HoloDetect: Few-Shot Learning for Error Detection

Alireza Heidari*, Joshua McGrath†, Ihab F. Ilyas*, Theodoros Rekatsinas†
a5heidar@uwaterloo.ca,mcgrath@cs.wisc.edu,ilyas@uwaterloo.ca,thodrek@cs.wisc.edu
*University of Waterloo and †University of Wisconsin - Madison

ABSTRACT

We introduce a few-shot learning framework for error detection. We show that data augmentation (a form of weak supervision) is key to training high-quality, ML-based error detection models that require minimal human involvement. Our framework consists of two parts: (1) an expressive model to learn rich representations that capture the inherent syntactic and semantic heterogeneity of errors; and (2) a data augmentation model that, given a small seed of clean records, uses dataset-specific transformations to automatically generate additional training data. Our key insight is to learn data augmentation policies from the noisy input dataset in a weakly supervised manner. We show that our framework detects errors with an average precision of ~94% and an average recall of ~93% across a diverse array of datasets that exhibit different types and amounts of errors. We compare our approach to a comprehensive collection of error detection methods, ranging from traditional rule-based methods to ensemble-based and active learning approaches. We show that data augmentation yields an average improvement of 20 F_1 points while it requires access to 3× fewer labeled examples compared to other ML approaches.

KEYWORDS

Error Detection; Machine Learning; Few-shot Learning; Data Augmentation; Weak Supervision

ACM Reference Format:
Heidari, McGrath, Ilyas, and Rekatsinas. 2019. HoloDetect: Few-Shot Learning for Error Detection. In *2019 International Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3299869.3319888

SIGMOD '19, June 30-July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-5643-5/19/06…$15.00
https://doi.org/10.1145/3299869.3319888

1 INTRODUCTION

Error detection is a natural first step in every data analysis pipeline [31, 44]. Data inconsistencies due to incorrect or missing data values can have a severe negative impact on the quality of downstream analytical results. However, identifying errors in a noisy dataset can be a challenging problem. Errors are often heterogeneous and exist due to a diverse set of reasons (e.g., typos, integration of stale data values, or misalignment), and in many cases can be rare. This makes manual error detection prohibitively time consuming.

Several error detection methods have been proposed in the literature to automate error detection [17, 20, 31, 48]. Most of the prior works leverage the side effects of data errors to solve error detection. For instance, many of the proposed methods rely on violations of integrity constraints [31] or value-patterns [33] or duplicate detection [18, 43] and outlier detection [16, 47, 61] methods to identify erroneous records. While effective in many cases, these methods are tailored to specific types of side effects of erroneous data. As a result, their recall for identifying errors is limited to errors corresponding to specific side effects (e.g., constraint violations, duplicates, or attribute/tuple distributional shifts) [2].

One approach to address the heterogeneity of errors and their side effects is to combine different detection methods in an ensemble [2]. For example, given access to different error detection methods, one can apply them sequentially or can use voting-based ensembles to combine the outputs of different methods. Despite the simplicity of ensemble methods, their performance can be sensitive to how different error detectors are combined [2]. This can be either with respect to the order in which different methods are used or the confidence-level associated with each method. Unfortunately, appropriate tools for tuning such ensembles are limited, and the burden of tuning these tools is on the end-user.

A different way to address heterogeneity is to cast error detection as a machine learning (ML) problem, i.e., a binary classification problem: given a dataset, classify its entries as erroneous or correct. One can then train an ML model to discriminate between erroneous and correct data. Beyond automation, a suitably expressive ML model should be able to capture the inherent heterogeneity of errors and their side effects and will not be limited to low recall. However,

the end-user is now burdened with the collection of enough labeled examples to train such an expressive ML model.

1.1 Approach and Technical Challenges

We propose a few-shot learning framework for error detection based on weak supervision [52, 54], which exploits noisier or higher-level signals to supervise ML systems. We start from this premise and show that *data augmentation* [45, 62], a form of weak supervision, enables us to train high-quality ML-based error detection models with minimal human involvement.

Our approach exhibits significant improvements over a comprehensive collection of error detection methods: we show that our approach is able to detect errors with an average precision of ~94% and an average recall of ~93%, obtaining an average improvement of 20 F_1 points against competing error detection methods. At the same time, our weakly supervised methods require access to 3× fewer labeled examples compared to other ML approaches. Our ML-approach also needs to address multiple technical challenges:

- **[Model]** The heterogeneity of errors and their side effects makes it challenging to identify the appropriate statistical and integrity properties of the data that should be captured by a model in order to discriminate between erroneous and correct cells. These properties correspond to attribute-level, tuple-level, and dataset-level features that describe the distribution governing a dataset. Hence, we need an appropriately expressive model for error detection that captures all these properties (features) to maximize recall.
- **[Imbalance]** Often, errors in a dataset are limited. ML algorithms tend to produce unsatisfactory classifiers when faced with imbalanced datasets. The features of the minority class are treated as noise and are often ignored. Thus, there is a high probability of misclassification of the minority class as compared to the majority class. To deal with imbalance, one needs to develop strategies to balance classes in the training data. Standard methods to deal with the imbalance problem such as resampling can be ineffective due to error heterogeneity as we empirically show in our experimental evaluation.
- **[Heterogeneity]** Heterogeneity amplifies the imbalance problem as certain errors and their side effects can be underrepresented in the training data. Resampling the training data does not ensure that errors with different properties are revealed to the ML model during training. While active learning can help counteract this problem in cases of moderate imbalance [8, 19], it tends to fail in the case of extreme imbalance [26] (as in the case of error detection). This is because the lack of labels prevents the selection scheme of active learning from identifying informative instances for labeling [26]. Different methods that are robust to extreme imbalance are needed.

A solution that addresses the aforementioned challenges needs to: (1) introduce an expressive model for error detection, while avoiding explicit feature engineering; and (2) propose novel ways to handle the extreme imbalance and heterogeneity of data in a unified manner.

1.2 Contributions and Organization

To obviate the need for feature engineering we design a representation learning framework for error detection. To address the heterogeneity and imbalance challenges we introduce a data augmentation methodology for error detection. We summarize the main contributions as follows:

- We introduce a template ML-model to learn a representation that captures attribute-, tuple-, and dataset-level features that describe a dataset. We demonstrate that representation learning obviates the need for feature engineering. Finally, we show via ablation studies that all granularities need to be captured by error detection models to obtain high-quality results.
- We show how to use data augmentation to address data imbalance. Data augmentation proceeds as follows: Given a small set of labeled data, it allows us to generate synthetic examples or errors by transforming correct examples in the available training data. This approach minimizes the amount of manually labeled examples required. We show that in most cases a small number of labeled examples are enough to train high-quality error detection models.
- We present a weakly supervised method to learn data transformations and data augmentation policies (i.e., the distribution over those data transformation) directly from the noisy input dataset. The use of different transformations during augmentation provides us with examples that correspond to different types of errors, which enables us to address the aforementioned heterogeneity challenge.

The remainder of the paper proceeds as follows: In Section 2 we review background concepts. Section 3 provides an overview of our weak supervision framework. In Section 4, we introduce our representation learning approach to error detection. In Section 5, we establish a data augmentation methodology for error detection, and in Section 6, we evaluate our proposed solutions. We discuss related work in Section 7 and summarize key points of the paper in Section 8.

2 BACKGROUND

We review basic background material for the problems and techniques discussed in this paper.

2.1 Error Detection

The goal of error detection is to identify incorrect entries in a dataset. Existing error detection methods can be categorized in three main groups: (1) Rule-based methods [12, 15] rely on integrity constraints such as functional dependencies and denial constraints, and suggest errors based on the violations of these rules. Denial Constraints (DCs) are first order logic formulas that subsume several types of integrity constraints [10]. Given a set of operators $B = \{=, <, >, \neq, \leq, \approx\}$, with $\approx$ denoting similarity, DCs take the form $\forall t_i, t_j \in D : \neg(P_1 \wedge \cdots \wedge P_k \wedge \cdots \wedge P_K)$ where D is a dataset with attributes $A = \{A_1, A_2, \ldots, A_N\}$, t_i and t_j are tuples, and each predicate P_k is of the form $(t_i[A_n]$ op $t_j[A_m])$ or $(t_i[A_n]$ op $\alpha)$ where $A_n, A_m \in A$, α is a constant and op $\in$ B. (2) Pattern-driven methods leverage normative syntactic patterns and identify erroneous entries such as those that do not conform with these patterns [33]. (3) Quantitative error detection focuses on outliers in the data and declares those to be errors [28]. A problem related to error detection is record linkage [17, 18, 43], which tackles the problem of identifying if multiple records refer to the same real-world entity. While it can also be viewed as a classification problem, it does not detect errors in the data and is not the focus of this paper.

2.2 Data Augmentation

Data augmentation is a form of weak supervision [54] and refers to a family of techniques that aim to extend a dataset with additional data points. Data augmentation is typically applied to training data as a way to reduce overfitting of models [62]. Data augmentation methods typically consist of two components: (1) a set of *data transformations* that take a data point as input and generate an altered version of it, and (2) an *augmentation policy* that determines how different transformations should be applied, i.e., a distribution over different transformations. Transformations are typically specified by domain experts while policies can be either pre-specified [45] or learned via reinforcement learning or random search methods [14, 53]. In contrast to prior work, we show that for error detection both transformations and policies can be learned directly from the data.

2.3 Representation Learning

The goal of representation learning is to find an appropriate representation of data (i.e., a set of features) to perform a machine learning task [5]. In our error detection model we build upon three standard representation learning techniques:

Neural Networks Representation learning is closely related to neural networks [24]. The most basic neural network takes as input a vector $\mathbf{x}$ and performs an affine transformation of the input $\mathbf{w}\mathbf{x} + b$. It also applies a non-linear activation function σ (e.g., a sigmoid) to produce the output $\sigma(\mathbf{w}\mathbf{x} + b)$. Multiple layers can be stacked together to create more complex networks. In a neural network, each hidden layer maps its input data to an internal representation that tends to capture a higher level of abstraction.

Highway Neural Networks Highway Networks, adapt the idea of having "shortcut" gates that allow unimpeded information to flow across non-consecutive layers [58]. Highway Networks are used to improve performance in many domains such as speech recognition [63] and language modeling [35], and their variants called Residual networks have been useful for many computer vision problems [27]

Distributed Representations Distributed representations of symbolic data [29] were first used in the context of statistical language model [6]. The goal here is to learn a mapping of a token (e.g., a word) to a vector of real numbers, called a *word embedding*. Methods to generate these mappings include neural networks [40], dimensionality reduction techniques such as PCA [38], and other probabilistic techniques [22].

3 FRAMEWORK OVERVIEW

We formalize the problem of error detection and provide an overview of our solution to error detection.

3.1 Problem Statement

The goal of our framework is to identify erroneous entries in a relational dataset D. We denote $A = \{A_1, A_2, \ldots, A_N\}$ the attributes of D. We follow set semantics and consider D to be a set of tuples. Each tuple $t \in D$ is a collection of cells $C_t = \{t[A_1], t[A_2], \ldots, t[A_N]\}$ where $t[A_i]$ denotes the value of attribute A_i for tuple t. We use C_D to denote the set of cells contained in D. The input dataset D can also be accompanied by a set of integrity constraints Σ, such as Denial Constraints as described in Section 2.1.

We assume that errors in D appear due to inaccurate cell assignments. More formally, for a cell c in C_D we denote by v_c^* its unknown true value and v_c its observed value. We define an *error* in D to be each cell c with $v_c \neq v_c^*$. We define a *training dataset* T to be a set of tuples $T = \{(c, v_c, v_c^*)\}_{c \in C_T}$ where $C_T \subset C_D$. T provides labels (i.e., correct or erroneous) for a subset of cells in D. We also define a variable E_c for each cell $c \in C_D$ with $E_c = -1$ indicating that the cell is erroneous and with $E_c = 1$ indicating that the cell is correct. For each E_c we denote e_c^* its unknown true assignment.

Our goal is stated as follows: given a dataset D and a training dataset T find the most probable assignment $\hat{e}_c$ to

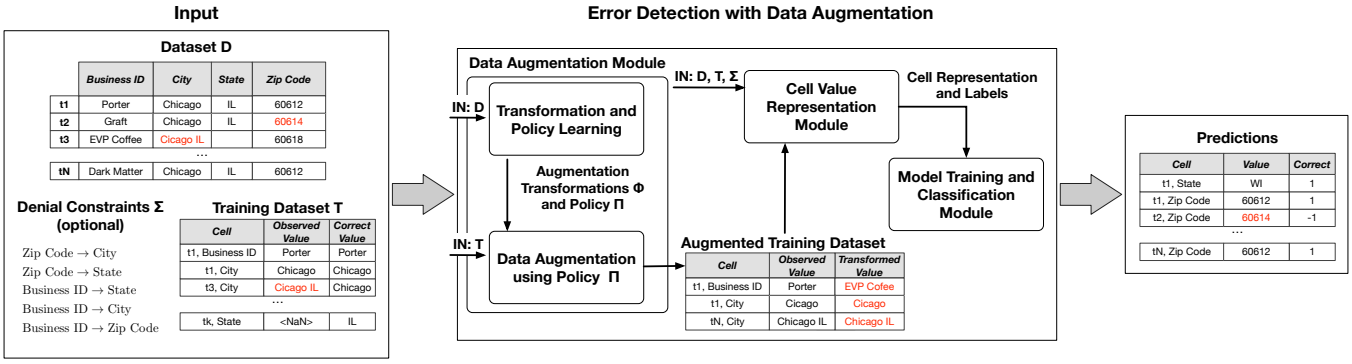

Figure 1: Overview of Error Detection with Augmentation.

each variable E_c with $c \in C_D \setminus C_T$. We say that a cell is correctly classified as erroneous or correct when $\hat{e}_c = e_c^*$.

3.2 Model Overview

Prior models for error detection focus on specific side effects of data errors. For example, they aim to detect errors by using only the violations of integrity constraints or aim to identify outliers with respect to the data distribution that are introduced due to errors. Error detectors that focus on specific side effects, such as the aforementioned ones, are not enough to detect errors with a high recall in heterogeneous datasets [3]. This is because many errors may not lead to violations of integrity constraints, nor appear as outliers in the data. We propose a different approach: we model the process by which the entries in a dataset are generated, i.e., we model the distribution of both correct and erroneous data. This approach enables us to discriminate better between these two types of data.

We build upon our recent Probabilistic Unclean Databases (PUDs) framework that introduces a probabilistic framework for managing noisy relational data [56]. We follow the abstract generative model for noisy data from that work, and introduce an instantiation of that model to represent the distribution of correct and erroneous cells in a dataset.

We consider a noisy channel model for databases that proceeds in two steps: First, a clean database is sampled from a probability distribution I^*. Distribution I^* captures how values within an attribute and across attributes are distributed and also captures the compatibility of different tuples (i.e., it ensures that integrity constraints are satisfied). To this end, distribution I^* is defined over attribute-, tuple-, and dataset-level features of a dataset. Second, given a clean database sampled by I^*, errors are introduced via a noisy channel that is described by a conditional probability distribution R^*. Given this model, I^* characterizes the probability of the unknown true value $P(v_c^*)$ of a cell c and R^* characterizes the conditional probability $P(v_c|v_c^*)$ of its observed value. Distribution I^* is such that errors in dataset D lead to low probability instances. For example, I^* assigns zero probability to datasets with entries that lead to constraint violations.

The goal is to learn a representation that captures the distribution of the correct cells (I^*) and how errors are introduced (R^*). Our approach relies on learning two models:

(1) *Representation Model* We learn a representation model Q that approximates distribution I^* on the attribute, record, and dataset level. We require that Q is such that the likelihood of correct cells given Q will be high, while the likelihood of erroneous cells given Q is low. This property is necessary for a classifier M to discriminate between correct and erroneous cells when using representation Q. We rely on representation learning techniques to learn Q jointly with M.

(2) *Noisy Channel* We learn a generative model H that approximates distribution R^*. This model consists of a set of transformations Φ and a policy Π. Each transformation $\phi \in \Phi$ corresponds to a function that takes as input a cell c and transforms its original value v_c to a new value v_c', i.e., $\phi(v_c) = v_c'$. Policy Π is defined as a conditional distribution $P(\Phi|v_c)$. As we describe next, we use this model to generate training data—via data augmentation—for learning Q and M.

We now present the architecture of our framework. The modules described next are used to learn the noisy channel H, perform data augmentation by using H, and learn the representation model Q jointly with a classifier M that is used to detect errors in the input dataset.

3.3 Framework Overview

Our framework takes as input a noisy dataset D, a training dataset T, and (optionally) a set of denial constraints Σ. To learn H, Q, and M from this input we use three core modules:

Module 1: Data Augmentation This module learns the noisy channel H and uses it to generate additional training examples by transforming some of the labeled examples in

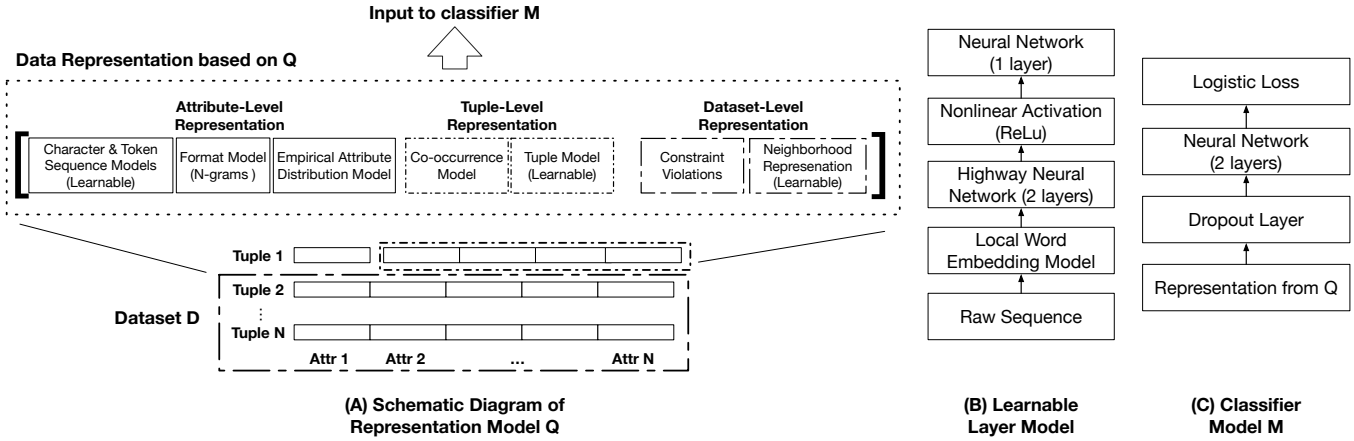

Figure 2: (A) A diagram of the representation model Q. Models associated learnable layers that are jointly trained with classifier M. (B) Architecture diagram of the learnable layers in Q. (C) The architecture of classifier M.

T. The output of this module is a set of additional examples T_H. The operations performed by this module are:

(1) *Transformation and Policy Learning:* The goal here is to learn the set of transformations Φ and the policy Π that follow the data distribution in D. We introduce a weakly supervised algorithm to learn Φ and Π. This algorithm is presented in Section 5.

(2) *Example Generation:* Given transformations Φ and policy Π, we generate a set of new training examples T_H that is combined with T to train the error detection model. To ensure high-quality training data, this part augments only cells that are marked correct in T. Using this approach, we obtain a balanced training set where examples of errors follow the distribution of errors in D. This is because transformations are chosen with respect to policy Π which is learned from D.

Module 2: Representation This module combines different representation models to form model Q. Representation Q maps a cell values v_c to to a fixed-dimension real-valued vector $f_c \in R^d$. To obtain f_c we concatenate the output of different representation models, each of which targets a specific context (i.e., attribute, tuple, or dataset context).

We allow a representation model to be learned during training, and thus, the output of a representation model can correspond to a vector of variables (see Section 4). For example, the output of a representation model can be an embedding u_c obtained by a neural network that is learned during training or may be fixed to the number of constraint violations value v_c participates in.

Module 3: Model Training and Classification This module is responsible for training a classifier M that given the representation of a cell value determines if it is correct or erroneous, i.e., $M : R^d \rightarrow$ {"correct" (+1), "error (-1)"}. During training, the classifier is learned by using both the initial training data T and the augmentation data T_A. At prediction time, the classifier M takes as input the cell value representation for all cells in $D \setminus T$ and assigns them a label from {"correct", "error"} (see Section 4).

An overview of how the different modules are connected is shown in Figure 1. First, Module 1 learns transformations Φ and policy Π. Then, Module 2 grounds the representation model Q of our error detection model. Subsequently, Q is connected with the classifier model M in Module 3 and trained jointly. The combined model is used for error detection.

4 REPRESENTATIONS OF DIRTY DATA

We describe how to construct the representation model Q (see Section 3.2). We also introduce the classifier model M, and describe how we train Q and M.

4.1 Representation Models

To approximate the data generating distribution I^*, the model Q needs to capture statistical characteristics of cells with respect to attribute-level, tuple-level, and dataset-level contexts. An overview of model Q is shown in Figure 2(A). As shown, Q is formed by concatenating the outputs of different models. Next, we review the representation models we use for each of the three contexts. The models introduced next correspond to a bare-bone set that captures all aforementioned contexts, and is currently implemented in our prototype. More details on our implementation are provided in Appendix A.1. Our architecture can trivially accommodate additional models or more complex variants of the current models.

Attribute-level Representation: Models for this context capture the distributions governing the values and format

for an attribute. Separate models are used for each attribute A_i in dataset D. We consider three types of models: (1) *Character and token sequence models* that capture the probability distribution over sequences of characters and tokens in cell values. These models correspond to learnable representation layers. Figure 2(B) shows the deep learning architecture we used for learnable layers. (2) *Format models* that capture the probability distribution governing the format of the attribute. In our implementation, we consider an n-gram model that captures the format sequence over the cell value. Each n-gram is associated with a probability that is learned directly from dataset D. The probabilities are aggregated to a fixed-dimension representation by taking the probabilities associated with the least-k probable n-grams. (3) *Empirical distribution models* that capture the empirical distribution of the attribute associated with a cell. These can be learned directly from the input dataset D. The representation here is a scalar that is the empirical probability of the cell value.

Tuple-level Representation: Models for this context capture the joint distribution of different attributes. We consider two types of models: (1) *Co-occurrence models* that capture the empirical joint distribution over pairs of attributes. (2) A learnable *tuple representation*, which captures the joint distribution across attributes given the observed cell value. Here, we first obtain an embedding of the tuple by following standard techniques based on word-embedding models [7]. These embeddings are passed through a learnable representation layer (i.e., a deep network) that corresponds to an additional non-linear transform (see Figure 2(B)). For co-occurrence, we learn a single representation for all attributes. For tuple embeddings, we learn a separate model per attribute.

Dataset-level Representation: Models for this context capture a distribution that governs the compatibility of tuples and values in the dataset D. We consider two types of models: (1) *Constraint-based models* that leverage the integrity constraints in Σ (if given) to construct a representation model for this context. Specifically, for each constraint $\sigma \in \Sigma$ we compute the number of violations associated with the tuple of the input cell. (2) A *neighborhood-based representation* of each cell value that is informed by a dataset-level embedding of D transformed via a learnable layer. Here, we train a standard word-embedding model where each tuple in D is considered to be a document. To ensure that the embeddings are not affected by the sequence of values across attributes we extend the context considered by word-embeddings to be the entire tuple and treat the tuple as a bag-of-words. These embeddings are given as input to a learnable representation layer that follows the architecture in Figure 2(B).

The outputs of all models are concatenated into a single vector that is given as input to Classifier M. Learnable layers are trained jointly with M. To achieve high-quality error detection, features from all contexts need to be combined to form model Q. In Section 6, we present an ablation study which demonstrates that all features from all types of contexts are necessary to achieve high-quality results.

4.2 Error Classification

The classifier M of our framework corresponds to a two-layer fully-connected neural network, with a ReLU activation layer, and followed by a *Softmax* layer. The architecture of M is shown in Figure 2(C). Given the modular design of our architecture, Classifier M can be easily replaced with other models. Classifier M is jointly trained with the representation model Q by using the training data in T and the data augmentation output T_H. We use ADAM [36] to train our end-to-end model.

More importantly, we calibrate the confidence of the predictions of M using *Platt Scaling* [25, 46] on a holdout-set from the training data T (i.e., we keep a subset of T for calibration). Platt Scaling proceeds as follows: Let z_i be the score for class i output by M. This score corresponds to non-probabilistic prediction. To convert it to a calibrated probability, Platt Scaling learns scalar parameters $a, b \in R$ and outputs $\hat{q}_i = \sigma(az_i + b)$ as the calibrated probability for prediction z_i. Here, σ denotes the sigmoid function. Parameters a and b are learned by optimizing the negative log-likelihood loss over the holdout-set. It is important to note that the parameters of M and Q are fixed at this stage.

5 DATA AUGMENTATION LEARNING

Having established a representation model Q for the data generating distribution I^*, we now move to modeling the noisy channel distribution R^*. We assume the noisy channel can be specified by a set of transformation functions Φ and a policy Π (i.e., a conditional distribution over Φ given a cell value). Our goal is to learn Φ and Π from few example errors and use it to generate training examples to learn model Q.

5.1 Noisy Channel Model

We aim to limit the number of manually labeled data required for error detection. Hence, we consider a simple noisy channel model that can be learned from few and potentially noisy training data. Our noisy channel model treats cell values as strings and introduces errors to a clean cell value v^* by applying a transformation ϕ to obtain a new value $v = \phi(v^*)$. We consider that each function $\phi \in \Phi$ belongs to one of the following three templates:

- Add characters: $\varnothing \longmapsto [a-z]^+$
- Remove characters: $[a-z]^+ \longmapsto \varnothing$
- Exchange characters: $[a-z]^+ \longmapsto [a-z]^+$ (the left side and right side are different)

Given these templates, we assume that the noisy channel model introduces errors via the following generative process: Given a clean input value v^*, the channel samples a transformation ϕ from a conditional distribution $\Pi(v^*) = P(\Phi|v^*)$, i.e., $\phi \sim \Pi(v^*)$ and applies ϕ once to a substring or position of the input cell value. We refer to Π as a *policy*. If the transformation ϕ can be applied to multiple positions or multiple substrings of v^* one of those positions or strings is selected uniformly at random.

For example, to transform Zip Code "60612" to "606152", the noisy channel model we consider can apply the exchange character function $T : 60612 \longmapsto 606152$, i.e., exchange the entire string. Applying the exchange function on the entire cell value can capture misaligned attributes or errors due to completely erroneous values. However, the same transformed string can also be obtained by applying either the exchange character function $T : 12 \longmapsto 152$ on the '12' substring of "60612" or the add character function $T : \varnothing \longmapsto 5$, where the position between '1' and '2' in "60612" was chosen at random. The distribution that corresponds to the aforementioned generative process dictates the likelihood of each of the above three cases.

Given Φ and Π, we can use this noisy channel on training examples that correspond to clean tuples to augment the available training data. However, both Φ and Π have to be learned from the limited number of training data. This is why we adopt the above simple generative process. Despite its simplicity, we find our approach to be effective during data augmentation (see Section 6). Next, we introduce algorithms to learn Φ and Π assuming access to labeled pairs of correct and erroneous values $L = \{(v^*, v)\}$ with $v \neq v^*$. We then discuss how to construct L either by taking a subset of the input training data T or, in the case of limited training data, via an unsupervised approach over dataset D. Finally, we describe how to use Φ and Π to perform data augmentation.

5.2 Learning Transformations

We use a pattern matching approach to learn the transformations Φ. We follow a hierarchical pattern matching approach to identify all different transformations that are valid for each example in L. For example, for $(60612, 6061x2)$ we want to extract the transformations $\{60612 \longmapsto 6061x2, 12 \longmapsto 1x2, \varnothing \longmapsto x\}$. The approach we follow is similar to the Ratcliff-Obershelp pattern recognition algorithm [51]. Due to the generative model we described above, we are agnostic to the position of each transformation.

The procedure is outlined in Algorithm 1. Given an example (v^*, v) from L, it returns a list of valid transformations Φ_e extracted from the example. The algorithm first extracts the string level transformation $T : v^* \longmapsto v$, and then proceeds recursively to extract additional transformations from the substrings of v^* and v. To form the recursion, we identify the longest common substring of v^* and v, and use that to split each string into its prefix (denoted by lv^*) and its postfix (denoted by rv^*). Given the prefix and the postfix substrings, we recurse on the combination of substrings that have the maximum similarity (i.e., overlap). We compute the overlap of two strings as $2 * C/S$, where C is the number of common characters in the two strings, and S is the sum of their lengths. Finally, we remove all identity (i.e., trivial) transformations from the output Φ_e. To construct the set of transformations Φ, we take the set-union of all lists Φ_e generated by applying Algorithm 1 to each entry $e \in L$.

Algorithm 1: Transformation Learning (TL)

Input: Example $e = (v^*, v)$ of a correct string and its corresponding erroneous string
Output: A list of valid transformations Φ_e for example e

1 **if** $v^* = \varnothing$ and $v = \varnothing$ **return** $\emptyset$;
2 $\Phi_e \leftarrow [v^* \longmapsto v]$;
3 $l \leftarrow$ Longest Common Substring(v^*, v);
4 $lv^*, rv^* \leftarrow v^* \setminus l$ /* Generate left and right substrings */;
5 $lv, rv \leftarrow v \setminus l$;
6 **if** *similarity*(lv^*, lv) + *similarity*$(rv^*, rv) >$ *similarity*(lv^*, rv) + *similarity*(rv^*, lv) **then**
7 Add $[lv^* \longmapsto lv,\ rv^* \longmapsto rv]$ in Φ_e;
8 Add $[\text{TL}(lv^*, lv),\ \text{TL}(rv^*, rv)]$ in Φ_e;
9 **else**
10 Add $[lv^* \longmapsto rv,\ rv^* \longmapsto lv]$ in Φ_e;
11 Add $[\text{TL}(lv^*, rv),\ \text{TL}(rv^*, lv)]$ in Φ_e;
12 **end**
13 Remove all identity transformations from Φ_e;
14 **return** Φ_e

5.3 Policy Learning

The set of transformations Φ extracted by Algorithm 1 correspond to all possible alterations our noisy channel model can perform on a clean dataset. Transformations in Φ range from specialized transformations for specific entries (e.g., $60612 \longmapsto 6061x2$) to generic transformations, such as $\varnothing \longmapsto x$, that can be applied to any position of any input. Given Φ, the next step is to learn the transformation policy Π, i.e., the conditional probability distribution $\Pi(v) = P(\Phi|v)$ for any input value v. We next introduce an algorithm to learn Π.

We approximate Π via a two-step process: First, we compute the empirical distribution of transformations informed by the transformation lists output by Algorithm 1. This process is described in Algorithm 2. Second, given an input string v, we find all transformations $str \longmapsto str'$ in Φ such that str is a subset of v. Let $\Phi_v \subseteq \Phi$ be the set of such transformations. We obtain a distribution $P(\Phi_v|v)$ by re-normalizing

Algorithm 2: Empirical Transformation Distribution

Input: A set of identified transformation lists $\{\Phi_e\}_{e \in L}$
Output: Empirical Distribution $\hat{\Pi}$

$\Phi \leftarrow$ Set of unique transformations in $\{\Phi_e\}_{e \in L}$;
$c \leftarrow \sum_e$ (element count of Φ_e);
for $\phi \in \Phi$ **do**
  $c_\phi \leftarrow$ number of times ϕ appears in $\{\Phi_e\}_{e \in L}$;
  $p(\phi) \leftarrow \frac{c_\phi}{c}$
end
return $\{p(\phi)\}_{\phi \in \Phi}$

the empirical probabilities from the first step. This process is outlined in Algorithm 3. Recall that we choose this simple model for Π as the number of data points in L can be limited.

Algorithm 3: Approximate Noisy Channel Policy

Input: An empirical transformation $\hat{\Pi}$ over transformations Φ; A string v
Output: Conditional Distribution $\hat{\Pi}(v) = P(\Phi|v)$

$\hat{\Pi}(v) \leftarrow \varnothing$;
$\Phi_v \leftarrow$ Subset of transformations $str \longmapsto str'$ in Φ such that str is a substring of v;
total mass $\leftarrow \sum_{\phi \in \Phi_v} \hat{\Pi}(\phi)$;
for $\phi \in \Phi_v$ **do**
  $\hat{\Pi}(v)[\phi] \leftarrow \frac{\hat{\Pi}(\phi)}{\text{total mass}}$;
end
return $\hat{\Pi}(v)$

5.4 Generating Transformation Examples

We describe how to obtain examples (v^*, v) to form the set L, which we use in learning the transformations Φ (Section 5.2) and the policy $\hat{\Pi}$ (Section 5.3). First, any example in the training data T that corresponds to an error can be used. However, given the scarcity of errors in some datasets, examples of errors can be limited. We introduce a methodology based on weak-supervision to address this challenge.

We propose a simple unsupervised data repairing model M_R over dataset D and use its predictions to obtain transformation examples (v^*, v). We form examples $(v^*, v) = (\hat{v}, v)$ with $\hat{v} \neq v$ by taking an original cell value v and the repair $\hat{v}$ suggested by M_R. We only require that this model has relatively high-precision. High-precision implies that the repairs performed by M_R are accurate, and thus, the predictions correspond to true errors. This approach enables us to obtain noisy training data that correspond to *good samples* from the distribution of errors in D. We do not require this simple prediction model to have high recall, since we are only after producing example errors, not repairing the whole data set.

We obtain a simple high-precision data repairing model by training a Naïve Bayes model over Dataset D. Specifically, we iterate over each cell in D, pretend that its value is missing and leverage the values of other attributes in the tuple to form a Naïve Bays model that we use to impute the value of the cell. The predicted value corresponds to the suggested repair for this cell. Effectively, this model takes into account value co-occurrence across attributes. Similar models have been proposed in the literature to form sets of potential repairs for noisy cells [55]. To ensure high precision, we only accept only repairs with a likelihood more than 90%. In Section 6, we evaluate our Naïve Bayes-based model and show that it achieves reasonable precision (i.e., above 70%).

5.5 Data Augmentation

To perform data augmentation, we leverage the learned Φ and $\hat{\Pi}$ and use the generative model described in Section 5.1. Our approach is outlined in Algorithm 4: First, we sample a correct example with cell value v from the training data T. Second, we sample a transformation ϕ from distribution $\hat{\Pi}[v]$. If ϕ can be applied in multiple positions or substrings of input v we choose one uniformly at random, and finally, compute the transformed value $v' = \phi(v)$. Value v' corresponds to an error as we do not consider the identity transformation. Finally, we add (v, v') in the set of augmented examples with probability α. Probability α is a hyper-parameter of our algorithm, which intuitively corresponds to the required balance in the overall training data. We set α via cross-validation over a holdout-set that corresponds to a subset of T. This is the same holdout-set used to perform Platt scaling during error classification (see Section 4.2).

Algorithm 4: Data Augmentation

Input: Training set T; Transformations Φ; Approximate Policy $\hat{\Pi}$; Probability α (hyper-parameter)
Output: Set T_H of augmented examples

$T_H \leftarrow \emptyset$;
$T_c \leftarrow$ set of correct examples in T;
$p \leftarrow$ number of correct examples in T;
$n \leftarrow$ number of erroneous examples in T;
/* we assume that p >> n due to imbalance */ ;
while $|T_H| < p - n$ **do**
  Draw a correct example $v \sim Uniform(T_c)$;
  $C \leftarrow$ Flip a coin with probability α;
  if $C =$ `True` *and* $\hat{\Pi}(v) \neq \emptyset$ **then**
    Draw a transformation $\phi \sim \hat{\Pi}(v)$;
    $v' \leftarrow \phi(v)$;
    $T_H \leftarrow T_H \cup \{(v, v')\}$
  end
end

Table 1: Datasets used in our experiments.

Dataset	Size	Attributes	Labeled Data	Errors (# of cells)
Hospital	1,000	19	1,000	504
Food	170,945	15	3,000	1,208
Soccer	200,000	10	200,000	31,296
Adult	97,684	11	97,684	1,062
Animal	60,575	14	60,575	8,077

6 EXPERIMENTS

We compare our approach against a wide-variety of error detection methods on diverse datasets. The main points we seek to validate are: (1) is weak supervision the key to high-quality (i.e., high-precision and high-recall) error detection models, (2) what is the impact of different representation contexts on error detection, (3) is data augmentation the right approach to minimizing human exhaust. We also perform extensive micro-benchmark experiments to examine the effectiveness and sensitivity of data augmentation.

6.1 Experimental Setup

We describe the dataset, metrics, and settings we use.

Datasets: We use five datasets from a diverse array of domains. Table 1 provides information for these datasets. As shown the datasets span different sizes and exhibit various amounts of errors: (1) The Hospital dataset is a benchmark dataset used in several data cleaning papers [12, 55]. Errors are artificially introduced by injecting typos. This is an easy benchmark dataset; (2) The Food dataset contains information on food establishments in Chicago. Errors correspond to conflicting zip codes for the same establishment, conflicting inspection results for the same establishment on the same day, conflicting facility types for the same establishment and many more. Ground truth was obtained by manually labeling 3,000 tuples; (3) The Soccer dataset provides information about soccer players and their teams. The dataset and its ground truth are provided by Rammerlaere and Geerts [49]; (4) Adult contains census data is a typical dataset from the UCI repository. Adult is also provided by Rammerlaere and Geerts [49]; (5) Animal was provided by scientists at UC Berkeley and has been used by Abedjan et al. [2] as a testbed for error detection. It provides information about the capture of animals, including the time and location of the capture and other information for each captured animal. The dataset comes with manually curated ground truth. The datasets used in our experiments exhibit different error distributions. Hospital contains only typos, Soccer [49] and Adult [49] have errors that were introduced with BART [4]: Adult has 70% typos and 30% value swaps, and Soccer has 76% typos and 24% swaps. Finally, the two datasets with real-world errors have the following error distributions: Food has 24% typos and 76% value swaps (based on the sampled ground truth); Animal has 51% typos and 49% swaps.

Methods: We compare our approach, referred to as AUG, against several competing error detection methods. First, we consider three baseline error detection models:

- **Constraint Violations (CV)**: This method identifies errors by leveraging violations of denial constraints. It is a proxy for rule-based errors detection methods [12].
- **HoloClean (HC)**: This method combines CV with HoloClean [55], a state-of-the-art data repairing engine. This method aims to improve the precision of the CV detector by considering as errors not all cells in tuples that participate in constraint violations but only those cells whose value was repaired (i.e., their initial value is changed to a different value).
- **Outlier Detection (OD)**: This method follows a correlation based outlier detection approach. Given a cell that corresponds to an attribute A_i, the method considers all correlated attributes in $A \setminus A_i$ with A_i rely on the pair-wise conditional distributions to detect if the value of a cell corresponds to an outlier.
- **Forbidden Item Sets (FBI)**: This method captures unlikely value co-occurrences in noisy data [50]. At its core, this method leverages the *lift* measure from association rule mining to identify how probably a value co-occurrence is, and uses this measure to identify erroneous cell values.
- **Logistic Regression (LR)**: This method corresponds to a supervised logistic regression model that classifies cells are erroneous or correct. The features of this model correspond to pairwise co-occurrence statistics of attribute values and constraint violations. This model corresponds to a simple supervised ensemble over the previous two models.

We also consider three variants of our model where we use different training paradigms. The goal is to compare data augmentation against other types of training. For all variations, we use the representation Q and the classifier M introduced in Section 3. We consider the following variants:

- **Supervised Learning (SuperL)**: We train our model using only the training examples in T.
- **Semi-supervised Learning (SemiL)**: We train our model using self-training [64]. First supervised learning used to train the model on the labeled data only. The learned model is then applied to the entire dataset to generate more labeled examples as input for a subsequent round of supervised learning. Only labels with high confidence are added at each step.
- **Active Learning (ActiveL)**: We train our model using an active learning method based on uncertainty sampling [57]. First, supervised learning is used to

Table 2: Precision, Recall and F_1-score of different methods for different datasets. AL results correspond to $k = 100$.

Dataset (T size)	M	AUG	CV	HC	OD	FBI	LR	SuperL	SemiL	ActiveL
Hospital (10%)	P	0.903	0.030	0.947	0.640	0.008	0.0	0.0	0.0	0.960
	R	0.989	0.372	0.353	0.667	0.001	0.0	0.0	0.0	0.613
	F_1	**0.944**	0.055	0.514	0.653	0.003	0.0	0.0	0.0	0.748
Food (5%)	P	0.972	0.0	0.0	0.240	0.0	0.0	0.985	0.813	0.990
	R	0.939	0.0	0.0	0.99	0.0	0.0	0.95	0.66	0.91
	F_1	**0.955**	0.0	0.0	0.387	0.0	0.0	0.948	0.657	0.948
Soccer (5%)	P	0.922	0.039	0.032	0.999	0.0	0.721	0.802	n/a[#]	0.843
	R	1.0	0.846	0.632	0.051	0.00	0.084	0.450	n/a	0.683
	F_1	**0.959**	0.074	0.061	0.097	0.00	0.152	0.577	n/a	0.755
Adult (5%)	P	0.994	0.497	0.893	0.999	0.990	0.051	0.999	n/a	0.994
	R	0.987	0.998	0.392	0.001	0.254	0.072	0.350	n/a	0.982
	F_1	**0.991**	0.664	0.545	0.002	0.405	0.059	0.519	n/a	0.988
Animal (5%)	P	0.832	0.0	0.0	0.85	0.0	0.185	0.919	n/a	0.832
	R	0.913	0.0	0.0	6×10^{-5}	0.0	0.028	0.231	n/a	0.740
	F_1	**0.871**	0.0	0.0	1×10^{-4}	0.0	0.048	0.369	n/a	0.783

[#] n/a = Semi-supervised learning did not terminate after two days.

train the model. At each subsequent round, we use an uncertainty-based selection scheme to obtain additional training examples and re-train the model. We use k to denote the number of iterations. In our implementation, we set the upper limit of labeled examples obtained per iteration to be 50 cells.

Evaluation Setup: To measure accuracy, we use Precision (P) defined as the fraction of error predictions that are correct; Recall (R) defined as the fraction of true error being predicted as errors by the different methods; and F_1 defined as $2PR/(P+R)$. For training, we split the available ground truth into three disjoint sets: (1) a training set T, from which 10% is always kept as a hold-out set used for hyper parameter tuning; (2) a sampling set, which is used to obtain additional labels for active learning; and (3) a test set, which is used for evaluation. To evaluate different dataset splits, *we perform* 10 *runs with different random seeds for each experiment.* To ensure that we maintain the coupling amongst Precision, Recall, and F_1, we report the median performance. The mean performance along with standard error measurements are reported in the Appendix. *Seeds are sampled at the beginning of each experiment, and hence, a different set of random seeds can be used for different experiments.* We use *ADAM* [36] as the optimization algorithm for all learning-based model and train all models for 500 epochs with a batch-size of five examples. We run Platt Scaling for 100 epochs. All experiments were executed on a 12-core Intel(R) Xeon(R) CPU E5-2603 v3 @ 1.60GHz with 64GB of RAM running Ubuntu 14.04.3 LTS.

6.2 End-to-end Performance

We evaluate the performance of our approach and competing approaches on detecting errors in all five datasets. Table 2 summarizes the precision, recall, and F_1-score obtained by different methods. For Food, Soccer, Adult, and Animal we set the amount of training data to be 5% of the total dataset. For Hospital we set the percentage of training data to be 10% (corresponding to 100 tuples) since Hospital is small. For Active Learning we set the number of active learning loops to $k = 100$ to maximize performance.

As Table 2 shows, our method consistently outperforms all methods, and in some cases, like Hospital and Soccer, we see improvements of 20 F_1 points. More importantly, we find that our method is able to achieve both high recall and high precision in all datasets despite the different error distribution in each dataset. This is something that has been particularly challenging for prior error detection methods. We see that for Food and Animal, despite the fact that most errors do not correspond to constraint violations (as implied by the performance of CV), AUG can obtain high precision and recall. This is because AUG models the actual data distribution and not the side-effects of errors. For instance, for Food we see that OD can detect many of the errors—it has high recall—indicating that most errors correspond to statistical outliers. We see that AUG can successfully solve error detection for this dataset. Overall, our method achieves an average precision of 92% and an average recall of 96% across these diverse datasets. At the same time, we see that the performance of competing methods varies significantly across datasets. This validates the findings of prior work [2] that depending on the side effects of errors different error detection methods are more suitable for different datasets.

We now discuss the performance of individual competing methods. For CV, we see that it achieves higher recall than precision. This performance is due to the fact that CV marks as erroneous all cells in a group of cells that participate in a violation. More emphasis should be put on the recall-related results of CV. As shown its recall varies dramatically from 0.0 for Food and Animal to 0.998 for Adult. For OD, we see that it achieves relatively high-precision results, but its recall

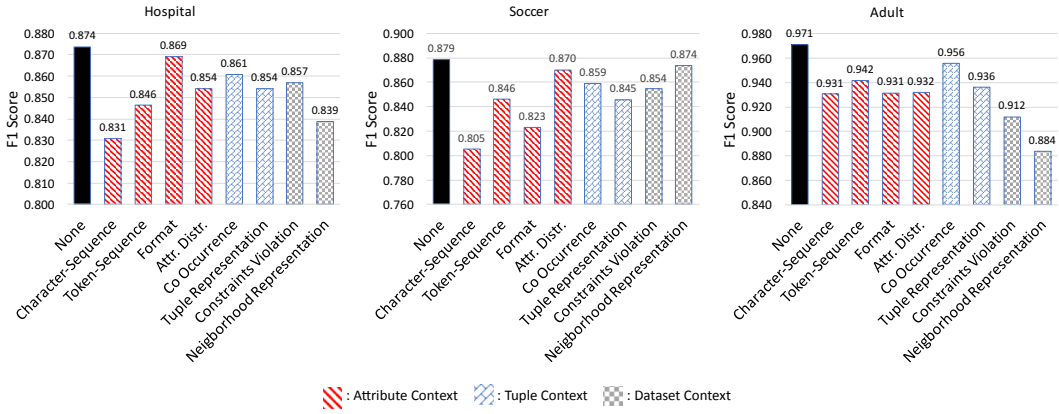

Figure 3: Ablation studies to evaluate the effect of different representation models.

is low. Similar performance is exhibited by FBI that leverages a different measure for outlier detection. We see that FBI achieves high precision when the forbidden item sets have significant support (i.e., occur relatively often). However, FBI cannot detect errors that lead to outlier values which occur a limited number of times. This is why we find OD to outperform FBI in several cases.

Using HC as a detection tool is limited to these cells violating integrity constraints. Hence, using HC leads to improved precision over CV (see Hospital and Adult). This result is expected as data repairing limits the number of cells detected as erroneous to only those whose values are altered. Our results also validate the fact that HC depends heavily on the quality of the error detection used [55]. As shown in Food and Animal, the performance of HC is limited by the recall of CV, i.e., since CV did not detect errors accurately, HC does not have the necessary training data to learn how to repair cells. At the same time, Soccer reveals that training HC on few clean cells—the recall of CV is very high while the precision is very low indicating that most cells were marked as erroneous—leads to low precision (HC achieves a precision of 0.032 for Soccer). This validates our approach of solving error detection separately from data repairing.

We also see that LR has consistently poor performance. This result reveals that combining co-occurrence features and violations features in a linear way (i.e., via a weighted linear combination such as in LR) is not enough to capture the complex statistics of the dataset. *This validates our choice of using representation learning and not engineered features.*

Finally, we see that approaches that rely on representation learning model achieve consistently high precision across all datasets. This validates our hypothesis that modeling the distribution of both correct and erroneous data allows us to discriminate better. However, we see that when we rely only on the training dataset T the recall is limited (see the recall for SuperL). The limited labeled examples in T is not sufficient to capture the heterogeneity of errors. Given additional training examples either via Active Learning or via Data Augmentation helps improve the recall. However, Data Augmentation is more effective than Active Learning at capturing the heterogeneity of errors in each dataset, and hence, achieves superior recall to Active Learning in all cases. **Takeaway:** The combination of representation learning techniques with data augmentation is key to obtaining high-quality error detection models.

6.3 Representation Ablation Study

We perform an ablation study to evaluate the effect of different representation models on the quality of our model. Specifically, we compare the performance of AUG when all representation models are used in Q versus variants of AUG where one model is removed at a time. We report the F_1-score of the different variants as well as the original AUG in Figure 3. Representation models that correspond to different contexts are grouped together.

Removing any feature has an impact on the quality of predictions of our model. We find that removing a single representation model results in drops of up to 9 F_1 points across datasets. More importantly, we find that different representation models have different impact on different datasets. For instance, the biggest drop for Hospital and Soccer is achieved when the character-sequence model is removed while for Adult the highest drop is achieved when the Neighborhood representation is removed. This validates our design of considering representation models from different contexts. **Takeaway:** It is necessary to leverage cell representations that are informed by different contexts to provide robust and high-quality error detection solutions.

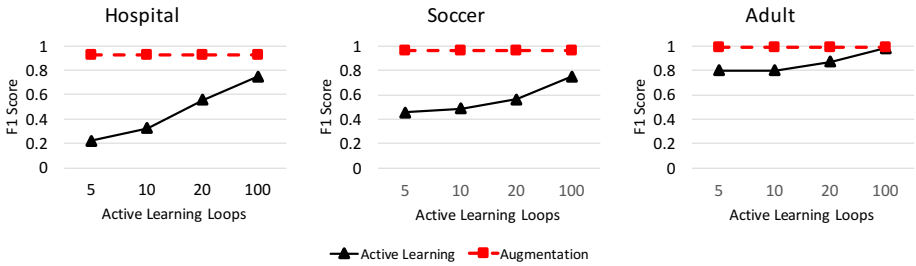

Figure 4: Data augmentation versus active learning as the number of active learning loops increases.

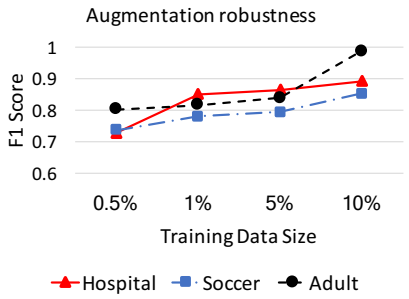

Figure 5: Data augmentation performance for various amounts of training data.

6.4 Augmentation versus Active Learning

We validate the hypothesis that data augmentation is more effective than active learning in minimizing human effort in training error detection models. In Table 2, we showed that data augmentation outperforms active learning. Furthermore, active learning needs to obtain more labeled examples to achieve comparable performance to data augmentation. In the next two experiments, we examine the performance of the two approach as we limit their access to training data.

In the first experiment, we evaluate active learning for different values of loops (k) over Hospital, Soccer, and Adult. We vary k in $\{5, 10, 20, 100\}$. We fix the amount of available training data to 5%. Each time we measure the F_1 score of the two algorithms. We report our results in Figure 4. Reported results correspond to median performance over ten runs.

We see that when a small number of loops is used (k=5), there is a significant gap between the two algorithms that ranges between 10 and 70 F_1 points. Active learning achieves comparable performance with data augmentation only after 100 loops. This corresponds to an additional $5,000$ ($k \times 50$) labeled examples (labeled cells). This behavior is consistent across all three datasets. In the second experiment, we seek to push data augmentation to the limits. Specifically, we seek to answer the question, can data augmentation be effective when the number of labeled examples in T is extremely small.

Table 3: A comparison between data augmentation and resampling. We report the F_1-score as we increase the size of the training data T. We also include supervised learning as a baseline.

Dataset	Size of T	AUG	Resampling	SuperL
Hospital	1%	**0.840**	0.041	0.0
	5%	**0.873**	0.278	0.0
	10%	**0.925**	0.476	0.079
Soccer	1%	**0.927**	0.125	0.577
	5%	**0.935**	0.208	0.654
	10%	**0.953**	0.361	0.675
Adult	1%	**0.844**	0.063	0.0
	5%	**0.953**	0.068	0.294
	10%	**0.975**	0.132	0.519

To this end, we evaluate the performance of our system on Hospital, Soccer, and Adult as we vary the size of the training data in $\{0.5\%, 1\%, 5\%, 10\%\}$. The results are shown in Figure 5. As expected the performance of data augmentation is improving as more training data become available. However, we see that data augmentation can achieve good performance—F_1 score does not drop below 70%—even in cases where labeled examples T are limited. These results provide positive evidence that data augmentation is a viable approach for minimizing user exhaust.

Takeaway: Our data augmentation approach is preferable to active learning for minimizing human exhaust.

6.5 Augmentation and Data Imbalance

We evaluate the effectiveness of data augmentation to counteract imbalance. Table 2 shows that using data augmentation yields high-quality error detection models for datasets with varying percentages of errors. Hence, data augmentation is robust to different levels of imbalance; each dataset in Table 2 has a different ratio of true errors to correct cells.

In Table 3, we compare data augmentation with traditional methods used to solve the imbalance problem, namely, resampling. In all the datasets, resampling had low precision

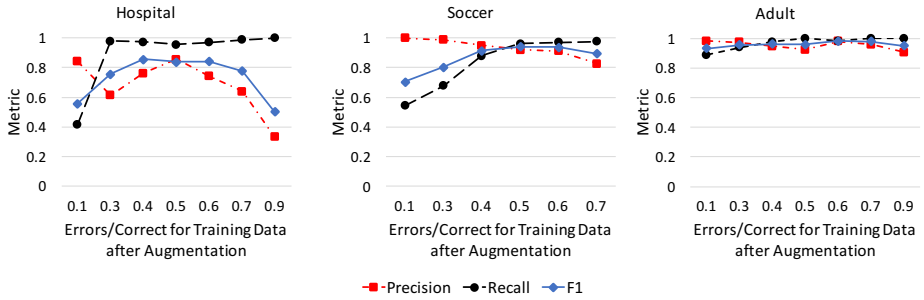

Figure 6: The effect of increasing the number of examples that correspond to errors via data augmentation.

and recall confirming our hypothesis discussed in Section 1: due to the heterogeneity of the errors, resampling from the limited number of negative examples was not enough to cover all types of errors. The best result for resampling was obtained in the Hospital data set (F_1 about 47%), since errors are more homogeneous than other data sets.

We also evaluate the effect of excessive data augmentation: In Algorithm 4 we do not use hyper-parameter α to control how many artificial examples should be generated via data augmentation. We manually set the ratio between positive and negative examples in the final training examples and use augmentation to materialize this ratio.

Our results are reported in Figure 6. We show that increasing the number of generated negative examples (errors) results in a lower accuracy as the balance between errors and correct example goes greater than 50%, as the model suffers from the imbalance problem again, this time as too few correct examples. We see that peak performance is achieved when the training data is almost balanced for all datasets. This reveals the robustness of our approach. Nonetheless, peak performance is not achieved exactly at a 50-50 balance (peak performance for Adult is at 60%). This justifies our model for data augmentation presented in Algorithm 4 and the use of hyper-parameter α.

Takeaway: Data augmentation is an effective way to counteract imbalance in error detection.

6.6 Analysis of Augmentation Learning

In this experiment, we validate the importance of learning the augmentation model (the transformations Φ, and the policy $\hat{\Pi}$). We compare three augmentation strategies: (1) Random transformations *Rand. Trans.*, where we randomly choose from a set of errors (e.g., typos, attribute value changes, attribute shifts, etc.). Here, we augment the data by using completely random transformations not inspired by the erroneous examples or the data; and (2) learned transformation Φ, but without learning the distribution policy(*Aug w/o Policy*). Given an input, we find all valid transformations in Φ and pick one uniformly at random. Table 4 shows the results for the three approaches. AUG outperforms the other two strategies. Rand. Trans. fails to capture the errors that exist in the dataset. For instance, it obtains a recall of 16.6% for Soccer. Even though the transformations are learned from the data, it is the results show that using these transformations in a way that conform with the distribution of the data is crucial in learning an accurate classifier.

Takeaway: Learning a noisy channel model from the data, i.e., a set of transformations Φ and a policy Π is key to obtaining high-quality predictions.

Table 4: A comparison between different data augmentation approaches. We report the F_1-score as we increase the size of the training data T.

Dataset	T	AUG	Rand. Trans.	AUG w/o Policy
Hospital	5%	**0.911**	0.873	0.866
	10%	**0.943**	0.884	0.870
Soccer	5%	**0.946**	0.212	0.517
	10%	**0.953**	0.166	0.522
Adult	5%	**0.977**	0.789	0.754
	10%	**0.984**	0.817	0.747

6.7 Other Experiments

Finally, we report several benchmarking results: (1) we measure the runtime of different methods, (2) validate the performance of our unsupervised Naïve Bayes model for generating labeled example to learn transformations Φ and Π (see Section 5.5), and (3) validate the robustness of AUG to misspecified denial constraints.

The median runtime of different methods is reported in Table 5. These runtimes correspond to prototype implementations of the different methods in Python. Also recall, that training corresponds to 500 epochs with low batch-size as reported in Section 6.1. As expected iterative methods such as SemiL and ActiveL are significantly slower than non-iterative

Table 5: Runtimes in seconds. Value n/a means that the method did not terminate after running for two days.

Approach	Hospital	Soccer	Adult
AUG	749.17	7684.72	6332.13
CV	204.62	1610.02	1359.46
OD	212.7	1588.06	1423.69
LR	347.95	3505.60	4408.27
SuperL	648.34	3928.46	3310.71
SemiL	14985.15	n/a	n/a
ActiveL	3836.15	56535.19	128132.56

Table 6: Performance of our weak supervision method for generating training examples for AUG.

Dataset	Precision	Recall
Hospital	0.895	0.636
Soccer	0.999	0.053
Adult	0.714	0.973

ones. Overall, we see that AUG exhibits runtimes that are of the same order of magnitude as supervised methods.

The performance of our Naïve Bayes-based weak supervision method on Hospital, Soccer, and Adult is reported in Table 6. Specifically, we seek to validate that the precision of our weak supervision method is reasonable, and thus, by using it we obtain good examples that correspond to good examples from the true error distribution. We see that our weak supervision method achieves a precision of more than 70% in all cases. As expected its recall can be some times low (e.g., for Soccer it is 5.3%) as emphasis is put on precision.

Finally, we evaluate AUG against missing and noisy constraints. The detailed results are presented in Appendix A.2 due to space restrictions. In summary, we find AUG to exhibit a drop of at most 6 F_1 points when only 20% of the original constraints are used to missing constraints and at most 8 F_1 points when noisy constraints are used.

7 RELATED WORK

Many algorithms and prototypes have been proposed for developing data cleaning tools [17, 20, 31, 48]. Outlier detection and quantitative data cleaning algorithms are after data values that looks "abnormal" with respect to the data distribution [16, 47, 61]. Entity resolution and record de-duplication focus on identifying clusters of records that represent the same real-world entity [18, 43]. Example de-duplication tools include the `Data Tamer` system [59], which is commercialized as `Tamr`. Rule-based detection proposals [1, 12, 21, 37, 60] use integrity constraints (e.g., denial constraints) to identify violations, and use the overlap among these violations to detect data errors. Prototypes such as such as `Nadeef` [15], and `BigDansing` [34] are example extensible rule-based cleaning systems. There have been also multiple proposals that identify data cells that don not follow a data "pattern". Example tools include `OpenRefine`, `Data Wrangler` [33] and its commercial descendant `Trifacta`, `Katara` [13], and `DataXFormer` [3]. An overview of these tools and how they can be combined for error detection is discussed in [2], where the authors show that even when all are used, these tools often achieve low recall in capturing data errors in real data sets.

Data Augmentation has also been used extensively in machine learning problems. Most state-of-the-art image classification pipelines use some limited for of data augmentation [45]. This consists of applying crops, flips, or small affine transformations in fixed order or at random. Other studies have applied heuristic data augmentation to modalities such as audio [42] and text [39]. To our knowledge, we are the first to apply data augmentation in relational data.

Recently, several lines of work have explored the use of reinforcement learning or random search to learn more principled data augmentation policies [14, 53]. Our work here is different as we do not rely on expensive procedures to learn the augmentation policies. This is because we limit our policies to applying a single transformation at a time. Finally, recent work has explored techniques based on Generative Adversarial Networks [23] to learn data generation models used for data augmentation from unlabeled data [41]. This work focuses mostly on image data. Exploring this direction for relational data is an exciting future direction.

8 CONCLUSIONS

We introduced a few-shot learning error detection framework. We adopt a noisy channel model to capture how both correct data and errors are generated use it to develop an expressive classifier that can predict, with high accuracy, whether a cell in the data is an error. To capture the heterogeneity of data distributions, we learn a rich set of representations at various granularities (attribute-level, record-level, and the dataset-level). We also showed how to address a main hurdle in this approach, which is the scarcity of error examples in the training data, and we introduced an approach based on data augmentation to generate enough examples of data errors. Our data augmentation approach learns a set of transformations and the probability distribution over these transformations from a small set of examples (or in a completely unsupervised way). We showed that our approach achieved an average precision of ~94% and an average recall of ~93% across a diverse array of datasets. We also showed how our approach outperforms previous techniques ranging from traditional rule-based methods to more complex ML-based method such as active learning approaches.

9 ACKNOWLEDGEMENTS

This work was supported by Amazon under an ARA Award, by NSERC under a Discovery Grant, and by NSF under grant IIS-1755676.

REFERENCES

[1] Ziawasch Abedjan, Cuneyt Akcora, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. 2015. Temporal Rules Discovery for Web Data Cleaning. *PVLDB* 9, 4 (2015), 336 –347.

[2] Ziawasch Abedjan, Xu Chu, Dong Deng, Raul Castro Fernandez, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Michael Stonebraker, and Nan Tang. 2016. Detecting data errors: Where are we and what needs to be done? *Proceedings of the VLDB Endowment* 9, 12 (2016), 993–1004.

[3] Ziawasch Abedjan, John Morcos, Ihab F. Ilyas, Paolo Papotti, Mourad Ouzzani, and Michael Stonebraker. 2016. DataXFormer: A Robust Transformation Discovery System. In *ICDE*.

[4] P. C. Arocena, B. Glavic, G. Mecca, R. J. Miller, P. Papotti, and D. Santoro. 2015. Messing-Up with BART: Error Generation for Evaluating Data Cleaning Algorithms. *PVLDB* 9, 2 (2015), 36–47.

[5] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation Learning: A Review and New Perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 8 (Aug. 2013), 1798–1828.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A Neural Probabilistic Language Model. *J. Mach. Learn. Res.* 3 (March 2003), 1137–1155.

[7] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.

[8] Nitesh V Chawla, Nathalie Japkowicz, and Aleksander Kotcz. 2004. Special issue on learning from imbalanced data sets. *ACM Sigkdd Explorations Newsletter* 6, 1 (2004), 1–6.

[9] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS 2016)*. 7–10.

[10] Jan Chomicki and Jerzy Marcinkowski. 2005. Minimal-change Integrity Maintenance Using Tuple Deletions. *Inf. Comput.* 197, 1-2 (Feb. 2005), 90–121. https://doi.org/10.1016/j.ic.2004.04.007

[11] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Discovering denial constraints. *PVLDB* 6, 13 (2013), 1498–1509.

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

[13] Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 1247–1261.

[14] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. 2018. AutoAugment: Learning Augmentation Policies from Data. *arXiv preprint arXiv:1805.09501* (2018).

[15] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: a commodity data cleaning system. In *SIGMOD*. ACM, 541–552.

[16] Tamraparni Dasu and Ji Meng Loh. 2012. Statistical Distortion: Consequences of Data Cleaning. *PVLDB* 5, 11 (2012), 1674–1683.

[17] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. 2012. *Principles of Data Integration*. Morgan Kaufmann.

[18] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. 2007. Duplicate Record Detection: A Survey. *IEEE Transactions on Data Engineering* 19, 1 (2007).

[19] Seyda Ertekin, Jian Huang, and C. Lee Giles. 2007. Active Learning for Class Imbalance Problem *(SIGIR '07)*. ACM, New York, NY, USA, 823–824.

[20] W. Fan and F. Geerts. 2012. *Foundations of Data Quality Management*. Morgan & Claypool.

[21] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2012. Towards certain fixes with editing rules and master data. *The VLDB journal* 21, 2 (2012), 213–238.

[22] Amir Globerson, Gal Chechik, Fernando Pereira, and Naftali Tishby. 2007. Euclidean Embedding of Co-occurrence Data. *JMLR* 8 (Dec. 2007), 2265–2295.

[23] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2672–2680.

[24] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press, Cambridge, MA, USA.

[25] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On Calibration of Modern Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 1321–1330.

[26] Haibo He and Yunqian Ma. 2013. *Imbalanced Learning: Foundations, Algorithms, and Applications* (1st ed.). Wiley-IEEE Press.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[28] Joseph M Hellerstein. 2008. Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)* (2008).

[29] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart. 1986. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. MIT Press, Cambridge, MA, USA, Chapter Distributed Representations, 77–109.

[30] Zhipeng Huang and Yeye He. 2018. Auto-Detect: Data-Driven Error Detection in Tables. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. 1377–1392.

[31] Ihab F. Ilyas and Xu Chu. 2015. Trends in Cleaning Relational Data: Consistency and Deduplication. *Foundations and Trends in Databases* 5, 4 (2015), 281–393.

[32] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759* (2016).

[33] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 3363–3372.

[34] Zuhair Khayyat, Ihab F. Ilyas, Alekh Jindal, Samuel Madden, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. 2015. BigDansing: A System for Big Data Cleansing. In *SIGMOD*. 1215–1230.

[35] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-Aware Neural Language Models.. In *AAAI*. 2741–2749.

[36] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014).

[37] Solmaz Kolahi and Laks V. S. Lakshmanan. 2009. On Approximating Optimum Repairs for Functional Dependency Violations. In *ICDT*.

[38] Rémi Lebret and Ronan Collobert. 2014. Word Embeddings through Hellinger PCA. EACL.
[39] Xinghua Lu, Bin Zheng, Atulya Velivelli, and ChengXiang Zhai. 2006. Enhancing text categorization with semantic-enriched representation and training data augmentation. *Journal of the American Medical Informatics Association* 13, 5 (2006), 526–535.
[40] Tomas Mikolov, Ilya Sutskever, Kai Chen, et al. 2013. Distributed Representations of Words and Phrases and Their Compositionality. NIPS.
[41] Mehdi Mirza and Simon Osindero. 2014. Conditional Generative Adversarial Nets. *CoRR* abs/1411.1784 (2014).
[42] Stefan Uhlich; Marcello Porcu; Franck Giron; Michael Enenkl; Thomas Kemp; Naoya Takahashi; Yuki Mitsufuji. 2017. Improving Music Source Separation based on DNNs through Data Augmentation and Network Blending. (2017).
[43] Felix Naumann and Melanie Herschel. 2010. *An Introduction to Duplicate Detection.* Morgan & Claypool Publishers.
[44] Jason W Osborne. 2013. *Best practices in data cleaning: A complete guide to everything you need to do before and after collecting your data.* Sage.
[45] Luis Perez and Jason Wang. 2017. The Effectiveness of Data Augmentation in Image Classification using Deep Learning. *CoRR* abs/1712.04621 (2017).
[46] J. Platt. 2000. Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In *Advances in Large Margin Classifiers.*
[47] Nataliya Prokoshyna, Jaroslaw Szlichta, Fei Chiang, Renée J Miller, and Divesh Srivastava. 2015. Combining quantitative and logical data cleaning. *PVLDB* 9, 4 (2015), 300–311.
[48] Erhard Rahm and Hong-Hai Do. 2000. Data Cleaning: Problems and Current Approaches. *DE* 23(4) (2000), 3–13.
[49] Joeri Rammelaere and Floris Geerts. 2018. Explaining Repaired Data with CFDs. *Proc. VLDB Endow.* 11, 11 (July 2018), 1387–1399.
[50] Joeri Rammelaere, Floris Geerts, and Bart Goethals. 2017. Cleaning data with forbidden itemsets. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on.* IEEE, 897–908.
[51] John W. Ratcliff and David E. Metzener. 1988. Pattern Matching: The Gestalt Approach. *Dr. Dobb's Journal of Software Tools* 13, 7 (July 1988), 46, 47, 59–51, 68–72.
[52] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment* 11, 3 (2017), 269–282.
[53] Alexander J. Ratner, Henry R. Ehrenberg, Zeshan Hussain, Jared Dunnmon, and Christopher Ré. 2017. Learning to Compose Domain-Specific Transformations for Data Augmentation, See [53], 3239–3249.
[54] Christopher Ré. 2018. Software 2.0 and Snorkel: Beyond Hand-Labeled Data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* ACM, 2876–2876.
[55] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1190–1201.
[56] Christopher De Sa, Ihab F. Ilyas, Benny Kimelfeld, Christopher Ré, and Theodoros Rekatsinas. 2019. A Formal Framework for Probabilistic Unclean Databases *(ICDT).*
[57] Burr Settles. 2012. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6, 1 (2012), 1–114.
[58] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks. *arXiv preprint arXiv:1505.00387* (2015).
[59] Michael Stonebraker, Daniel Bruckner, Ihab F. Ilyas, George Beskales, Mitch Cherniack, Stan Zdonik, Alexander Pagan, and Shan Xu. 2013. Data Curation at Scale: The Data Tamer System. In *CIDR.*
[60] Jiannan Wang and Nan Tang. 2014. Towards dependable data repairing with fixing rules. In *SIGMOD.* 457–468.
[61] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining Away Outliers in Aggregate Queries. *PVLDB* 6, 8 (June 2013), 553–564.
[62] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2017. Understanding deep learning requires rethinking generalization. https://arxiv.org/abs/1611.03530
[63] Yu Zhang, Guoguo Chen, Dong Yu, Kaisheng Yaco, Sanjeev Khudanpur, and James Glass. 2016. Highway long short-term memory rnns for distant speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on.* IEEE, 5755–5759.
[64] Xiaojin Zhu. 2007. Semi-supervised learning tutorial. In *International Conference on Machine Learning (ICML).* 1–135.

A APPENDIX

We provide additional details for the representation models in our framework and present additional micro-benchmark experimental results on the robustness of our error detection approach to noisy denial constraints.

A.1 Details on Representation Models

Our model follows the wide and deep architecture of Cheng et al. [9]. Thus the model can be thought of as a representation stage, where each feature is being operated on in isolation, and an inference step in which each feature has been concatenated to make a joint representation. The joint representation is then fed through a two-layer neural network. At training time, we backpropogate through the entire network jointly, rather than training specific representations. Figure 7 illustrates this model's topology.

A summary of representation models used in our approach along with their dimensions is provided in Table 7. As shown we use a variety of models that capture all three attribute-level, tuple-level, and dataset-level contexts. We next discuss the embedding-based models and format models we use.

Embedding-based Models: We treat different views of the data as expressing different language models, and so embed each to capture their semantics. The embeddings are taken at a character, cell and tuple level tokens, and each uses a FastText Embedding in 50 dimensions [7, 32]. Rather than doing inference directly on the embeddings, we employ a two-step process of a non-linear transformation and dimensionality reduction. At the non-linear transformation stage, we use a two-layer Highway Network [58] to extract useful representations of the data. Then, a dense layer is used to reduce the dimensionality to a single dimension. In this way, the embeddings do not dominate the joint representation. Figure 2(B) shows this module more explicitly.

In addition to using these singular embeddings, we also use a distance metric on the learned corpus as a signal to be fed into the model (see Neighborhood representation). The intuition behind this representation is that in the presence of other signals that would imply a cell is erroneous, there may

Table 7: A summary of representation models used in our approach along with their dimension.

Context	Representation Type	Description	Dimension
Attribute-Level	Character Embedding	FastText Embedding where tokens are characters	1
	Word Embedding	FastText Embedding where tokens are words in the cell	1
	Format models	3-Gram: Frequency of the least frequent 3-gram in the cell	1
	Format models	Symbolic 3-Gram; each character is replaced by a token $\{Char, Num, Sym\}$	1
	Empirical distribution model	Frequency of cell value	1
	Empirical distribution model	One Hot Column ID; Captures per-column bias	1
Tuple-Level	Co-occurrence model	Co-occurrence statistics for a cell's value	#attributes -1
	Tuple representation	FasText-based embedding of the union of tokens after tokenizing each attribute value	1
Dataset-Level	Constraint violations	Number of violations per denial constraint	#constraints
	Neighborhood representation	Distance to top-1 similar word using a FastText tuple embedding over the non-tokenized attribute values	1

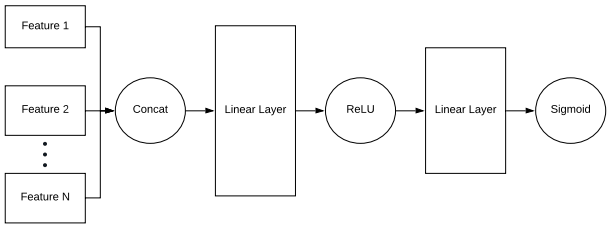

Figure 7: The architecture of our representation learning model following a wide and deep architecture.

be some similar cell in the dataset with the correct value; hence, the distance to it will be low. For this, we simply take the minimum distance to another embedding in our corpus, and this distance is fed to the joint representation.

Forma Models (3-Grams): We follow a similar approach to that of Huang and He [30]. This work introduces custom language models to do outlier detection. We follow a simplified variation of this approach and use two fixed length language models. They correspond to the 3-Gram models shown in Table 7. To build these representation models, we build a distribution of 3-Grams present in each column, this is done using the empirical distribution of the data and Laplace smoothing. For 3-Gram, the distribution is based on all possible ASCII 3-Grams. The difference in the symbol based variation of 3-Gram is that the distribution is based off the alphabet $\{Charcater, Number, Symbol\}$. The value returned for each model is the least frequency of all 3-grams present in the cell value.

A.2 Effect of Misspecified Constraints

We conduct a series of micro-benchmark experiments to evaluate the robustness of AUG against misspecified denial constraints. First, we evaluate AUG's performance as only a subset of constraints is given as input, and second, we evaluate AUG's performance as constraints become noisy.

A.2.1 Limiting the number of Constraints. We consider Hospital, Adult, and Soccer with the denial constraints used for our experiments in Section 6 and perform the following experiment: For each dataset, we define a vary the number of constraints given as input to AUG by taking only a proportion ρ of the initial constraints. We vary ρ in $\{0.2, 0.4, 0.6, 0.8, 1.0\}$ where 0.2 indicates that a random subset of 20% of the constraints is used while 1.0 indicates that all constraints are used. For each configuration for ρ we obtain 21 samples of the constraints and evaluate AUG for these random subsets. We report the median F_1, precision, and recall in Table 8. As shown, AUGs performance gradually decreases as the number of denial constraints is reduced and converges to the performance reported in the study in Section 6.3 when no constraints are used in AUG. The results in Table 8 also show that AUG is robust to small variations in the number of constraints provided as input. We see that when $\rho > 0.4$ the F_1 score of AUG does not reduce more than two points.

Table 8: Median performance of AUG over 21 runs as we randomly limit the input constraints to $\rho \times$ |initial constraints|.

Dataset	M	$\rho = 0.2$	$\rho = 0.4$	$\rho = 0.6$	$\rho = 0.8$	$\rho = 1$
Hospital	P	0.857	0.829	0.927	0.925	0.936
	R	0.848	0.877	0.857	0.896	0.901
	F_1	0.852	0.852	0.891	0.910	0.918
Adult	P	0.860	0.890	0.897	0.917	0.934
	R	0.994	0.992	0.999	0.999	0.999
	F_1	0.922	0.938	0.945	0.956	0.965
Soccer	P	0.836	0.855	0.864	0.860	0.863
	R	0.868	0.879	0.872	0.887	0.894
	F_1	0.852	0.867	0.868	0.873	0.878

A.2.2 Noisy Denial Constraints. We now turn our attention to noisy constraints. We use the following definition of noisy constraints:

Definition A.1. The denial constraint dc is α-noisy on the dataset D if it satisfies α percent of all tuple pairs in D.

We want to see the effect of noisy denial constraints on the performance of AUG. We use the following strategy to identify noisy denial constraints for each dataset: We use the denial constraint discovery method of Chu et al. [11] and group the discovered constraints in four ranges with respect to the

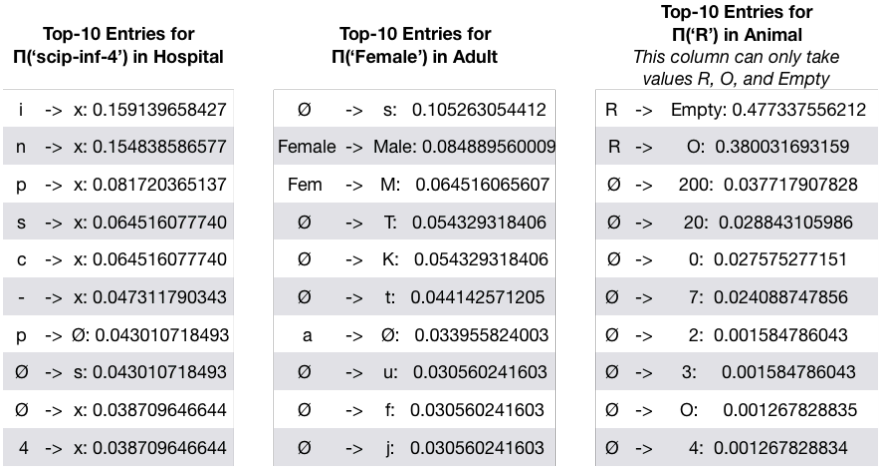

Figure 8: Examples of learned augmentation policies for clean entries in Hospital and Adult.

Table 9: Median performance of AUG over 21 runs with noisy constraints that correspond to different noise levels α.

Dataset	M	$\alpha \in (0.55, 0.65]$	$(0.65, 0.75]$	$(0.75, 0.85]$	$(0.85, 0.95]$
Hospital	P	0.859	0.876	0.912	0.925
	R	0.822	0.869	0.899	0.914
	F_1	0.840	0.873	0.906	0.920
Adult	P	0.911	0.949	0.961	0.984
	R	0.875	0.930	0.952	0.961
	F_1	0.893	0.939	0.956	0.972
Soccer	P	0.821	0.849	0.867	0.863
	R	0.864	0.862	0.880	0.891
	F_1	0.842	0.855	0.873	0.877

noise level α. Constraints with $\alpha \in (0.55, 0.65]$, constraints with $\alpha \in (0.65, 0.75]$, constraints with $\alpha \in (0.75, 0.85]$, and constraints with $\alpha \in (0.85, 0.95]$. For each range, we obtain 21 constraint-set samples, such that each sampled constraint set has the same cardinality as the original clean constraints associated with each of the Hospital, Adult, and Soccer datasets. We report the median performance of AUG in Table 9. As shown, the impact of noisy denial constraints on AUG's performance is not significant. The reason is that during training AUG can identify that the representation associated with denial constraints corresponds to a noisy feature and thus reduce its weight in the final classifier.

A.3 Learned Augmentation Policies

We provide examples of learned policies for clean entries in Hospital, Adult, and Animal. For Hospital and Adult, we know how errors were introduced, and hence, can evaluate the performance of our methods for learning augmentation policies. Errors in Hospital correspond to typos introduced artificially by swapping a character in the clean cell values with the character 'x'. On the other hand, errors in the gender attribute of Adult are introduced either by swapping the two gender values 'Female' and 'Male' or by introducing typos via injection of characters. For Animal, we do not know how errors are introduced. However, we focus on an attribute that can only take values in {R, O, Empty} to evaluate the performance of our methods.

Figure 8 depicts the top-10 entries in the conditional distribution corresponding to entry 'scip-inf-4' for Hospital and entry 'Female' for Adult. As shown, for Hospital, almost all transformations learned by our method correspond to either swapping a character a character with the character 'x' or injecting 'x' in the original string. The performance of our approach is similar for Adult. We observe that a mix of value swaps, e.g., 'Female' $\longmapsto$ 'Male', and character injection transformations are learned. Finally, for Animal, we see that most of the mass of the conditional distribution (almost 86%) is concentrated in the value swap transformations 'R' $\longmapsto$ 'Empty' and 'R' $\longmapsto$ 'O' while all other transformations have negligible probabilities. These results demonstrate that our methods can effectively learn how errors are introduced and distributed in noisy relational datasets.