Detecting Errors in Databases with Bidirectional Recurrent Neural Networks

Severin Holzer
severin.holzer@zuerich.ch
Statistics Office, City of Zurich
Zurich, Switzerland

Kurt Stockinger
kurt.stockinger@zhaw.ch
Zurich University of Applied Sciences
Winterthur, Switzerland

ABSTRACT

In this paper we introduce an architecture based on *bidirectional recurrent neural networks* to detect errors in databases. The experimental results with 6 different datasets demonstrate that our approach outperforms state-of-the-art error detection systems when considering the average of the F1-scores over all datasets. Moreover, our approach achieves a lower standard deviation than existing work, which shows that our system is more robust. Finally, our approach does *not require additional data augmentation techniques* to achieve high F1-scores.

1 INTRODUCTION

The amount of data is growing exponentially since it is collected by many different stakeholders such as people, companies and machines at many different locations across the globe – but also with varying degrees of quality. When data scientists want to analyze data, they often have to clean it first, which typically takes the most time of the whole data science pipeline [3]. Hence, finding ways of cleaning datasets (semi-)automatically with minimal user involvement is an important aspect for many companies and data science projects. However, due to the heterogeneity of different datasets and the different ways of how data is collected and stored, data cleaning in practice is a hard problem.

The cleaning part can be split into two steps, first to *detect errors* and second to *repair errors*, i.e. to find the ground truth. Errors are values which deviate in the dirty dataset from the clean dataset (see highlighted values in Table 1) and can be categorized as *missing value* ('NaN'), *value/syntactic error* ('80,000', 'Romr', '12', 'BER', '850'), *integrity constraint violation* ('75000') and *data duplication.*

Table 1: Overview of a dirty and the corresponding clean dataset (A=Age, Sal=Salary).

A	Sal	ZIP	City
21	**80,000**	8000	**NaN**
45	98000	00100	**Romr**
30	92000	75000	Paris
12	99000	**BER**	Berlin
26	**850**	**75000**	Vienna

A	Sal	ZIP	City
21	80000	8000	Zurich
45	98000	00100	Rome
30	92000	75000	Paris
42	99000	10115	Berlin
26	85000	1010	Vienna

The focus of this work is on error detection. While related work have used different error detection strategies [1, 2, 4, 6–9, 11], there is ample room for improvement. In our approach we use *bidirectional recurrent neural networks* [12] (RNN), which have received little attention for solving this kind of challenge. Our approach generates a model which tries to find the best parameter settings during learning the content of the data. For training the system, we need labelled data. To reduce the effort for the user to provide labels for the training data, we use only 20 tuples – similar to the state-of-the-art approaches. For selecting the data, we developed a novel algorithm to choose a diverse trainset, which gives our system the best impact and thus the most information content for learning error patterns in the data.

This paper has the following **contributions:**

- We introduce our end-to-end system architecture for error detection based on *bidirectional two-stacked RNNs.*
- The experimental results demonstrate that our approach shows a higher average F1-score and lower standard deviation than state-of-the-art error detection systems on 6 benchmark datasets, which demonstrates that our system is more robust than existing work.

2 SYSTEM ARCHITECTURE

2.1 System in Action

A typical data science pipeline that applies our error detection algorithm looks as follows: The user gives our system a dataset and chooses the number of tuples for training. The system is responsible for the data preparation step and uses our novel label sampling algorithm *DiverSet* to find the tuples with the most information content for the neural network architecture to learn. Afterwards, the user has to label the chosen tuples with either '0' (correct) or '1' (wrong) to produce the trainset. Finally, the system uses the trainset to learn which data points of the tuples are correct or wrong. The involvement of the user is only in the phase of labelling the data.

We now describe the two different recurrent neural network (RNN) architectures that we use for error detection.

2.2 Two-Stacked Bidirectional RNN (TSB-RNN)

As the first model we use a *two-stacked bidirectional RNN*, which has 64 units and as activation function the hyperbolic tangent (see top left part of Figure 1). The orange parts refer to the stacking level $a1$ and the yellow parts to the stacking level $a2$. Moreover, our architecture uses a *bidirectional* RNN, which means that in addition to the forward layer (highlighted in light blue), we also have a backward layer (highlighted in dark blue).

As input the model receives the datasets X_train and X_test, where every character is encoded as a sequence of numbers. For instance, in Figure 1, the input is the character string 'e3' represented as a sequence of numbers (7,9,0,0). First, in the embedding layer the system transforms the sequence of numbers into an embedding vector. The output of our model is a concatenation of the output from the forward path (dim 64) and the backward path (dim 64). In addition, we use a fully connected layer with dimension 32 and ReLu for activation. At the end there is a batch normalization [5] to standardize the input to the softmax. It reduce the generalization error and is a regularization technique.

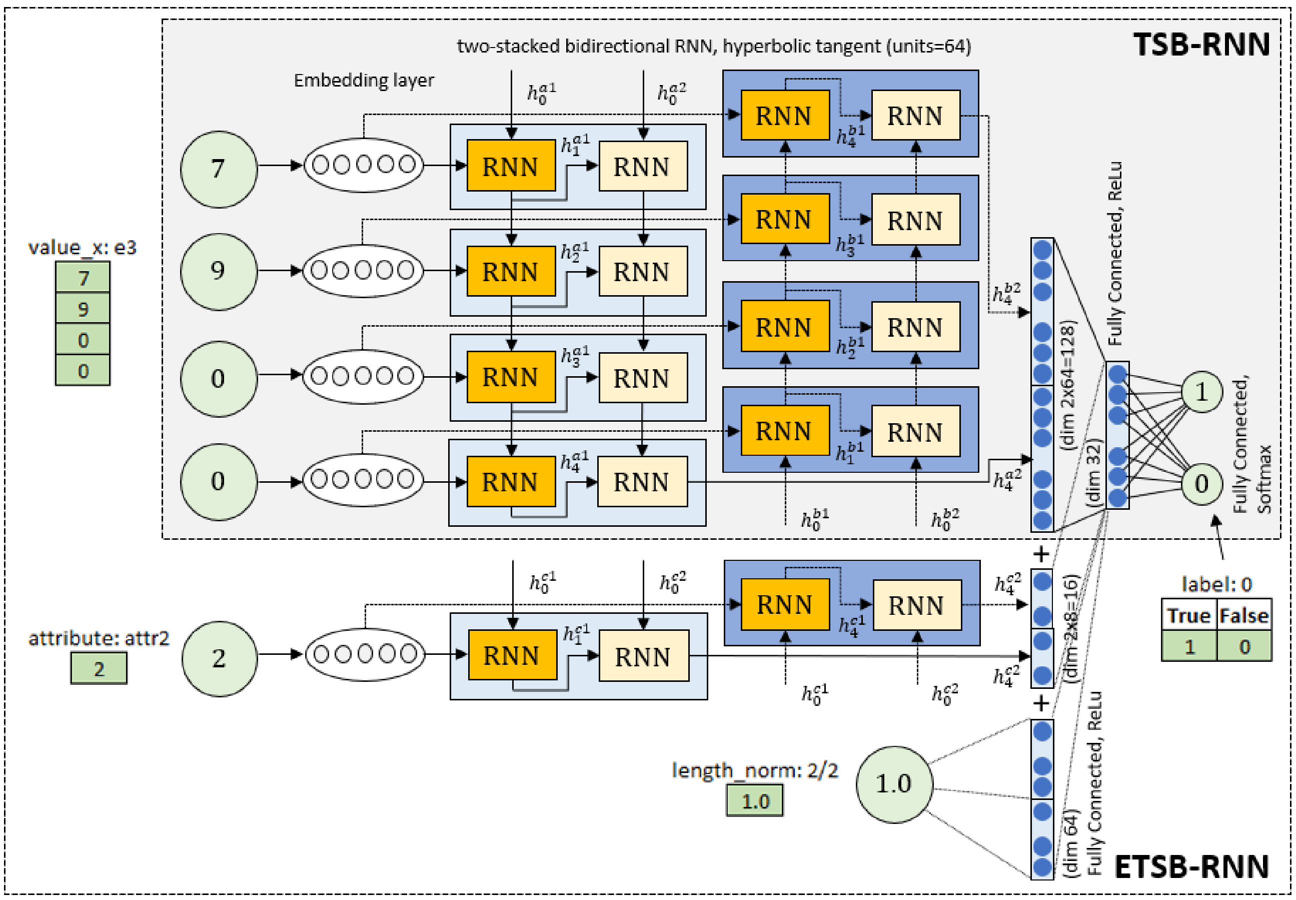

Figure 1: Overview of our bidirectional RNN architecture for error detection. Top part: TSB-RNN = Two-Stacked Bidirectional RNN. Bottom part: ETSB-RNN = Enriched Two-Stacked Bidirectional RNN.

Finally, the model has to decide if the sequence of characters has the value '0' (correct) or '1' (wrong). For this last step the model uses a fully connected layer with a softmax.

2.3 Enriched Two-Stacked Bidirectional RNN (ETSB-RNN)

Our second architecture ETSB-RNN is an enriched version of TSB-RNN. In particular, we additionally provide the *attribute information* X_train_attribute and X_test_attribute as a second input. The attribute information tells the neural network that, for instance, the encoding of the character sequence 'e3' refers to attribute with the name attr2 (see in the left lower part of Figure 1).

As an additional layer the model uses an embedding layer and a two-stacked bidirectional RNN which has 8 units and as activation function the hyperbolic tangent. Moreover, there is a fully connected layer with dimension 64 and the activation function ReLu. We concatenate the output of these two new layers with the output of the previous two-stacked bidirectional RNNs. Afterwards we again use the two fully connected layers and the batch normalization described in TSB-RNN.

2.4 Optimal Training Data Selection

Finally, we introduce a novel label sampling Algorithm 1, whose goal is to select a diverse trainset. The intuition is to select tuples with values that have not been seen previously and thus increase the information content of our trainset. If we selected tuples randomly, we could get observations which have the same values in the attributes and the system would have less information to learn from.

In case two candidate tuples contain the same number of unseen attribute values, the tuple with the highest number of empty attribute values is chosen. Our hypothesis is that empty values give us more information for the system to learn – because if there are empty values in other attributes, we can learn if they should be empty or not. In case all candidate tuples have the same number of unseen attributes and empty attribute values, the tuples are chosen randomly. After the selection the user has to provide a label if the attributes in the tuples are correct or wrong.

3 EXPERIMENTS

In this section we evaluate our two different architectures and measure how well they can detect errors in 6 commonly-used benchmark datasets that were also used by the state-of-the-art error detection systems, e.g. from Raha [8] and Rotom [10]. To enable the reproducibility of our experiments, we share our code on Github[1].

[1]https://github.com/holzesev/E_TSB-RNN

Algorithm 1 DiverSet

```
Input: $n\_obs$ -> number of tuples (IDs) for training
$ds$ -> dataset with all data
Output: $ID\_train$ -> IDs of tuples which are used for the trainset (size: $n\_obs$)
1: function DiverSet($n\_obs, ds$)
2:     $ds\_rest \leftarrow ds$
3:     $ID\_train \leftarrow []$
4:     for i=1 to n_obs do
5:         $\#unseenAttr \leftarrow \mathbf{count}(ds\_rest).\mathbf{groupby}('ID')$
6:         $\#empty \leftarrow \mathbf{sum}(ds\_rest['empty']).\mathbf{groupby}('ID')$
7:         // candidateID contains ID, #unseen and #empty attributes
8:         $candidateID \leftarrow (ID, \#unseenAttr, \#empty)$
9:         // Find candidate with highest number of unseen attributes
10:        $candidateID1 \leftarrow candidateID.\mathbf{max}(\#unseenAttr)$
11:        // Find candidate with highest number of empty attributes
12:        $candidateID2 \leftarrow candidateID1.\mathbf{max}(\#empty)$
13:        // Draw random sample in case of multiple candidates
14:        $sampled\_ID \leftarrow \mathbf{random\_sample}(1, candidateID2)$
15:        $ID\_train \leftarrow ID\_train \cup sampled\_ID$
16:        // Store attributes with seen values
17:
   $seenAttr \leftarrow \mathbf{unique}(ds['concat'].where(ID = ID\_train))$
18:        // Remove attributes with seen values
19:        $ds\_rest \leftarrow ds[ds['concat'] \neq seenAttr]$
20:    end for
21:    return $ID\_train$
22: end function
```

We validated the models 10 times and used Algorithm 1 (DiverSet) for choosing 20 tuples for training. For instance, for the dataset Beers we got a trainset of size 220, i.e. 20 tuples x 11 attributes, and a testset of size 26,290, i.e. 2,390 tuples x 11 attributes.

The number of epochs was 120. After every epoch we saved the training weights (with a callback) if the computed loss of the trainset was less than in the previous epochs. For the loss-function we used the binary cross-entropy and as the optimizer RMSprop. We used the testset to measure precision, recall and F1-score for the best weights. Moreover, we computed the averages and standard deviations of these measures.

3.1 Comparison

We now compare our experimental results with the state-of-the-art error detection systems Raha [8]) and Rotom [10]) (see Table 2).

Table 2: F1-Scores of the different models (20 labeled tuples). B = Beers, F = Flights, H = Hospital, M = Movies, R= Ryan, T = Tax. Our approaches are TSB-RNN and ETSB-RNN.

Dataset	B	F	H	M	R	T
Raha	**0.99**	**0.81**	0.72	0.86	0.79	0.91
Rotom	**0.99**	n/a	**1.00**	0.68	**0.86**	0.97
Rotom+SSL	**0.99**	n/a	**1.00**	0.54	0.76	**1.00**
TSB-RNN	0.96	0.69	0.97	0.87	0.78	0.85
ETSB-RNN	0.98	0.74	0.97	**0.88**	0.85	0.86

Note that in the original paper, Rotom did not take the dataset Flights for validation, hence the values of Flights are marked as n/a. In general we can observe that Raha performs best for the dataset Flights and is en par with both Rotom versions for the dataset Beers. Rotom performs best for the dataset Ryan and is en par with Rotom+SSL for the datasets Beers and Hospital. Rotom+SSL performs performs best for dataset Tax.

Let us now analyze the results of our approaches. We note that ETSB-RRN, which also uses the metadata information, outperforms the simpler model TSB-RNN on all datasets except Movies. Moreover, ETSB-RNN outperforms Raha on 3 out of 6 datasets. In particular, we got better results for Hospital (+0.25), for Movies (+0.02) and for Rayyan (+0.06). We achieved similar results for Beers (-0.01) and Tax (-0.04), only for Flights (-0.07) we got considerably worse results.

In comparison to Rotom and Rotom+SSL, our model ETSB-RNN got a better result for Movies (+0.20 / +0.34). For Beers (-0.01 / -0.01), Hospital (-0.03 / -0.03) and Rayyan (-0.01 / +0.09) we achieved similar results – only for Tax (-0.11 / -0.13) our approach performs considerably worse than Rotem+SSL.

In Table 3 we show the average (AVG) and standard deviation (S.D.) for the error detection systems for all datasets without Flights (1) and in addition for all datasets including Flights (2). Note that Rotom did not take the dataset Flights for validation.

When comparing the average F1-score (AVG) without Flights (1), our approach ETSB-RNN outperforms Raha and Rotom by (+0.06) and (+0.01), respectively. Also the standard deviation is smaller for our system, which indicates that our system is more robust for different datasets. When considering average F1-score (AVG) with Flights (2), again our approach ETSB-RNN outperforms Raha. Also the standard deviation is slightly smaller.

Table 3: Average F1-score (AVG) and Standard Deviation (S.D.) for the different models of Table 2. Note: Smaller S.D. is better.

Name	Without Flights (1)		With Flights (2)	
	AVG	S.D.	AVG	S.D.
Raha	0.85	0.08	0.85	0.07
Rotom	0.90	0.10	n/a	n/a
Rotom+SSL	0.86	0.17	n/a	n/a
TSB-RNN	0.89	0.06	0.85	0.08
ETSB-RNN	**0.91**	**0.05**	**0.88**	**0.06**

3.2 Learning Analysis

We will now analyze the learning behavior of our best approach ETSB-RNN in more detail. To show the improvements of the accuracy over various epochs of our models, we compared the average train- and test-accuracy (see Figure 2). The results show that the model performs well and does not suffer from overfitting. For all datasets our model achieves almost a perfect result for the train-accuracy. There are also gaps in the curves of the test-accuracy but these are not critical because the model chooses the epochs which do not have gaps and learns how to choose the optimal parameter setting based on the decreasing training loss.

The dataset Flights, which reached the worst F1-score, has a big gap between the train- and test-accuracy and also the confidence interval is large. Therefore, our model does not work well for this dataset. The curve for test-accuracy for Hospital looks almost perfect, also because the confidence interval is very small. The datasets Beers, Movies and Rayyan also show a converging test-accuracy curve and the confidence interval is small. For Tax we got a curious result. After epoch 30 there are no wave movements

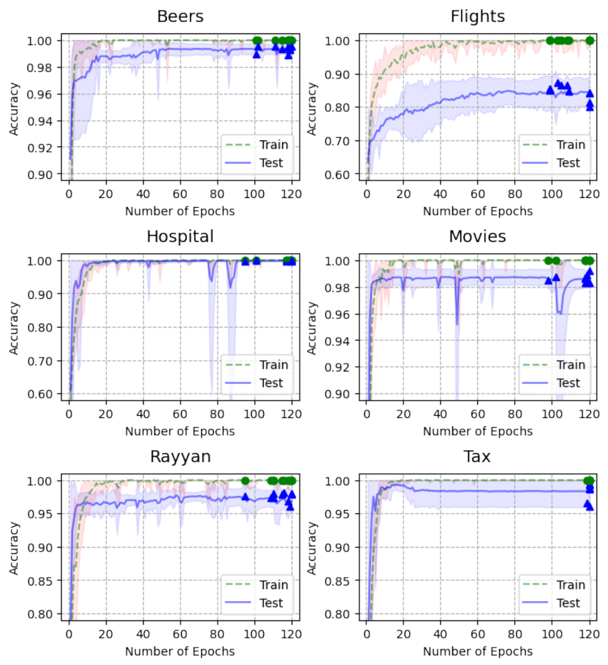

Figure 2: Comparison of average train- and test-accuracy (10-times experiment) and the confidence interval for the different epochs of ETSB-RNN. The green dots shows the epoch and corresponding train-accuracy with the lowest train-loss per experiment. The blue triangles show the corresponding test-accuracy.

in the test-accuracy. In addition, the model chooses the optimal parameter settings from training close to epoch 120, i.e. the green dots and blue triangles.

3.3 Error Analysis

We will now perform a more detailed error analysis of our algorithm ETSB-RNN for each of the datasets.

As described previously, the dataset Flights contains duplicate values due to different information sources for the same flight. ETSB-RNN reached an average F1-score of 0.81. However, it had some problems to detect errors due to varying departure and arrival times of different sources, e.g. flight 'UA-257-JFK-SFO' of source 'orbitz' has a departure time of '2:46 p.m.' while the same flight of source 'flightstats' has a departure time of '2:26 p.m.' The reason is that the model does not share the information about the name of the flight (i.e. the IDs to recognize that flights are the same) and therefore it cannot detect duplicate records. In short, *ETSB-RNN can not identify well functional dependencies between attributes*.

Detecting errors in the Hospital dataset is quite straightforward because the errors are marked with 'x' (e.g. 'hexrt fxilure'). Our approach received an almost perfect result with an average F1-score of 0.97. These types of *errors where single characters are inserted into the data are easy to detect* for our approach.

For the dataset Movies the model ETSB-RNN does not recognize errors in the attribute Creator. The reason is that some parts of the values are missing. (i.e. 'Roger Kumble' instead of 'Choderlos de Laclos, Roger Kumble'). For the attribute Duration the model ETSB-RNN cannot detect the value 'NaN' because in some rows the correct value is 'NaN' and in others the correct value is '96 min'. In short, since our model is based on character embedding as apposed to word embedding, our *approach has problems in identifying missing words.*

The errors of the dataset Rayyan are mostly due to *non-recognized special characters.* For the last dataset Tax we achieved a high standard deviation. Note that between 15 and 20 epochs our system reached a good result. However, after 20 epochs our system seems to overfit and thus does not generalize well.

4 CONCLUSION

We introduced a new way to detect errors in databases using two-stacked bidirectional recurrent neural networks. We trained and tested our models with 6 publicly available datasets and compared the results with the state-of-the-art error detection systems Raha, Rotom and Rotom+SSL. Our approach called Enriched Two-StackedBidirectional (ETSB-RNN) outperforms state-of-the-art systems when considering the average of the F1-scores over all datasets.

REFERENCES

[1] 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.* 1247–1261.
[2] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: a commodity data cleaning system. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data.* 541–552.
[3] Dong Deng, Raul Castro Fernandez, Ziawasch Abedjan, Sibo Wang, Michael Stonebraker, Ahmed K Elmagarmid, Ihab F Ilyas, Samuel Madden, Mourad Ouzzani, and Nan Tang. 2017. The Data Civilizer System.. In *Cidr.*
[4] Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. 2019. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 International Conference on Management of Data.* 829–846.
[5] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning.* PMLR, 448–456.
[6] 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.* 3363–3372.
[7] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. 2016. Activeclean: Interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment* 9, 12 (2016), 948–959.
[8] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A configuration-free error detection system. In *Proceedings of the 2019 International Conference on Management of Data.* 865–882.
[9] Zelda Mariet, Rachael Harding, Sam Madden, et al. 2016. Outlier detection in heterogeneous datasets using automatic tuple expansion. *MIT-CSAIL-TR-2016-002* (2016).
[10] Zhengjie Miao, Yuliang Li, and Xiaolan Wang. 2021. Rotom: A Meta-Learned Data Augmentation Framework for Entity Matching, Data Cleaning, Text Classification, and Beyond. In *Proceedings of the 2021 International Conference on Management of Data.* 1303–1316.
[11] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820* (2017).
[12] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing* 45, 11 (1997), 2673–2681.