''.} \\ \texttt{Remove ``Edward ''.} \\ \texttt{Replace ``L'' with ``l''.} \end{cases}$$

Thus, we can update the value-based models of the four operators with this user-corrected example. □

Overall, we have $2 \times 4 = 8$ value-based error corrector models because of 2 erroneous value encoders and 4 correction operators. To implement the abstract definition of error corrector models in Equation 1, a value-based model considers the encoded erroneous value encode(e_{val}) as the context e_m and the correction operation o that has to be applied on this erroneous value as the correction c. Formally, $P(c|e_m) = P(o|\text{encode}(e_{val})) = \frac{\text{count}(o|\text{encode}(e_{val}))}{\text{count}(\text{encode}(e_{val}))}$.

Example 4 (Value-based models). Assume that we have a value-based error corrector model equipped with the identity encoder and the swapper operator. Assume that the model encounters the erroneous value "Holland" 5 times in different spots of the dataset, i.e., count(encode(e_{val})) = 5. Assume that the user fixes this data error to "Netherlands" 4 times, and to "HL" 1 time. Let us call these two swapping operations o_1 and o_2, respectively. Therefore, this value-based error corrector model proposes two correction candidates "Netherlands" with $P(o_1|\text{encode}(e_{val})) = 0.8$ and "HL" with $P(o_2|\text{encode}(e_{val})) = 0.2$ for any detected data error e that holds the value "Holland". □

4.1.2 Vicinity-Based Models

Vicinity-based error corrector models learn to fix data errors based on column relationships. A vicinity-based model proposes clean values of the active domain as potential corrections based on their relationship with clean data values of other columns. We limit all kinds of column relationships and correlations [4] to functional dependencies that have one attribute on their left-hand side. This way, we reasonably limit the exponential space of all the functional dependencies as these functional dependencies have been known to be more useful for data cleaning [32]. Similar to our error detection system Raha [27], we consider every $j_1 \rightarrow j_2$ to be a functional dependency for each pair of columns $\forall j_1 \neq j_2 \in [1, |A|]$. To implement the abstract definition of error corrector models in Equation 1, for each functional dependency $j_1 \rightarrow j_2$, a vicinity-based model considers the clean co-occurring value $d[i, j_1]$ in the vicinity context $e_{vic} = d[i, :]$ as the context e_m and the clean value $d[i, j_2]$ as the correction c. Formally, $P(c|e_m) = P(d[i, j_2]|d[i, j_1]) = \frac{\text{count}(d[i,j_2]|d[i,j_1]))}{\text{count}(d[i,j_1])}$. This conditional probability shows how often the left-hand-side value $d[i, j_1]$ determines the right-hand-side value $d[i, j_2]$.

Overall, we have $|A| \times (|A| - 1)$ vicinity-based error corrector models as we consider the functional dependencies from and to each attribute.

Example 5 (Vicinity-based models). Considering the functional dependency *Name* → *Address* in Example 1, a vicinity-based model learns that a name value "Gandom" must always have the address value "7th Street". Thus, the corresponding vicinity-based model proposes one correction candidate P("7th Street"|"Gandom") = 1.0 for any data error e in column *Address* that has the neighboring value "Gandom" in neighboring column *Name*. □

4.1.3 Domain-Based Models

Domain-based error corrector models learn to fix data errors using the existing values inside their columns. A domain-based model proposes the most relevant clean values from the active domain as potential corrections. As tuples inside a dataset are generally independent of each other, the order, distance, or neighborhood of values inside a column does not indicate any relevance. However, the frequency of clean values can be used as a signal to estimate their relevance. More frequent clean values inside a column are more likely to also be corrections for a data error in the same column. Thus, to implement the abstract definition of error corrector models in Equation 1, a domain-based model considers the domain context $e_{dom} = d[:, j]$ as the context e_m and each clean value inside this domain as the correction c. Formally, $P(c|e_m) = P(d[i, j]|e_{dom}) = \frac{\text{count}(d[i,j]|e_{dom})}{\text{count}(e_{dom})}$, where count($e_{dom}$) is the number of clean values and count($d[i, j]|e_{dom}$) is the frequency of clean value $d[i, j]$ in the active domain of the data error. This conditional probability shows the chance of observing a clean value $d[i, j]$ in column j.

Overall, we have one domain-based error corrector model per column, resulting in $|A|$ domain-based models.

Example 6 (Domain-based models). Considering the clean values of column *Name* in Example 1, a domain-based model learns that all the clean values have the same probability to be a correction for each data error inside this column because they all appear just once. Thus, the corresponding domain-based model proposes two correction candidates P("Hana"|Name) = 0.5 and P("Gandom"|Name) = 0.5 for any data error e in column *Name*. □

4.2 Learning to Ensemble Models

The trained error corrector models generate various potential corrections for any data error using its value, vicinity, and domain contexts. Thus, we need to identify the actual correction among all the proposed correction candidates from the different models. First, we need to define an

appropriate sampling and labeling strategy to collect user labels. Then, we need to design a feature representation and a classification task in a way that all generated corrections for all data errors can be effectively and efficiently evaluated.

4.2.1 Tuple Sampling and Labeling

Baran incorporates user supervision in the form of a limited number of manual corrections for data error examples. It leverages these examples to update all error corrector models and to train classifiers. To sample tuples for user labeling, Baran follows an iterative procedure. In each iteration, Baran draws a tuple t^* that maximizes the tuple scoring formula

$$t^* = \underset{t \in d}{\operatorname{argmax}} \prod_{d[i,j] \in t \cap E'_j} \exp(\frac{|E'_j|}{|E_j|}) \exp(\frac{\text{count}(d[i,j]|E'_j)}{|E'_j|}), \quad (2)$$

where E_j is the set of all data errors in column j; E'_j is the set of those data errors in column j that have not been fixed yet; and $\text{count}(d[i,j]|E'_j)$ is the number of unfixed data errors in column j whose value is exactly $d[i,j]$. This scoring formula benefits tuples that (1) contain more unfixed data errors, (2) their data errors reside in columns that have a high number of unfixed data errors, and (3) their erroneous values are frequent among the unfixed data errors. This way, Baran obtains informative labeled data points for the classifiers of underlabeled columns. Once the user fixes data errors of the sampled tuple t^*, the value-based, vicinity-based, and domain-based models will be updated accordingly.

Example 7 (**Updating models**)**.** Assume that Baran samples the first tuple in Example 1. The user fixes the data errors in this tuple. Therefore, the value-based models will be updated with the new example of erroneous value "5th Str" that is corrected to the value "5th Street". In particular, a value-based model with the Unicode encoder and the adder operator learns to add "eet" after "Str" for all encounters of erroneous values with a similar pattern to "5th Str". The corresponding vicinity-based model of *Name* $\rightarrow$ *Address* will be updated with a new association between the value "Hana" and the value "5th Street". Furthermore, the domain-based model of column *Address* will be updated as now we have two clean values in this column. □

Choosing the right form of human supervision has been always a challenge in data cleaning. Most data cleaning tasks need to incorporate human supervision as the desired corrections might be use case dependent or subjective. However, if human supervision is erroneous itself, the data cleaning task will be flawed as well. This problem has been identified in the previous constraint-based approaches in the form of obsolete [43] or inaccurate [9] integrity constraints.

In our example-driven system, this problem may appear in the form of wrong user correction examples. However, we argue that Baran is more robust against this issue than constraint-based approaches because of three reasons. First, example-based supervision is less complicated and thus less error-prone than rule generation. For example, it is more intuitive for the user to fix erroneous value "November (16, 1990)" to "16 November 1990" instead of writing a regular expression rule to do so. Note that a rule is always a generalization of many examples and has to undergo several tests before it can be considered as a trustworthy business rule [42]. Second, Baran limits all means of human supervision, such as providing rules, parameters, external sources, and annotations, to just labeling a few tuples. While, in existing approaches, the number of labels scales with the size of dataset (e.g., $1\% - 10\%$ of dataset [20, 41]), in Baran, the number of labels scales with the number of data error types of dataset (i.e., $10 - 20$ tuples). This limited human interaction naturally diminishes the possibility of human mistakes as well. At the same time, the user can always skip examples that are hard to label. Third, when learning through examples, one can compensate human labeling errors by simply considering more user-provided examples. Constraint-based approaches however are not that flexible as wrong input rules are harder to be compensated by other rules inside a dataset.

4.2.2 Feature Generation and Classification

We can now leverage user labels to train classifiers that predict the final correction of a data error. A straightforward approach is to define a multiclass classification task, where each correction candidate resembles a target class and the classifier has to choose one of these target classes for each data error. However, formulating this classification task as a multiclass classification leads to the sparsity issue of the feature vector [10]. In the use case at hand, the feature vector has to encode the probabilities of all the error corrector models for each correction candidate. Formally, the feature vector of data error e would be $v(e) = [P(c|e_m) \mid \forall m \in M, \forall c \in C]$, where M is the set of all models and C is the set of all correction candidates. Thus, the size of the feature vector would scale with the number of correction candidates, while not every correction candidate is relevant for each data error. As a result, there will be a large number of zero elements in the feature vector.

Example 8 (**Multiclass classification**)**.** Assume we have 20 error corrector models, each proposing 100 correction candidates for any data error. In the multiclass classification task, we have to encode all the $20 \times 100 = 2000$ model probabilities into the feature vector of each data error. □

To avoid the sparsity issue of the multiclass classification, we formulate the classification task as a binary decision. The role of the binary classifier is to decide whether a correction candidate is the actual correction of a data error or not. We generate a feature vector that represents the fitness of one particular correction for one particular data error inside a column. Thus, for any combination of data error e and correction candidate c, we collect all the error corrector model probabilities as a feature vector. Formally,

$$v(e,c) = [P(c|e_m) \mid \forall m \in M], \quad (3)$$

where M is the set of all the error corrector models. When a feature vector contains mostly close-to-one probabilities (i.e., $P(c|e_m) \approx 1.0$ for the most of models $m \in M$), it is more likely that the correction candidate c is the actual correction of the data error e; Because, in this case, most error corrector models with high confidence propose this correction candidate for this data error. Hence, the set of feature vectors of data errors inside a particular column j is

$$V_j = \{v(e,c) \mid \forall e \in E_j, \forall c \in C_e\}, \quad (4)$$

where E_j is the set of data errors of column j and C_e is the set of all correction candidates for a data error e. The number of feature vectors depends on the number of correction candidates that the error corrector models propose.

This feature representation has three benefits in contrast to the feature representation of the multiclass classification task. First, this feature vector is small and dense. The number of features is equal to the number of error corrector models and the ratio of non-zero features is higher because the considered models are relevant for the pair at hand. Second, this feature representation leads to fast performance convergence with only a few user labels as we can transform one user label into several training data points. Assume that the user fixes data error e with correction c^*. Baran extends this one user label into multiple training data points. Naturally, we have a positive data point that indicates correction $c^* \in C_e$ is the actual correction of data error e. Furthermore, we have many negative data points that indicate all corrections $c \in C_e \backslash \{c^*\}$ are not the actual correction of data error e. Third, this highly imbalanced training set, with a few positive and a large number of negative data points, makes our classifier conservative in predicting a correction. In fact, our classifier is more biased towards the negative class and prevents false positive corrections. That is why the precision of our system is generally high.

Instead of training one classifier for the whole dataset, we train one binary classifier per column because data errors, their required correction techniques, and the usefulness of their contexts are better comparable inside their domain. Although Baran trains one classifier per column, it preserves all inter-column dependency signals via the vicinity-based error corrector models, which are encoded as features for each pair of a data error and a correction candidate. The vicinity-based models propose corrections for a data error based on the functional dependency of the given column with a different column.

In each iteration, Baran trains all the classifiers and applies each on all data errors of the corresponding column. The classifiers do not overwrite corrections provided by the user. For each pair of data error e and correction candidate c, the corresponding classifier predicts a label with a confidence score. For a data error e, the classifier can determine zero or multiple correction candidates c as the final corrections. If the binary classifier predicts the label 0 for every correction candidate, no correction will be selected for the corresponding data error. If it predicts the label 1 for multiple correction candidates, Baran selects the correction with the highest confidence score as the final correction. This iterative procedure is repeated as long as $|L| < \theta_{\text{Labels}}$, where $|L|$ is the number of labeled tuples. Baran considers the output of the last iteration as the final system output.

Example 9 (Feature generation and classification). Considering column *Address* in Example 1, the following table contains pairs of data errors and correction candidates and their corresponding features. For brevity, we just demonstrate a few pairs and features. The features are a value-based model with the Unicode encoder and the adder operator ($m_{\text{Unicode+Adder}}$), a vicinity-based model ($m_{\text{Name}\rightarrow\text{Address}}$), and a domain-based model (m_{Address}). The first two pairs are labeled as the user already validated tuple 1 of our toy dataset in Example 7.

Error	Correction	$\mathbf{m}_{\text{Unicode+Adder}}$	$\mathbf{m}_{\text{Name}\rightarrow\text{Address}}$	$\mathbf{m}_{\text{Address}}$	Label
5th Str	5th Street	1.0**	1.0*	0.5	1
5th Str	7th Street	0.0	0.0	0.5	0
-	5th Street	0.0	1.0*	0.5	
-	7th Street	0.0	0.0	0.5	
9th Str	5th Street	0.0	0.0	0.5	
9th Str	7th Street	0.0	0.0	0.5	
9th Str	9th Street	1.0**	0.0	0.0	

The classifier of column *Address* receives these pairs as data points. It trains with the first two labeled data points and then predicts the label of the rest of unlabeled data points. In particular, the classifier predicts "5th Street" as the final correction of erroneous value "-" because of the vicinity-based feature $m_{\text{Name}\rightarrow\text{Address}}$ (marked with *). This model returns the probability of 1.0 for the pair ("-", "5th Street") because, after labeling tuple 1 and updating models in Example 7, the vicinity-based model learned the association of value "Hana" in column *Name* and value "5th Street" in column *Address*. Furthermore, the classifier predicts "9th Street" as the final correction of erroneous value "9th Str" because of the value-based feature $m_{\text{Unicode+Adder}}$ (marked with **). This model returns the probability of 1.0 for the pair ("9th Str", "9th Street") because data error "5th Str" matches "9th Str" based on the Unicode encoding and the correction candidate "9th Street" is generated with the same adder operator that generates "5th Street" for "5th Str". □

5. PRETRAINING MODELS

So far, the correction candidates were either provided by the correct values inside the given dataset or through user corrections. This approach might face two general limitations. First, the limited number of user corrections might not be enough to train the error corrector models sufficiently. Second, some out-of-dataset corrections might never be found. Fortunately, our problem formulation allows us to extract additional correction candidates from external sources to pretrain the error corrector models.

As mentioned, we have three groups of error corrector models. Since the vicinity-based and domain-based error corrector models are schema dependent, they should be pretrained on datasets with the same schema. Thus, pretraining them would be straightforward as the historical data would be a structured dataset with the same schema. The value-based error corrector models can be schema independent and hence can be pretrained on any dataset where value corrections can be extracted. In the absence of structured datasets with ground truth, we can resort to non/semi-structured datasets with user-committed corrections. There are publicly available general-purpose revision histories, such as the Wikipedia page revision history. Extracting value-based corrections from these revision histories requires two main steps. First, we need to break down the non/semi-structured revision texts into text segments (i.e., chunks). Second, we need to align text segments across subsequent revisions to collect value-based corrections. Here, we briefly discuss the implementation of these steps for the Wikipedia page revision history as a general-purpose and publicly available revision dataset. To apply the same approach on other similar revision histories, such as web pages', we just need to adapt the text segmenter to the corresponding markup language, e.g., HTML.

The Wikipedia page revision history contains terabytes of human-committed revisions. It is available in Wikipedia released dumps [1]. The semi-structured Wikipedia pages are written in the Wikitext markup language, which recognizes a set of entities [2]. To segment texts, Baran recursively breaks each page revision text down into its entities. Baran then aligns the corresponding segments in every consecutive segment lists by performing diff checking again, but this time on the segment level. Baran discards value-based corrections that involve null values. Finally, Baran pretrains

the value-based error corrector models with these additional correction examples. The pretrained models generate more correction candidates for data errors of the dataset at hand. Therefore, Baran can fix more data errors with the same user labeling budget as more correction candidates are now available and, at the same time, the corresponding features of the pretrained models exhibit more evidence for the classifiers. Note that Baran can still avoid irrelevant correction candidates with the help of the user labels. The user labels let the classifier learn and prioritize across all available correction candidates.

Example 10 (**Pretraining models**). A Wikipedia page $P = \{r_1, r_2\}$ with a history of two revisions could be as follows:

r_1 = "'''Chris Nolan''' (born on ''30/07/1970'') is a well-known [[British]] film-maker."

r_2 = "'''Christopher Nolan''' (born on ''30.07.1970'') is a well-known [[English]] filmmaker."

Baran breaks down these page revision texts into two lists of text segments:

S_1 = ["Chris Nolan", "(born on ", "30/07/1970", ") is a well-known ", "British", " film-maker."]

S_2 = ["Christopher Nolan", "(born on ", "30.07.1970", ") is a well-known ", "English", " filmmaker."]

Baran then aligns the corresponding segments of the two consecutive segment lists:

$$\text{diff}(S_1, S_2) = \begin{cases} \text{Replace ``Chris Nolan'' with ``Christopher Nolan''.} \\ \text{Replace ``30/07/1970'' with ``30.07.1970''.} \\ \text{Replace ``British'' with ``English''.} \\ \text{Replace `` film-maker.'' with `` filmmaker.''.} \end{cases}$$

Thus, we obtain more value-based corrections as training data, such as fixing "Chris Nolan" to "Christopher Nolan". We also tokenize each segment and increase the training data with finer granular value-based correction examples, such as fixing "Chris" to "Christopher".

Pretraining the value-based models with these new correction examples makes it possible to fix the last remaining data error of our toy dataset from Example 1, without any further user label. Considering column *Name* in Example 1, the following table contains pairs of data errors and correction candidates and their corresponding features. For brevity, we just demonstrate a few pairs and features. The features are a value-based model with the identity encoder and the adder operator ($m_{\text{Identity+Adder}}$) and a domain-based model (m_{Name}). The first two pairs are labeled as the user already validated tuple 1 of our toy dataset in Example 7.

Error	Correction	$m_{\text{Identity+Adder}}$	m_{Name}	Label
H	Hana	1.0*	0.67	1
H	Gandom	0.0	0.33	0
Chris	Hana	0.0	0.67	
Chris	Gandom	0.0	0.33	

The classifier of column *Name* receives these pairs as data points. It trains with the first two labeled data points and then predicts the label of the rest of unlabeled data points. With only these two labeled data points, the classifier has no chance to fix the erroneous value "Chris" to its actual correction "Christopher", because it is not among the correction candidates at all. However, if we pretrain the value-based model $m_{\text{Identity+Adder}}$ with the previously extracted example of fixing "Chris" to "Christopher" from Wikipedia, we will have the following new pair of a data error and a correction candidate as a new data point.

Error	Correction	$m_{\text{Identity+Adder}}$	m_{Name}	Label
Chris	Christopher	1.0*	0.0	

The classifier now has enough evidence to fix the erroneous value "Chris" without any further user label. Since the user already fixed the erroneous value "H" to "Hana", the classifier learned that the value-based model $m_{\text{Identity+Adder}}$ (marked with *) is an important feature. The classifier also observes that the same value-based feature $m_{\text{Identity+Adder}}$ has a high probability for the pair ("Chris", "Christopher"), as the model learned this probability in the pretraining phase. Therefore, the classifier can predict "Christopher" as the final correction of the erroneous value "Chris". □

6. EXPERIMENTS

Our experiments aim to answer the following questions. (1) How does Baran, with and without transfer learning, compare to the existing error correction systems? (2) How do the error corrector models affect the system performance? (3) What is the impact of our tuple sampling approach? (4) How does the choice of the classifier affect the system performance? (5) How does the result quality of the upstream error detection engine affect the correction performance? We first introduce our experimental setting and then detail our experiments.

6.1 Setup

Datasets. We evaluate our system on 7 well-known datasets from existing literature as described in Table 2. The difficulty level of the error correction task depends on the error rate, the diversity of error types, and the availability of error context signals. We manually examined the datasets to identify prevalent data error types and useful contextual information for correcting these data errors.

Hospital [37] and *Flights* [25, 37] have rich contextual information, including a high degree of data redundancy in the form of duplicate tuples and correlated columns. At the same time, they are challenging datasets for different reasons. Since the data errors of *Hospital* are scarce and randomly imposed, sampling informative tuples is particularly challenging on this dataset. On the other hand, since *Flights* has a high error rate, the degree of trustworthy contextual information is lower. *Address* is a proprietary and *Tax* is a synthetic dataset from the BART repository [7]. Both datasets are large, contain various data error types, but contain fewer duplicate tuples and correlated columns. Their large size leads to a huge search space for finding actual corrections. *Beers* [21], *Rayyan* [30], and *IT* [3] are also real-world datasets that were cleaned by the dataset owners. These three datasets lack data redundancy, which makes it challenging to fix data errors.

Baselines. We compare Baran to 4 recent baseline systems.

- *KATARA* [12] is a data cleaning system powered by knowledge bases that takes a set of entity relationships as input and fixes the violating data errors accordingly. We ran KATARA with all the entity relationships that are available in the DBpedia knowledge base [8].
- *SCARE* [48] is an error correction system that partitions the dataset and uses the clean values to choose corrections of data errors based on their statistical likelihood. We ran SCARE with random data partitioning. We set its maximum number of value corrections to the number of detected errors ($\delta = |E|$) to accommodate all data errors.

Table 2: Dataset characteristics. The error types are missing value (MV), typo (T), formatting issue (FI), and violated attribute dependency (VAD) [36].

Name	Size	Error Rate	Error Types	Data Constraints
Hospital	1000 × 20	3%	T, VAD	city → zip, city → county, zip → city, zip → state, zip → county, county → state, index (digits), provider number (digits), zip (5 digits), state (2 letters), phone (digits)
Flights	2376 × 7	30%	MV, FI, VAD	flight → actual departure time, flight → actual arrival time, flight → scheduled departure time, flight → scheduled arrival time
Address	94306 × 12	14%	MV, FI, VAD	address → state, address → zip, zip → state, state (2 letters), zip (digits), ssn (digits)
Beers	2410 × 11	16%	MV, FI, VAD	brewery id → brewery name, brewery id → city, brewery id → state, brewery id (digits), state (2 letters)
Rayyan	1000 × 11	9%	MV, T, FI, VAD	journal abbreviation → journal title, journal abbreviation → journal issn, journal issn → journal title, authors list (not null), article pagination (not null), journal abbreviation (not null), article title (not null), article language (not null), journal title (not null), journal issn (not null), article journal issue (not null), article journal volume (not null), journal created at (date)
IT	2262 × 61	20%	MV, FI	support level (not null), app status (not null), curr status (not null), tower (not null), end users (not null), account manager (not null), decomm dt (not null), decomm start (not null), decomm end (not null), end users (not 0), retirement (predefined list), emp dta (predefined list), retire plan (predefined list), division (predefined list), bus import (predefined list)
Tax	200000 × 15	4%	T, FI, VAD	zip → city, zip → state, first name → gender, area code → state, gender (predefined list), area code (3 digits), phone (7 formatted digits), state (2 letters), zip (non-zero-leading digits), material status (predefined list), has child (predefined list), salary (digits)

- *Holistic* [11] is a data cleaning system that uses denial constraints. We ran Holistic with all the data constraints (i.e., integrity rules and column patterns) provided by datasets owners (Table 2).
- *HoloClean* [37] is an error correction system that leverages integrity rules, matching dependencies, and statistical signals to fix data errors holistically. We ran HoloClean with all the data constraints and matching dependencies that are provided by datasets owners (Table 2).

Evaluation measures. We report precision, recall, and the F_1 score to evaluate the effectiveness. Precision is the number of correctly fixed data errors divided by the number of all fixed data errors. Recall is the number of correctly fixed data errors divided by the number of all data errors. The F_1 score is the harmonic mean of precision and recall. We also report the runtime in seconds. We report the number of labeled tuples to evaluate the human involvement. For each metric, we report the mean of 10 independent runs.

Baran default setting. By default, we run Baran with all described error corrector models, without pretraining them. We use AdaBoost [16] as the classifier and set the labeling budget to $\theta_{\text{Labels}} = 20$. We run the experiments on an Ubuntu 16.04 LTS machine with 28 2.60 GHz cores and 264 GB memory. Baran is integrated with our error detection system Raha and is available online[1].

6.2 Comparison with the Baselines

We compare the performance of Baran with the baselines in terms of effectiveness, efficiency, and human involvement. All the error correction systems in this section take as the input the same correct and complete set of data errors.

Effectiveness. Table 3 shows the effectiveness of all systems in error correction. Baran outperforms all the baselines in terms of the F_1 score on all the datasets as our approach in formulating error correction achieves both high precision and recall. In fact, since Baran trains a comprehensive set of error corrector models based on different contexts of data errors, these models propose a large set of potential corrections that increases the achievable recall bound significantly. Later, when Baran leverages a few user labels to ensemble these potential corrections into the final corrections, the high error correction precision is also maintained.

The effectiveness of Baran depends on the amount of value-based, vicinity-based, and domain-based context information that each dataset provides. On information-rich datasets, such as *Hospital* and *Flights*, with many duplicate rows and correlated columns, Baran can train and ensemble effective error corrector models leveraging all the three data error contexts. Therefore, Baran achieves high F_1 scores on these datasets. In particular, Baran achieves perfect performance on the *Flights* dataset, which has a high degree of redundancy. On the other hand, on datasets with less error context information, such as *Address*, where the erroneous values are often missing values, Baran cannot fix all the data errors accurately.

Pretraining value-based models on the revision history of more than 300000 Wikipedia pages (row *Baran (with TL)* in Table 3) improves the F_1 score more significantly on datasets with prevalent syntactic data issues, such as *Hospital* and *Rayyan*. The pretraining provides more evidence for the classifiers to predict the actual correction. In particular, the pretrained value-based models generate additional correction candidates that enable Baran to converge to its reported F_1 score, with fewer than 20 labeled tuples for the same F_1 score. On the *Hospital* dataset, pertaining generates 455390 new correction candidates. As a result, we just need 13 labeled tuples to achieve the same F_1 score achieved with 20 labeled tuples. On the *Rayyan* dataset, pertaining generates 77569 new correction candidates. With pretraining, we reach the same F_1 score that previously needed 20 labeled tuples with only 18 labeled tuples. On the *Tax* dataset, pretraining generates 7134254 new correction candidates. As a result, we just need 19 labeled tuples to achieve the same F_1 score achieved with 20 labeled tuples. Contrary, the effectiveness improvement is minor on datasets like *IT*, where most of the erroneous values are missing values. Here, the value-based models cannot propose potential corrections effectively, regardless of being pretrained or not.

KATARA has poor precision because the ambiguity of concepts leads to a mismatch between the dataset and the knowledge base. This system also has poor recall because most parts of datasets cannot be matched to knowledge bases. Holistic has poor precision and recall because the provided integrity rules can only fix a portion of data errors. Although HoloClean has relatively high precision and recall on datasets with a high degree of redundancy, such as *Hospital* and *Flights*, it cannot achieve the same effectiveness on the rest of datasets. On datasets with lower degrees of redundancy or fewer predefined data constraints, HoloClean cannot find the correction of data errors accurately; Because the actual correction either does not exist anywhere in data or is not covered by data constraints. SCARE suffers from the same drawbacks as HoloClean.

As long as a dataset provides rich contextual information, the error rate does not affect the effectiveness of Baran sig-

[1] `https://github.com/BigDaMa/raha`

Table 3: System effectiveness in comparison to the baselines.

System	Hospital			Flights			Address			Beers			Rayyan			IT			Tax		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
KATARA	0.98	0.24	0.39	0.00	0.00	0.00	0.79	0.01	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.01	0.02	0.59	0.01	0.02
SCARE	0.67	0.53	0.59	0.57	0.06	0.11	0.10	0.10	0.10	0.16	0.07	0.10	0.00	0.00	0.00	0.20	0.10	0.13	0.01	0.01	0.01
Holistic	0.52	0.38	0.44	0.21	0.01	0.02	0.41	0.31	0.35	0.49	0.01	0.02	0.85	0.07	0.13	1.00	0.78	0.88	0.96	0.26	0.41
HoloClean	1.00	0.71	0.83	0.89	0.67	0.76	0.01	0.01	0.01	0.01	0.01	0.01	0.00	0.00	0.00	0.01	0.01	0.01	0.11	0.11	0.11
Baran	0.88	0.86	0.87	1.00	1.00	**1.00**	0.67	0.32	**0.43**	0.91	0.89	**0.90**	0.76	0.40	0.52	0.98	0.98	**0.98**	0.84	0.78	0.81
Baran (with TL)	0.94	0.88	**0.91**	1.00	1.00	1.00	0.67	0.32	0.43	0.94	0.87	0.90	0.80	0.44	**0.57**	0.98	0.98	0.98	0.95	0.73	**0.83**

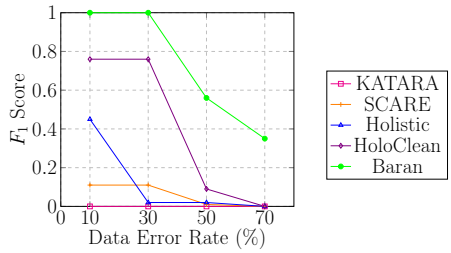


Figure 2: Scaling the error rate on *Flights*.

nificantly. For example, Baran achieves high effectiveness on both context-rich datasets *Hospital* and *Flights* although the former has a low (3%) and latter has a high (30%) data error rate. To further analyze the effect of the data error rate, we select our most erroneous dataset *Flights* and generate four without-replacement samples of it with 10%, 30%, 50%, and 70% erroneous data cells. Figure 2 shows the F_1 score of the error correction systems on these four datasets. As the error rate increases, the F_1 score of all the systems naturally drops because trustworthy evidence diminishes. However, Baran consistently outperforms all the other systems because it leverages the remaining scarce trustworthy contexts of data errors more effectively.

Human involvement. Figure 3 shows the effectiveness of systems in error correction with respect to the number of labeled tuples. The F_1 score of Baran quickly converges with only a few labeled tuples and outperforms the F_1 score of all the other systems. This fast convergence speed is due to the Baran's tuple sampling and feature generation approach that generates a large number of informative training data points with a few user labels. Since KATARA, SCARE, Holistic, and HoloClean do not leverage user labels, their F_1 score is independent of the number of labeled tuples. However, we argue that they leverage human supervision in other more tedious forms. KATARA needs the user to provide related knowledge bases. SCARE needs the user to provide statistical parameters. Holistic and HoloClean need the user to provide the correct and complete set of integrity rules and matching dependencies.

Efficiency. Table 4 shows the runtime of the systems in seconds. Although efficiency is not the main concern of Baran, it displays a competitive runtime in comparison to the other baselines. The reported runtime captures the online phase of Baran as the offline phase is totally independent of the input dataset. Optimizing machine runtime efficiency has not been the main goal of data cleaning systems as optimizing effectiveness and human involvement are more important objectives [3, 37]. However, it is important to develop systems that can work in a reasonable runtime.

6.3 Error Corrector Models Impact Analysis

We conduct an ablation experiment on the error corrector models to better understand their effect on the overall

Table 4: System runtime (in seconds).

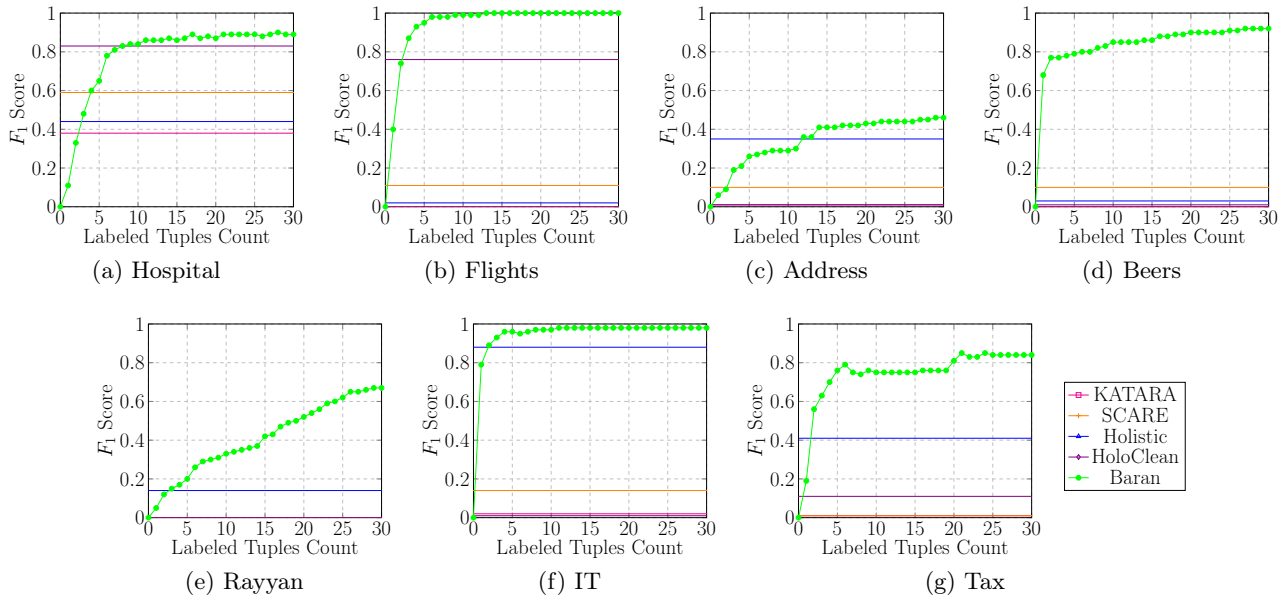
System	Hospital	Flights	Address	Beers	Rayyan	IT	Tax
KATARA	234	116	5739	180	134	2031	15992
SCARE	76	123	11853	363	216	8717	55495
Holistic	15	10	69	9	8	3	247
HoloClean	148	39	17582	96	112	885	25778
Baran	23	22	11073	114	26	247	11936

effectiveness of Baran. First, we run Baran with all the default error corrector models (row *All* in Table 5). Then, we exclude each type of models, one at a time, to analyze its impact. For example, *All - VaM* means that Baran leverages all the models but the value-based ones. Finally, we also evaluate the performance of Baran with all the default models together with custom dataset-specific models (row *All + CM*) obtained from the data constraints in Table 2. Baran leverages these data constraints as hard-coded correctors that overwrite our default models if necessary.

As shown in Table 5, Baran has the highest F_1 score with all the default error corrector models on most of the datasets. By collecting the proposed potential corrections from all the error corrector models, Baran has more context information to fix the data errors. However, on some datasets, we observe that excluding one type of error corrector models can lead to a higher F_1 score. Excluding vicinity-based or domain-based models improves the F_1 score on the *Tax* dataset. The reason is that, on this large dataset with thousands of rows, these models propose thousands of clean values from the active domain of the data error as potential corrections. Learning to find the actual correction among this huge search space needs more learning iterations and user labels. Excluding value-based models improves the F_1 score on the *Hospital* dataset. Since this dataset has randomly imposed typos, the value-based models cannot effectively learn value-based corrections from this randomness. That is why the F_1 score is higher on the *Hospital* dataset when the value-based models are excluded. Excluding the vicinity-based error corrector models significantly drops the F_1 score on datasets with high inter-column dependencies, such as *Hospital* and *Flights*. This decline shows that Baran effectively fixes inter-column dependency violations with including vicinity-based models. Adding custom dataset-specific models to the default set of error corrector models does not affect F_1 score on most of the datasets as our default models are general enough and already cover these prevalent data constraints. For example, one-attribute to one-attribute functional dependencies are already incorporated into our system due to the vicinity-based models.

6.4 Tuple Sampling Impact Analysis

We analyze the impact of our tuple sampling approach on the effectiveness of Baran by comparing two versions of our system with two different sampling approaches. the *Uniform sampling* approach selects erroneous tuples for user labeling according to a uniform probability distribution. *Our tuple*



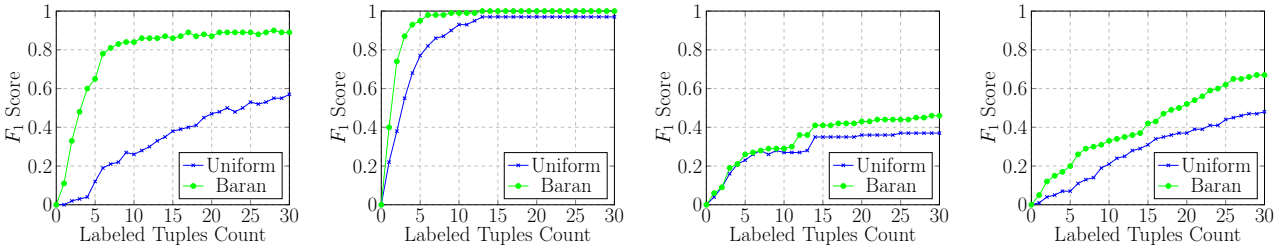
(a) Hospital (b) Flights (c) Address (d) Beers

(e) Rayyan (f) IT (g) Tax

Figure 3: System effectiveness with respect to the number of labeled tuples.

Table 5: System effectiveness with different error corrector models: value-based models (VaM), vicinity-based models (ViM), domain-based models (DoM), all default models (All), and custom models (CM).

Error Corrector Models	Hospital			Flights			Address			Beers			Rayyan			IT			Tax		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
All - VaM	0.95	0.88	**0.91**	1.00	1.00	1.00	0.61	0.06	0.11	0.67	0.42	0.52	0.19	0.07	0.10	0.98	0.95	0.96	0.44	0.24	0.31
All - ViM	0.64	0.31	0.42	0.08	0.06	0.07	0.40	0.25	0.31	0.87	0.86	0.86	0.48	0.34	0.40	0.98	0.98	0.98	0.88	0.88	**0.88**
All - DoM	0.88	0.87	0.87	1.00	1.00	1.00	0.56	0.25	0.35	0.91	0.88	0.89	0.54	0.35	0.42	0.98	0.98	0.98	0.90	0.81	0.85
All	0.88	0.86	0.87	1.00	1.00	**1.00**	0.67	0.32	**0.43**	0.91	0.89	**0.90**	0.76	0.40	**0.52**	0.98	0.98	**0.98**	0.84	0.78	0.81
All + CM	0.90	0.89	0.89	1.00	1.00	1.00	0.67	0.32	0.43	0.91	0.89	0.90	0.76	0.40	0.52	0.98	0.98	0.98	0.84	0.78	0.81



(a) Hospital (b) Flights (c) Address (d) Rayyan

Figure 4: System effectiveness with different tuple sampling approaches.

sampling approach selects tuples according to their informativeness for the classifiers.

As shown in Figure 4, our tuple sampling approach speeds up the convergence of the system. This higher convergence speed is more significant on datasets with more randomly dispersed data errors, such as *Hospital* and *Rayyan*. On these datasets, the classifiers need to have enough labeled data errors from each column and data error type to be able to fix all the data errors accurately. That is why our tuple sampling approach that oversamples the underlabeled columns converges faster. Due to the space limitation, we just report the results on 4 datasets. Baran's sampling improvement is only minor on the remaining datasets.

6.5 Classifier Impact Analysis

We analyze the impact of the classifier on the effectiveness of Baran. We tested *AdaBoost*, *Decision Tree*, *Gradient Boosting*, and *Stochastic Gradient Descent*, all implemented in *scikit-learn* Python module [33]. We applied grid search to find the best hyperparameters for each classifier.

Table 6 shows that the choice of the classifier does not have a significant impact on the effectiveness of the system. Although on some datasets, such as *Address*, the F_1 score varies more, there are always multiple classifiers that achieve almost the same F_1 score. In our current prototype, we deploy AdaBoost because it is an advanced ensemble classifier [16], which is less susceptible to overfitting [38].

6.6 Error Detection Impact Analysis

Although error detection and error correction have been considered as two orthogonal tasks in literature [3, 20, 37, 48], it is important to analyze the influence of imperfect error detection on the downstream error correction effectiveness. Naturally, the effectiveness of error correction depends

Table 6: System effectiveness with different classifiers.

Classifier	Hospital			Flights			Address			Beers			Rayyan			IT			Tax		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
AdaBoost	0.88	0.86	0.87	1.00	1.00	1.00	0.67	0.32	0.43	0.91	0.89	0.90	0.76	0.40	0.52	0.98	0.98	0.98	0.84	0.78	0.81
Decision Tree	0.88	0.85	0.86	1.00	1.00	1.00	0.70	0.34	0.46	0.91	0.89	0.90	0.62	0.34	0.44	0.98	0.98	0.98	0.74	0.73	0.73
Gradient Boosting	0.94	0.68	0.79	1.00	1.00	1.00	0.63	0.12	0.20	0.92	0.81	0.86	0.66	0.41	0.51	0.99	0.98	0.98	0.97	0.59	0.73
Stochastic Gradient Descent	0.95	0.92	0.93	1.00	1.00	1.00	0.66	0.25	0.36	0.95	0.87	0.91	0.59	0.21	0.31	0.99	0.98	0.98	0.83	0.63	0.72

Table 7: System effectiveness with imperfect and perfect error detection (ED) and error correction (EC).

System	Hospital			Flights			Address			Beers			Rayyan			IT			Tax		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
Perfect ED + Baran	0.88	0.86	0.87	1.00	1.00	1.00	0.67	0.32	0.43	0.91	0.89	0.90	0.76	0.40	0.52	0.98	0.98	0.98	0.84	0.78	0.81
Raha + Perfect EC	0.98	0.58	0.73	0.97	0.75	0.85	0.83	0.85	0.84	0.98	1.00	0.99	0.83	0.79	0.81	0.99	0.98	0.98	0.97	0.98	0.97
Raha + HoloClean	0.19	0.41	0.26	0.08	0.16	0.11	0.01	0.01	0.01	0.01	0.01	0.01	0.00	0.00	0.00	0.01	0.01	0.01	0.11	0.11	0.11
Raha + Baran	0.89	0.52	0.66	0.88	0.53	0.66	0.57	0.32	0.41	0.93	0.87	0.90	0.50	0.27	0.35	0.98	0.96	0.97	0.84	0.66	0.74
Raha + Baran (In)	0.95	0.52	0.67	0.84	0.56	0.67	0.53	0.32	0.40	0.93	0.87	0.90	0.44	0.21	0.28	0.99	0.97	0.98	0.84	0.77	0.80

on the effectiveness of error detection. Typically, the error detection recall is the upper bound of the error correction recall [37]. We leverage Raha [27], a state-of-the-art error detection system, to test the performance of end-to-end data cleaning pipelines. Raha is also a configuration-free system and needs only a few labeled tuples of each dataset to detect data errors. We compare the effectiveness of three end-to-end data cleaning scenarios. The first scenario is as before, Baran takes the perfectly detected data errors as the input and fixes them (row *Perfect ED + Baran* in Table 7). In the second scenario, Raha detects data errors of the dataset and then the user fixes all the detected data errors perfectly (row *Raha + Perfect EC*). The effectiveness of this virtual approach is the upper bound of error correction systems. Finally, in the last scenario, Raha detects data errors and then an error correction system, such as Baran or HoloClean, fixes the detected data errors. In particular, we study the effectiveness of two versions of end-to-end data cleaning pipelines with Raha and Baran. In the first pipeline, Raha and Baran work orthogonal and each of them separately asks the user to label 20 tuples (row *Raha + Baran*). In the second integrated pipeline, only Raha asks the user to label 20 tuples and then it passes these labels along with the detected data errors to Baran (row *Raha + Baran (In)*). We also report the effectiveness of HoloClean when it takes the same set of detected data errors (row *Raha + HoloClean*).

As shown in Table 7, imperfect error detection naturally leads to a slight drop of Baran's error correction effectiveness. This decline is minor on most of the datasets, such as *Beers* and *IT*. Both pipelines with Raha and Baran achieve almost the same F_1 score and both clearly outperform the pipeline with HoloClean. Interestingly, the second pipeline achieves higher effectiveness on large datasets, such as *Tax*. This is promising as it shows Raha's clustering-based sampling [27] is effective enough to sample informative tuples for both error detection and correction tasks and we do not need separate user labels for Baran.

7. RELATED WORK

We review related research in error correction and transfer learning as they were the main focus of this work. Furthermore, we discuss data transformation, programming by example, spell checking, and error detection as these areas are partially touched by our system.

Error correction. Existing error correction systems leverage various signals, such as integrity rules [11, 13, 17, 18], external sources [12], active learning [24, 49], cleaned samples [45], statistical likelihoods [48], and a combination of rules and statistics [37]. These approaches show promises in settings where data redundancy and user-provided rules and parameters are available. Baran offers a new task formulation that does not need these prerequisites.

Transfer learning. Transfer learning has been used for entity matching [50], missing value imputation [44], and performance estimation [26]. Baran is the first system that leverages transfer learning for the error correction task.

Data transformation and programming by example. Data transformation is the task of transforming data values from one format into another [6, 23, 5]. Programming by example is the task of synthesizing a program that satisfies a set of input-output examples [19, 39, 40]. Baran leverages data transformation and programming by example as one type of error correction, i.e., value-based correction.

Spell checking. Spell checking is the task of identifying and fixing typos (i.e., misspellings) in texts [34, 28]. Baran not only fixes spell and other linguistic mistakes via value-based models, but also fixes other types of data errors, such as missing values and formatting issues [36].

Error detection. Error detection is the task of detecting data values that are wrong [3]. Previous approaches leverage various techniques to detect data errors, such as data augmentation [20], web tables [46], metadata [41], active learning [29], and combining error detection algorithms [27]. The task of Baran is orthogonal to the error detection task, as the output of any error detection approaches can be fed as the input into Baran.

8. CONCLUSION

We proposed a new error correction system that fixes data errors with respect to their value, vicinity, and domain contexts. Baran trains multiple error corrector models based on these different contexts and then combines them into a final correction for each data error. Furthermore, Baran provides the option of transfer learning. As our experiments show, Baran significantly outperforms existing error correction systems. Despite Baran's promises, there are still future directions for improvement. In particular, designing an effective data cleaning dashboard is an important direction that could support the user to avoid correction mistakes.

Acknowledgements. This project has been supported by the German Research Foundation (DFG) under grant agreement 387872445.

9. REFERENCES

[1] Wikipedia:database download. https://en.wikipedia.org/wiki/Wikipedia:Database_download, 2019. Accessed: 12.09.2019.

[2] Wikitext. https://www.mediawiki.org/wiki/Wikitext, 2019. Accessed: 12.09.2019.

[3] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *PVLDB*, 9(12):993–1004, 2016.

[4] Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: A survey. *VLDBJ*, 24(4):557–581, 2015.

[5] Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker. Dataxformer: A robust transformation discovery system. In *ICDE*, pages 1134–1145, 2016.

[6] A. Arasu, S. Chaudhuri, and R. Kaushik. Learning string transformations from examples. *PVLDB*, 2(1):514–525, 2009.

[7] P. C. Arocena, B. Glavic, G. Mecca, R. J. Miller, P. Papotti, and D. Santoro. Messing up with bart: Error generation for evaluating data-cleaning algorithms. *PVLDB*, 9(2):36–47, 2015.

[8] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC*, pages 722–735, 2007.

[9] G. Beskales, I. F. Ilyas, L. Golab, and A. Galiullin. On the relative trust between inconsistent data and inaccurate constraints. In *ICDE*, pages 541–552, 2013.

[10] M. Blondel, K. Seki, and K. Uehara. Block coordinate descent algorithms for large-scale sparse multiclass classification. *Machine learning*, 93(1):31–52, 2013.

[11] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469, 2013.

[12] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*, pages 1247–1261, 2015.

[13] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. Nadeef: A commodity data cleaning system. In *SIGMOD*, pages 541–552, 2013.

[14] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, and N. Tang. The data civilizer system. In *CIDR*, 2017.

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *arXiv preprint arXiv:1810.04805*, 2018.

[16] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *JCSS*, 55(1):119–139, 1997.

[17] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. The llunatic data-cleaning framework. *PVLDB*, 6(9):625–636, 2013.

[18] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. Cleaning data with llunatic. *VLDBJ*, pages 1–26, 2019.

[19] S. Gulwani. Programming by examples: Applications, algorithms, and ambiguity resolution. In *IJCAR*, pages 9–14, 2016.

[20] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas. Holodetect: Few-shot learning for error detection. In *SIGMOD*, pages 829–846, 2019.

[21] J.-N. Hould. Craft beers dataset. https://www.kaggle.com/nickhould/craft-cans, 2017. Version 1.

[22] J. W. Hunt and M. D. MacIlroy. *An algorithm for differential file comparison*. Bell Laboratories Murray Hill, 1976.

[23] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *SIGCHI*, pages 3363–3372, 2011.

[24] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *PVLDB*, 9(12):948–959, 2016.

[25] X. Li, X. L. Dong, K. Lyons, W. Meng, and D. Srivastava. Truth finding on the deep web: Is the problem solved? *arXiv preprint arXiv:1503.00303*, 2015.

[26] M. Mahdavi and Z. Abedjan. Reds: Estimating the performance of error detection strategies based on dirtiness profiles. In *SSDBM*, pages 193–196, 2019.

[27] M. Mahdavi, Z. Abedjan, R. Castro Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Raha: A configuration-free error detection system. In *SIGMOD*, pages 865–882, 2019.

[28] A. Max and G. Wisniewski. Mining naturally-occurring corrections and paraphrases from wikipedia's revision history. In *LREC*, 2010.

[29] F. Neutatz, M. Mahdavi, and Z. Abedjan. Ed2: A case for active learning in error detection. In *CIKM*, pages 2249–2252, 2019.

[30] M. Ouzzani, H. Hammady, Z. Fedorowicz, and A. Elmagarmid. Rayyan—a web and mobile app for systematic reviews. *Systematic reviews*, 5(1):210, 2016.

[31] S. J. Pan and Q. Yang. A survey on transfer learning. *TKDE*, 22(10):1345–1359, 2009.

[32] T. Papenbrock and F. Naumann. A hybrid approach to functional dependency discovery. In *SIGMOD*, pages 821–833, 2016.

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *JMLR*, 12(Oct):2825–2830, 2011.

[34] T. A. Pirinen and K. Lindén. State-of-the-art in weighted finite-state spell-checking. In *CICLing*, pages 519–532, 2014.

[35] N. Prokoshyna, J. Szlichta, F. Chiang, R. J. Miller, and D. Srivastava. Combining quantitative and logical data cleaning. *PVLDB*, 9(4):300–311, 2015.

[36] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *DE*, 23(4):3–13, 2000.

[37] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB*, 10(11):1190–1201, 2017.

[38] R. E. Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. 2013.

[39] R. Singh. Blinkfill: Semi-supervised programming by example for syntactic string transformations. *PVLDB*, 9(10):816–827, 2016.

[40] R. Singh and S. Gulwani. Transforming spreadsheet data types using examples. In *POPL*, pages 343–356,

2016.

[41] L. Visengeriyeva and Z. Abedjan. Metadata-driven error detection. In *SSDBM*, pages 1–12, 2018.

[42] L. Visengeriyeva and Z. Abedjan. Anatomy of metadata for data curation. *JDIQ*, 12(3), 2020.

[43] M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller. Continuous data cleaning. In *ICDE*, pages 244–255, 2014.

[44] G. Wang, J. Lu, K.-S. Choi, and G. Zhang. A transfer-based additive ls-svm classifier for handling missing data. *IEEE transactions on cybernetics*, 50(2):739–752, 2018.

[45] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD*, pages 469–480, 2014.

[46] P. Wang and Y. He. Uni-detect: A unified approach to automated error detection in tables. In *SIGMOD*, pages 811–828, 2019.

[47] K. Whistler and L. Iancu. Unicode character database. `http://www.unicode.org/reports/tr44/`, 2019. Accessed: 12.09.2019.

[48] M. Yakout, L. Berti-Équille, and A. K. Elmagarmid. Don't be scared: Use scalable automatic repairing with maximal likelihood and bounded changes. In *SIGMOD*, pages 553–564, 2013.

[49] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, 4(5):279–289, 2011.

[50] C. Zhao and Y. He. Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning. In *WWW*, pages 2413–2424, 2019.