ED2: A Case for Active Learning in Error Detection

Felix Neutatz
felix.neutatz@dfki.de
DFKI GmbH

Mohammad Mahdavi
mahdavilahijani@tu-berlin.de
TU Berlin

Ziawasch Abedjan
abedjan@tu-berlin.de
TU Berlin / DFKI GmbH

ABSTRACT

State-of-the-art approaches formulate error detection as a semi-supervised classification problem. Recent research suggests that active learning is insufficiently effective for error detection and proposes the usage of neural networks and data augmentation to reduce the number of these user-provided labels. However, we can show that using the appropriate active learning strategy, it is possible to outperform the more complex models that rely on data augmentation. To this end, we propose a multi-classifier approach with two-stage sampling for active learning. This intuitive and neat sampling method chooses the most promising cells across rows and columns for labeling. On three datasets, ED2 achieves state-of-the-art detection accuracy while for large datasets, the required number of user labels is lower by one order of magnitude compared to the state of the art.

1 INTRODUCTION

Data cleaning is an essential process in maintaining high data quality and reliable analytics. A core step in data cleaning is identifying erroneous values [1] and repairing them [14]. In this paper, we consider the problem of error detection. Traditional approaches to detecting errors were either rule-based or quantitative [1].

Recent research proposes to consider error detection as a classification task with the goal to predict whether a cell is erroneous or not [8, 17]. These approaches assume that the required clean/dirty labels are given by the user or drawn uniformly from the dataset. In both scenarios the required amount of labels for a promising classification result is large. Therefore, Heidari et al. suggest using data augmentation to reduce the labeling effort [8]. Given a set of initial labels, data augmentation generates more training data for erroneous examples to better fit the dataset. However, as they report in their paper, their approach still needs an initial set of user-provided labels, in the orders of $5\% - 10\%$ of the whole dataset. For a dataset, such as *Soccer* [13] with $200,000$ rows and 10 columns, the user would need to label around $200,000$ data cells (10% of the dataset), which is huge. Heidari et al. claim that active learning based on uncertainty sampling fails in case of extreme class imbalance [8]. We were working on exactly this problem in parallel and had a different experience. Therefore, we want to make a case for active learning and show that by leveraging an appropriate active learning strategy, it is possible to achieve better accuracy across all openly available datasets that Heidari et al. used [8]. In short, our sampling approach starts with a small seed of uniformly sampled data cells, i.e., fewer than < 100 data cells. Then, we ask the user to label those cells that the current machine learning model is not certain about. This way, the error detection classifier can converge faster with much fewer initial user labels, as we will show in the experiments. Our approach can be used to bootstrap the initially required user labels for state-of-the-art error detection approaches, such as HoloDetect [8].

In general, active learning-based sampling approaches need to address two questions: (1) Considering that errors in different columns are differently hard to detect, which columns should be selected for user labeling? (2) Considering the class imbalance ratio between the dirty and clean values inside any selected column, which data cells should be selected for user labeling? We propose to use a two-stage sampling approach with one classifier per column, requiring a separate training set per column [16]. Each classifier can use features from the entire dataset for classifying the values of its corresponding column. In particular, we discuss the following contributions:

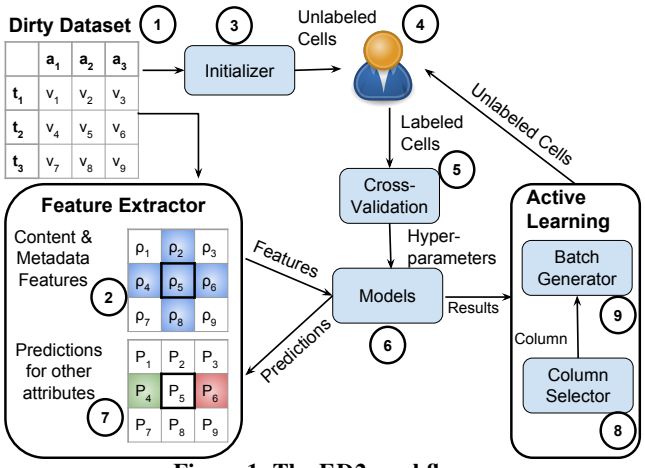

Figure 1: The ED2 workflow.

- To reduce the user labeling effort, we present a new two-stage active learning policy to progressively choose the appropriate cells for labeling. This policy advances standard active learning by not only finding the right cells within one column (first stage), but also across all columns (second stage).
- We conduct several experiments on all openly available datasets provided by Heidari et al. [8] showing that ED2 surpasses state-of-the-art F_1-score while requiring fewer user labels. The required number of user labels is lower by an order of magnitude for large datasets.

We provide more microbenchmarks [11], the implementation of ED2, and results of other baselines in our repository[1].

2 PROBLEM STATEMENT

We address the problem of error detection in relational datasets and use the error definition of previous literature [1, 14]: Given a dataset $D = \{t_1, t_2, ..., t_N\}$ with tuples t_i, its schema $A = \{a_1, a_2, ..., a_M\}$ with attributes a_j, $D[i, j]$ represents the cell value of the attribute a_j in the tuple t_i. Now, given D_C as the cleaned version of D, we define an error as any cell value $D[i, j]$ that deviates from its ground-truth value $D_C[i, j]$. In particular, we want to develop a sampling strategy that chooses the most promising training set to maximizes the F_1-score defined as $F_1 = 2 \times (P \times R)/(P + R)$, where the precision (P)

[1] https://github.com/BigDaMa/ExampleDrivenErrorDetection

is the fraction of correctly detected errors and the recall (R) is the fraction of the actual errors that are discovered.

3 TWO-STAGE ACTIVE LEARNING

We present the workflow of ED2 in Figure 1. ED2 takes a dirty dataset D as input ❶. The *Feature Extractor* generates features ρ that, similar to HoloDetect [8], cover information on the attribute, tuple (across attributes), and dataset level for each data cell of the dataset ❷. The details of the feature representation are described in our technical report [11]. Then, the user receives an initial sample of cells from all columns, which should be labeled as erroneous or correct ❸. Leveraging the user-provided labels ❹, ED2 trains one classifier for each column. ED2 leverages the user-provided labels to automatically optimize the hyperparameters of the classification models using grid search and cross-validation ❺. Afterward, ED2 applies this model to all data cells of the corresponding column and estimates the probability of a cell to be erroneous ❻. In step ❼, ED2 leverages the predictions P to augment the feature vector and enable knowledge sharing across models. For instance, P_6 denotes the prediction that v_6 is erroneous (red). Once the models for all columns are initialized ❻, the actual active learning process starts.

Our two-stage active learning policy is implemented via the *Column Selector* component and the *Batch Generator* component. As we train one classifier per column, the *Column Selector* has to choose the column that should be labeled next ❽. In Section 3.1, we describe how the *Column Selector* leverages the results of the models to make this decision. Then, the *Batch Generator* selects the most promising cells for the given column ❾ as detailed in Section 3.2. For each data cell in the batch, its corresponding tuple is presented to the user for manual verification ❹. The new labels are then added to the training set of the corresponding column, model hyperparameters are optimized ❺, and the classifier is retrained on the new data ❻. The process continues and repeats the aforementioned steps from ❹ to ❾ in a loop as long as the user is willing to provide additional labels. Then, the system applies the latest classification models for each column, marks the errors, and returns the result to the user.

3.1 Column Selection

As we train one classifier per column, we need a strategy for the *Column Selector* to decide the next column for user labeling. A naive approach for column selection is the random column selection strategy (RA). This strategy randomly chooses one column and lets the user label one batch of cells for this selected column. A more orderly approach is the selection in a round-robin fashion (RR). So, we assign the batches to the user for all columns in equal portions and in a circular order.

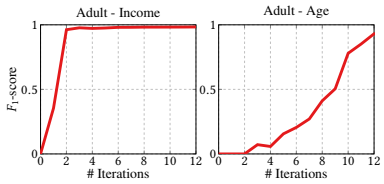

Figure 3: Example of different model convergence behavior.

Both of the presented naive strategies do not consider that, for some columns, the model might detect errors easily and converge quickly, whereas, for other columns, the model might converge slower. Slower convergence also means that the user needs to label more cells to achieve the same performance. Figure 3 illustrates one example of significant differences between the convergence behavior of two columns. In this example, the optimal column selection strategy would choose to label the column *Income* only for two iterations because after that the classifier is already relatively accurate in classifying the *Income* values. Then, the strategy would ask the user for labels for the column *Age*. We studied three more strategies to choose the next best column.

Min Certainty (MC). The classification model returns a probability score for each prediction, i.e., the certainty. High certainty correlates with model convergence [18]. Thus, we calculate the average certainty for all cells inside a column and choose the column with the lowest average certainty.

Max Error (ME). In each iteration, we apply cross-validation on the current labeled training set for the corresponding column. The cross-validation scores are an estimate of the overall performance on the whole column data and thereby also correlate with convergence [18]. We calculate the average of all cross-validation F_1-scores per column and choose the column with the minimum average F_1-score.

Max Prediction Change (MPC). Prediction change is the fraction of predictions that changed after the previous iteration. Prediction change negatively correlates with active learning convergence [3]. We choose the column with the maximum prediction change.

Our experiments show that apart from the naive RA approach all other approaches are similarly effective for the convergence of the models, as long as no column starves for selection.

3.2 Distinct Batch Sampling

Once the next column has been selected by the Column Selector component, the next step is to select actual values within a column. For the data cell values within a column, we apply the query-by-committee algorithm [15] because it is a good fit for XGBoost [4] that has the fastest convergence in comparison to other models, such as a support vector machine or naive Bayes, as preliminary experiments showed [11]. The combination of the query-by-committee algorithm and XGBoost also performs well in case of class imbalance. Other advanced active learning algorithms that are designed for the class imbalance problem can be applied as well [2]. Furthermore, we apply batch active learning [15] to reduce the runtime. In order to keep the sampled batch diverse, we choose cells with distinct values if possible. To train a classifier, we need positive and negative examples for erroneous cells. As a minimum, we should start active learning with two erroneous and two correct cells per column. Instead of browsing through the whole dataset and identifying these cells to start up the classification, we run an outlier detection method to pre-filter the possible cells. Here, we employ a frequency-based ranker that ranks the values of each column by their frequency. Hence, the initial set of cells contain rare and frequent values as potential erroneous and clean training examples.

4 EXPERIMENTS

We measure the effectiveness of the several error detection methods using precision (P), recall (R), and F_1-score (F_1). All experiments

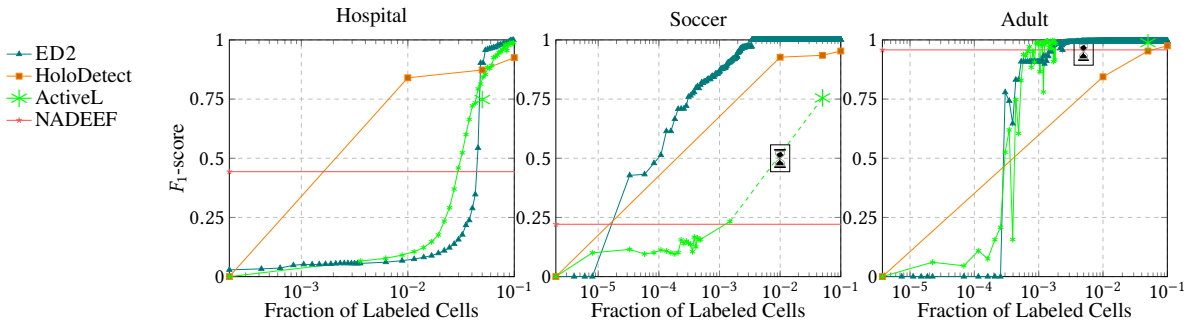

Figure 2: F_1-score and labeling effort of different error detection methods for different datasets. The hourglass marks a time out.

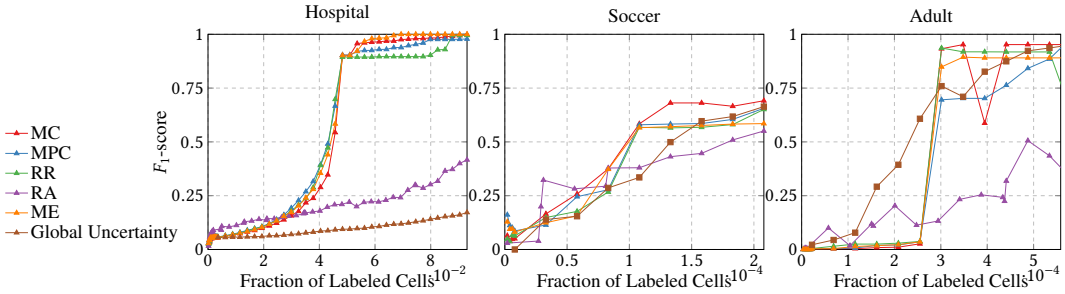

Figure 4: Comparison of different column selection strategies.

were executed on a machine with 14 2.60GHz Intel Xeon E5-2690 CPUs. Our implementation uses 4 cores at maximum.

4.1 Experimental Setup

Baselines. We evaluate ED2 against the following competing error detection approaches. **HoloDetect [8]** is an error detection framework that leverages data augmentation. Note that in contrast to ED2, the user does not only need to label cells as erroneous or correct but additionally has to provide the correct value for erroneous cells. **ActiveL [8]** is another version of HoloDetect that uses vanilla active learning based on uncertainty sampling with a neural network [15]. **NADEEF** [6] is a rule violation detection system based on user-provided rules. We leverage all useful constraints that were detected by Metanome [12] and provide them on our repository[1].

Our Approach. The default column selection strategy of ED2 is *MC*. We choose the same active learning batch size of $k = 50$ as reported by HoloDetect [8]. We choose XGBoost [4] as classifier because of its robustness against irrelevant features [7, 11]. Since ED2 is not deterministic, we apply it 10 times and report the average.

Table 1: Experimental datasets.

Dataset	Columns	Rows	Errors
Hospital	19	1,000	2.65%
Soccer	10	200,000	1.56%
Adult	11	97,684	0.10%

Datasets. We conducted our experiments on all three openly available datasets that were used to evaluate HoloDetect [8]. Table 1 lists these datasets along with their size and the corresponding fraction of erroneous cells divided by all cells in the dataset. **Hospital** is a commonly used dataset to benchmark data cleaning algorithms [6, 8, 14]. Errors are introduced synthetically by randomly replacing characters by the letter 'x'. The **Soccer** and **Adult** datasets and their corresponding ground truth were provided by Rammerlaere et al. [13]. Both datasets contain typographical errors and value swaps.

Table 2: Best F_1-score results of each method in Figure 2 and their corresponding precision (P) and recall (R).

	Hospital			Soccer			Adult		
	P	R	F_1	P	R	F_1	P	R	F_1
ED2	1.00	1.00	**1.00**	1.00	1.00	**1.00**	1.00	0.99	**0.99**
HoloDetect	0.90	0.99	0.94	0.92	1.00	0.96	0.99	0.99	0.99
ActiveL	0.96	0.61	0.75	0.84	0.68	0.76	0.99	0.98	0.99
NADEEF	1.00	0.28	0.44	1.00	0.12	0.22	1.00	0.92	0.96

4.2 Effectiveness

Figure 2 illustrates the F_1-score of different error detection methods with respect to the number of required labels, if applicable. Note that the x-axis is scaled logarithmically.

For the small dataset Hospital, *HoloDetect* and ED2 require labeling of 5% of the data to reach around 90% F_1-score. However, for large datasets, such as Soccer and Adult, ED2 outperforms *HoloDetect* with one order of magnitude fewer labels, namely 0.2% and 0.4% accordingly. ED2 also reaches the highest F_1-score faster than *ActiveL*. *ActiveL* converges slightly faster in the beginning for the Hospital dataset because all columns have the same errors. This characteristic fits the global uncertainty strategy a bit better. For Soccer and Adult, we stopped *ActiveL* after one day. Therefore, we

Table 3: Runtime (seconds) of the error detection methods.

Method	Hospital	Soccer	Adult
ED2	590	11,469	6115
HoloDetect	749	7685	6332
ActiveL	3836	56,535	128,132
NADEEF	19	26	15

can only report the initial performance and the estimated trend based on the results reported by Heidari et al. [8].

In all our experiments, ED2 achieves state-of-the-art error detection F_1-score with comparably fewer labels. Only on Hospital, *HoloDetect* has an initial edge as the errors in this dataset are all the same and easy to generate. The main reason why ED2 requires less labeling than the competing methods is its two-stage sampling policy. This way, it can direct the user's effort at those columns that require the user's attention the most. For instance, 92% of the errors in the dataset Adult are present in the column *Income* alone. NADEEF performs so well on the Adult dataset because we found two integrity constraints that detect errors in this column perfectly.

Table 3 shows that the runtime of ED2 is comparable to the runtime reported for *HoloDetect*. ED2 does not need to perform data augmentation but instead, it has to train a new model for each active learning iteration. *ActiveL* requires significantly more time than ED2 because training the neural network and predicting for all cells of the table instead of only for one column per iteration is more time-intensive. *NADEEF* requires very low runtime because we assume that the constraints are known.

4.3 Column Selection Strategy

Figure 4 illustrates how the column selection strategies proposed in Section 3.1 perform on the three datasets. As expected, the *Random* strategy performs poorly because it does not consider model convergence. The strategies that do consider model convergence specifically, such as *MC*, *MPC*, and *ME*, slightly outperform the *Round-Robin* strategy. Furthermore, they all reach the optimal F_1-score faster than *ActiveL*. For the Adult dataset, we see that the performance drops for one iteration. The reason is that the *MC* strategy will focus first on the column *Income*, which has the most errors. Then, it will start to improve the recall on the other columns. If the model is not immediately perfect, the overall precision drops more than we gain in recall and therefore we see the F_1-score drop.

5 RELATED WORK

Traditional Error Detection. Error detection has received extensive attention in the information management community, such as NADEEF [6] and KATARA [5]. In contrast to learning-based methods, these methods require the user to provide rules.

Machine Learning-Based Error Detection. Machine learning has been leveraged to address the error detection task as a classification problem [8, 17]. Although these approaches require the user to provide a portion of the dataset as initial labeled and corrected data cells, ED2 progressively asks the user to label the most promising data cells to reduce user involvement. Finally, there are methods that aggregate the results of other error detection methods and thereby achieve very accurate detection [1, 10, 17]. This line of research is orthogonal to our work because our method could be plugged into any of the corresponding frameworks. ActiveClean contains an error detection method that assumes that the user trains an application-specific machine learning model on the dataset in question [9]. Therefore, the user has to provide labels for the corresponding machine learning task for all tuples of the dataset.

6 CONCLUSIONS

We showed that unlike previous installments of active learning for error detection, using a two-stage active learning approach results in state-of-the-art F_1-score while requiring comparably low user effort. For large datasets, ED2 requires one order of magnitude fewer labels than competing learning-based approaches.

ACKNOWLEDGMENTS

This work was supported by the German Federal Ministry of Transport and Digital Infrastructure - project DAYSTREAM (19F2031A).

REFERENCES

[1] Ziawasch Abedjan et al. 2016. Detecting Data Errors: Where are we and what needs to be done? *PVLDB* 9, 12 (2016), 993–1004.
[2] Josh Attenberg and Seyda Ertekin. 2013. Class imbalance and active learning. *H. He, & Y. Ma, Imbalanced Learning: Foundations, Algorithms, and Applications* (2013), 101–149.
[3] Michael Bloodgood and K Vijay-Shanker. 2009. A method for stopping active learning based on stabilizing predictions and the need for user-adjustable stopping. In *CoNLL*. 39–47.
[4] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *SIGKDD*. 785–794.
[5] Xu Chu et al. 2015. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*. 1247–1261.
[6] Michele Dallachiesa et al. 2013. NADEEF: a commodity data cleaning system. In *SIGMOD*. 541–552.
[7] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Ann. Stat.* (2001), 1189–1232.
[8] Alireza Heidari et al. 2019. HoloDetect: Few-Shot Learning for Error Detection. In *SIGMOD*.
[9] Sanjay Krishnan et al. 2016. ActiveClean: Interactive Data Cleaning For Statistical Modeling. *PVLDB* 9, 12 (2016), 948–959.
[10] Mohammad Mahdavi et al. 2019. Raha: A Configuration-Free Error Detection System. In *SIGMOD*.
[11] Felix Neutatz et al. 2019. ED2: Two-stage Active Learning for Error Detection – Technical Report. (2019). arXiv:1908.06309
[12] Thorsten Papenbrock et al. 2015. Data Profiling with Metanome. *PVLDB* 8, 12 (2015), 1860–1863.
[13] Joeri Rammelaere and Floris Geerts. 2018. Explaining repaired data with CFDs. *PVLDB* 11, 11 (2018), 1387–1399.
[14] Theodoros Rekatsinas et al. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* 10, 11 (2017), 1190–1201.
[15] Burr Settles. 2010. Active learning literature survey. *University of Wisconsin, Madison* 52, 55-66 (2010), 11.
[16] Grigorios Tsoumakas and Ioannis Katakis. 2007. Multi-label classification: An overview. *IJDWM* 3, 3 (2007), 1–13.
[17] Larysa Visengeriyeva and Ziawasch Abedjan. 2018. Metadata-driven error detection. In *SSDBM*. 1.
[18] Jingbo Zhu and Eduard H Hovy. 2007. Active Learning for Word Sense Disambiguation with Methods for Addressing the Class Imbalance Problem. In *EMNLP-CoNLL*, Vol. 7. 783–790.