Evaluation of Deep Learning Techniques for Entity Matching

Paulo Henrique Santos Lima, Douglas Rolins Santana, Wellington Santos Martins and Leonardo Andrade Ribeiro
Instituto de Informática (INF), Universidade Federal de Goiás (UFG), Goiânia, GO, Brazil

Keywords: Data Cleaning and Integration, Deep Learning, Entity Matching, Experiments and Analysis.

Abstract: Application data inevitably has inconsistencies that may cause malfunctioning in daily operations and compromise analytical results. A particular type of inconsistency is the presence of duplicates, e.g., multiple and non-identical representations of the same information. Entity matching (EM) refers to the problem of determining whether two data instances are duplicates. Two deep learning solutions, DeepMatcher and Ditto, have recently achieved state-of-the-art results in EM. However, neither solution considered duplicates with character-level variations, which are pervasive in real-world databases. This paper presents a comparative evaluation between DeepMatcher and Ditto on datasets from a diverse array of domains with such variations and textual patterns that were previously ignored. The results showed that the two solutions experienced a considerable drop in accuracy, while Ditto was more robust than DeepMatcher.

1 INTRODUCTION

A well-known adage in the Database area is "real-world data is dirty" (Hernández and Stolfo, 1998). In other words, data generated and used application programs invariably exhibit inconsistencies, such as outliers, violations of syntactic and semantic patterns, violations of integrity constraints, and *fuzzy duplicates*, i.e., multiple representations of the same entity (Abedjan et al., 2016). Besides causing malfunctioning of daily operations, e.g., billing and inventory management, such inconsistencies may jeopardize data analysis processes. Thus, identifying and correcting data errors are fundamental tasks in data-driven information systems.

In this paper, we focus on identifying fuzzy duplicates (duplicates, for short). This problem is often referred to as *entity matching* (Barlaug and Gulla, 2021). Entity matching (EM) is challenging because duplicates are not exact copies of one another. Figure 1 illustrates different types of duplicates in a relational database. Duplicates arise in a database for various reasons. Examples of such are data entry errors, different naming conventions, and integration of data sources containing overlapping information.

Deep learning (DL) has promoted over the past decade tremendous progress in machine learning (Krizhevsky et al., 2012). DL achieves state-of-the-art results on perceptual tasks (e.g., object detection, image understanding, and speech recognition) and natural language processing tasks. The underlying data of such tasks is characterized by some hidden structure, which DL excels at uncovering. Given a set of labeled examples, DL completely automates feature engineering, thereby precluding manual intervention. Thus, techniques based on DL have been making a great impact on a wide range of fields, including image, speech, and text processing, among many others.

Recently, DL has also been investigated to solve the EM problem (Barlaug and Gulla, 2021). The work in (Mudgal et al., 2018) presented *DeepMatcher*, a solution based on recurrent neural network (RNN) and attention mechanism. DeepMatcher obtained significant accuracy gains on unstructured and dirty data as compared to *Magellan* (Konda et al., 2016), which was the best learning-based solution for EM. Another approach to EM employs pre-trained language models for *transfer learning* (Brunner and Stockinger, 2020; Li et al., 2020). The work in (Li et al., 2020) presented *Ditto*, a solution that adopts pre-trained models based on the Transformer architecture (Vaswani et al., 2017), with an additional layer fine-tuned for the downstream EM task. Ditto outperformed DeepMatcher in all scenarios analyzed, particularly those with less training data available.

DeepMatcher e *Ditto* considered three types of input data: structured, textual, and "dirty". Structured data has a rigid schema, with simple and atomic val-

Lima, P., Santana, D., Martins, W. and Ribeiro, L.
Evaluation of Deep Learning Techniques for Entity Matching.
DOI: 10.5220/0011996200003467
In *Proceedings of the 25th International Conference on Enterprise Information Systems (ICEIS 2023) - Volume 1*, pages 247-254
ISBN: 978-989-758-648-4; ISSN: 2184-4992

	Title	Author	Publisher
t_1	The Hobbit	J. R. R. Tolkien	George Allen & Unwin
t_2	The Hobbit	Tolkien	Allen & Unwin
t_3	The Jobbit	J. R. R. Token	George Alen & Unwiin

(a) Structured data.

	Description
t_1	Gaming Headset HyperX Cloud II Wireless Black/Red
t_2	Headset HyperX Cloud 2 Wireless, NC Microphone, Black/Red
t_3	Gamming Headset HyperX Cloud II Wirless Back/Red

(b) Textual data.

	Product	Company	Price
t_1	Xbox Series X	Microsoft	663.82
t_2	Xbox Series X Microsoft		663.82
t_3	Xbox Serles X Mircosoft		663.82

(c) Dirty data.

Figure 1: Examples of the three types of duplicates considered in this paper.

ues, such as short strings and numbers (Figure 1(a)). Textual data are characterized by long texts such as product descriptions, texts obtained from web pages, and content from social networks (Figure 1(b)). Finally, dirty data also has a rigid schema like structured data but has attributes with missing values or referring to other attributes. This situation is common when structured data is obtained from information extraction processes. For example, a company name may be incorrectly entered into the Product attribute (Figure 1(c)).

However, DeepMatcher and Ditto did not consider data with character-level textual variations. Such variations are pervasive in real databases, being caused by errors during manual data ingestion, such as typos, or automated ingestion, such as failures in digitization processes. These errors can introduce tokens not covered in the training data (i.e., out-of-vocabulary tokens) and degrade, for example, the effectiveness of approaches based on pre-trained models. Note that the previously mentioned dirty data contains only the transposition of values between attributes, but not variations in attribute values. In Figure 1, tuples `t1` and `t2` in the three data types exemplify the instances considered by DeepMatcher and Ditto, whereas tuples `t3` illustrate instances with the presence of typos.

This paper presents an evaluation of DeepMatcher and Ditto on data with character-level textual variations. To this end, we use the datasets considered in these previous works, but with the injection of random textual modifications in different proportions. The objective of this evaluation is to answer the following questions: 1) how robust are DeepMatcher and Ditto in data with the previously mentioned variations?; 2) how do these variations affect the performance of these solutions on each data type? and 3) how these variations comparatively affect DeepMatcher and Ditto?

The rest of this paper is organized as follows. Section 2 presents background material. Section 3 presents and analyzes the experimental results. Related work is discussed in Section 4. Finally, Section 5 presents the conclusions and outlines future work.

2 BACKGROUND

In this section, we first present the formal definition of the EM problem before covering the details of DeepMatcher and Ditto.

2.1 Problem Definition

We adopt the formalization of the EM problem in (Mudgal et al., 2018). Let D and D' be two data sources with the same schema with attributes $A_1,...,A_N$. In both sources, each tuple represents a real-world entity, which may be a physical, abstract, or conceptual object. The objective of the EM process is to find the largest binary relation $M \in D \times D'$, where each pair $(e_1,e_2) \in M, e_1 \in D, e_2 \in D'$, represents the same entity; we have $D = D'$, if the goal is to find duplicates in a single data source. Further, we assume the availability of a training dataset T of tuples $\{(e_1^i, e_2^i, r)\}_{i=1}^{|T|}$, where $\{(e_1^i, e_2^i)\}_{i=1}^{|T|} \subseteq D \times D'$ and r is a label with values in {"*match*", "*no-match*"}. Given T, a DL-based solution aims to build a classifier that can correctly distinguish pairs of tuples in $D \times D'$ between "match" and "no-match".

2.2 DeepMatcher

DeepMatcher is based on the architectural template illustrated in Figure 2, which enables a rich design space. The input data consists of pairs of tuples representing possible duplicates. In the tokenization step, textual values of each attribute of the two tuples are segmented into sequences of words (the terms token and word are used interchangeably in this paper). In the following steps, DL can be used in various ways to build different EM solutions.

In the embedding step, sequences of words are represented as sequences of numerical vectors. Possible approaches for this step can be defined along two axes: embedding granularity and training strategy. The granularity of the embedding can be word-based or character-based. In the first case, a lookup table is learned mapping each word to a numerical value. In the second case, a model is trained to produce embed-

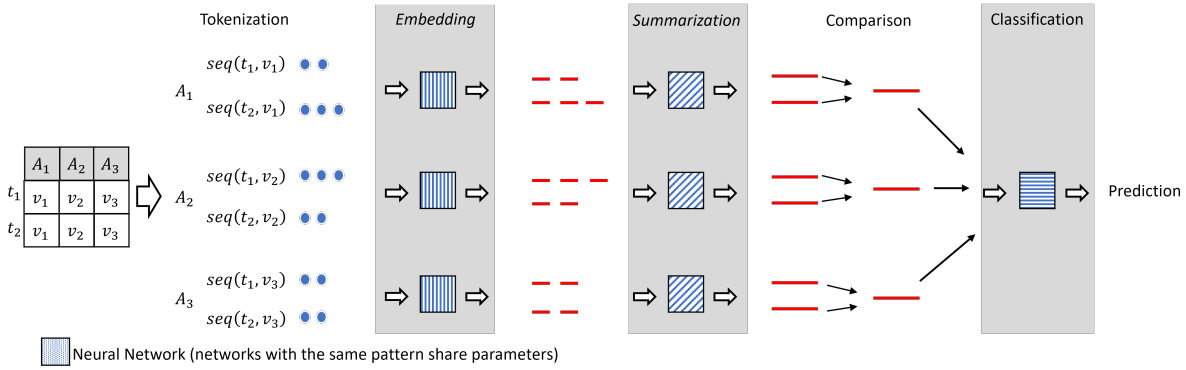

Figure 2: DeepMatcher architectural template.

dings for words that contain characters in its vocabulary. Character-level embeddings are more robust in dealing with out-of-vocabulary words, which can be caused by typing errors, as mentioned earlier. Regarding the training strategy, one can use pre-trained embeddings or train them from scratch. The last option may be preferable in domains that contain tokens with specific semantics, such as product codes.

In the summarization step, the sequences of vectors representing the attributes of each tuple are aggregated into a single vector. To this end, RNNss can be used to capture the order and semantics of token sequences. This approach has limitations, however. First, in general, RNNs have difficulty learning meaningful representations of long sequences of tokens. Second, the two sequences are not considered together in the summarization process; as a result, the underlying similarity between sequences of different sizes may not be captured. A different approach is to use an attention mechanism so that the sequence of vectors from one tuple is used as context in summarizing the sequence of vectors from the other tuple. However, information about the position of input tokens will be lost in the summarization process. This information may be important in some cases, for example, when the most relevant token is the first. Finally, it is possible to adopt a hybrid strategy, combining RNNs and attention mechanisms to obtain the advantages and avoid the problems of each approach at the cost of increasing the complexity of the learning model and, in turn, requiring longer training times.

In the comparison step, the similarity (or distance) of the summarized vectors is assessed using a metric such as the cosine similarity or the Euclidean distance. The result of comparing each attribute is a scalar similarity value; these values will compose the set of characteristics of the neural network used in the classification step. Another possibility is to use operations such as concatenation and the element-wise absolute difference between elements as a comparison function and relegate the task of learning a similarity function to the classifier. Finally, the classifier's output will determine whether the input pair of tuples represent the same entity.

In the empirical evaluation of (Mudgal et al., 2018), DeepMatcher obtained results on structured data similar to simpler learning models, such as random forests and logistic regression, which require much less training time. On the other hand, expressive gains in accuracy were obtained on textual and dirty data. In addition, several instances of the architectural template of Figure 2 were evaluated. The model that obtained the best results used FastText (Bojanowski et al., 2017), a character-level pre-trained embedding model, a hybrid summarization approach combining bidirectional RNN with attention mechanism, learnable similarity function, and a multi-layer perceptron as the classifier. We evaluated this model in our experiments.

2.3 Ditto

The advent of language models based on the Transformer architecture promoted further progress in solutions for EM (Brunner and Stockinger, 2020; Li et al., 2020). These models are pre-trained on large text corpora, such as Wikipedia, in an unsupervised manner. The Transformer architecture completely replaces RNNs with a self-attention mechanism (Vaswani et al., 2017) to generate token embeddings considering all the other tokens of the input sequence. Thus, these embeddings capture semantic and contextual information, including intricate linguistic aspects such as polysemy and synonymy. BERT (Devlin et al., 2019) is the most popular pre-trained language model based on Transformers.

Ditto fine-tunes such pre-trained models for the EM task. To this end, a fully connected layer and an output layer using the softmax function are added. The modified network is initialized with the pre-

Figure 3: Ditto preprocessing pipeline.

trained model parameters and then trained with the data in T until it converges. In addition to this fine-tuning, Ditto uses a method to serialize the input pairs into a sequence of tokens and performs three optimizations in a preprocessing step: insertion of domain knowledge, summarization of long sequences, and augmentation of training data with difficult examples. Figure 3 illustrates these preprocessing steps.

Ditto serializes a tuple t as follows:

$$serialize(t) = [COL]A_1[VAL]v_1...[COL]A_N[VAL]v_N,$$

where `[COL]` and `[VAL]` are special tokens indicating the beginning of attribute names and values, respectively. For example, the result of serializing the tuple t_2 in Figure 1(c) is given by: `[COL]` *Product* `[VAL]` *Xbox Series X Microsoft* `[COL]` *Company* `[VAL]` *NULL* `[COL]` *Price* `[VAL]` *663.82*.

Pairs of tuples are serialized as follows:

$$serialize(t_1,t_2) = [CLS]serialize(t_1)[SEP]serialize(t_2)[SEP],$$

where `[SEP]` is the special token delimiting the representations of t_1 and t_2 and `[CLS]` is the token whose associated embedding will represent the encoding of the tuple pair.

Domain knowledge can be inserted into serialized inputs in two ways: including special tokens identifying snippets with specific semantics, such as product codes and street numbers, and rewriting text spans that refer to the entity (i.e., synonyms) into a single string. Summarization reduces long strings in serialized input to the maximum length allowed by BERT, which is 512 tokens. Training data augmentation is performed via operators that generate new serialized input pairs from existing pairs. These operators apply random modifications to the original pairs, such as token deletion and transposition.

In contrast to DeepMatcher, Ditto does not require the input data to have the same schema. The serialization method even allows applying Ditto to hierarchical data such as XML and JSON using special tokens to represent nested attribute-value pairs. Another difference with DeepMatcher is that in Ditto the cross-attention mechanism between the pair of tuples is not limited to words of the same attribute. These differences can be mitigated by applying the serialization method also to DeepMatcher and associating the result to a single attribute — the evaluation of this strategy is left for future work. Nevertheless, Ditto still has a simpler architecture than DeepMatcher, where the components of embedding, summarization, and comparison (see Figure 2) are replaced by a pre-trained language model.

Finally, Ditto has support for language models other than BERT. The model that showed the best results was RoBERTa (Liu et al., 2019), a variant of BERT with a set of improvements and trained on a larger volume of data. We evaluated the version of Ditto based on RoBERTa in our experiments.

3 EXPERIMENTS

This section presents the results of our experimental study, whose objective was to evaluate and compare the performance of DeepMatcher and Ditto on data with character-level textual variations. We first describe the experimental environment, i.e., datasets, the configuration of the analyzed techniques, hardware and software configuration, and metrics. Then, we present and discuss the results.

3.1 Experimental Setup

We experimented with publicly available datasets, which were also used for evaluating DeepMatcher and Ditto. The datasets are from a variety of domains and have different characteristics, five of which are structured, one textual, and four dirty; the latter were generated from structured datasets by transposing values between attributes, as mentioned in Section 1. In all datasets, each data item is composed of a pair of tuples and a label that classifies this pair as ”match” or ”no-match”. Details about these datasets are presented in Table 1. The size of datasets, the number of pairs classified as duplicates, and the number of attributes are informed in columns 4 to 6, respectively.

For each dataset, we generated eight copies with different percentages of changed tuple pairs; we call these copies *erroneous datasets*. We injected modifications on each changed pair by applying character-level transformations, i.e., character insertion, deletion, and substitution. Table 2 describes the erroneous datasets. For example, in dataset `E1`, 10% of tuple pairs had 1–2 character transformations. Each modified pair was randomly selected from the original dataset, as well as the character and type of each modification. As such, modifications can affect both tuples of each pair or just one of them. For each

Table 1: Description of the original datasets.

Type	Dataset	Domain	Size	# Positives	# Attributes
Structured	Amazon-Google	softwares	11.460	1.167	3
	BeerAdvo-RateBeer	drinks	450	68	4
	DBLP-ACM	citation	12.363	2.220	4
	DBLP-Google	citations	28.707	5.347	4
	Walmart-Amazon	electronics	10.242	962	5
Textual	Abt-Buy	products	9.575	1.028	3
Dirty	DBLP-ACM	citations	12.363	2.220	4
	DBLP-Google	citations	28.707	5.347	4
	iTunes-Amazon	music	539	132	8
	Walmart-Amazon	electronics	10.242	962	5

dataset, two attributes with values considered more informative were chosen to be modified. For example, in DBLP-ACM, the attributes `title` and `author` were modified. Each modified pair of tuples replaces the original pair while maintaining the associated label. Therefore, the E1-8 erroneous datasets maintain statistics of the original datasets shown in Table 1. Finally, we split each dataset into training, validation, and test sets according to the ratio 3:1:1, the same ratio used in the DeepMatcher and Ditto papers.

As already mentioned, we evaluated the models of DeepMatcher and Ditto that obtained the best results in the original papers: for DeepMatcher, we considered the Hybrid model, which employs the more complex summarization strategy; for Ditto, we considered the version that employs all the optimizations and the pre-trained RoBERTa language model. The experiments used implementations of DeepMatcher[1] and Ditto [2] made publicly available by the authors. DeepMatcher was implemented using Torch[3], a framework with support for using GPUs to speed up training. Ditto was implemented using PyTorch[4] and Hugging Face Transformers[5]. Further for Ditto, the maximum size of the input sequence was fixed at 256, the learning rate at *3e-5* in a linearly decreasing manner, and the batch size was set to 32. All experiments were performed through Google Collaboratory[6], which allows the user to run Python through the browser and use GPUs for free. Due to the limited computational resources, training was carried out in 15 epochs. Other parameters were defined according to the settings described in the respective paper.

Accuracy results are reported by the F1 metric, which is the harmonic mean between precision and recall. Given the result of testing a model, let VP be the number of pairs correctly classified as a match, FP the number of pairs incorrectly classified as a match, and FN the number of pairs incorrectly classified as a no-match. Precision P is given by $P = TP/(TP+FP)$ and recall R is given by $R = TP/(TP+FN)$. Therefore, $F1$ is given by $2 \times ((P \times R)/(P+R))$.

[1]https://github.com/anhaidgroup/deepmatcher.

[2]https://github.com/megagonlabs/ditto.

[3]http://torch.ch/.

[4]https://pytorch.org.

[5]https://huggingface.co.

[6]https://colab.research.google.com.

3.2 Results and Discussion

Accuracy results of DeepMatcher and Ditto on structured, textual, and dirty data are shown in Tables 3, 4, and 5, respectively. For each dataset, we report results obtained in its original version, without modifications, and in its erroneous versions, E1–8, with character-level textual variations. Note that we ran experiments on the original version of the datasets from scratch instead of simply reproducing the numbers in the papers of DeepMatcher and Ditto. The results of each erroneous version are accompanied by their difference from the result obtained in the original version. The best result obtained in each dataset is highlighted in red.

We first discuss the results obtained in the original datasets. They followed the same trends observed in (Li et al., 2020), with Ditto showing a clear advantage over DeepMatcher owing to its language understanding capability. In structured data, Ditto is superior in 4 out of 5 datasets; the largest accuracy advantage is 22.27% on the Walmart-Amazon dataset. Ditto also outperforms DeepMatcher on the textual dataset by 19.12%. On the dirty datasets, Ditto's results are close to those of DeepMatcher on the DBLP-ACM and DBLP-Google datasets but still better, while on the iTunes-Amazon dataset, there is an advantage of 25.38%. DeepMatcher and Ditto both perform poorly on the Walmart-Amazon dataset. In particular, we consider F1 score of Ditto for this dataset an outlier because it excessively deviated from the number reported in (Li et al., 2020): 17.21% and 85.69%, respectively. This was the only case we observed such a great divergence between our results and those in the original papers. After rerunning the tests for this dataset with 40 epochs, we obtained a much better

Table 2: Description of the erroneous datasets.

erroneous datasets	E1	E2	E3	E4	E5	E6	E7	E8
% changed pairs	10%	30%	10%	30%	50%	90%	50%	90%
# transformations	1-2	1-2	3-5	3-5	1-2	1-2	3-5	3-5

Table 3: F1 scores on structured datasets.

dataset/technique		Original	E1	E2	E3	E4	E5	E6	E7	E8
Amazon-Google	**DeepMatcher**	68.94	62.78	53.50	54.74	47.20	56.07	40.60	48.16	42.83
	ΔF_1	-	-6.16	-15.44	-14.20	-21.74	-12.87	-28.34	-20.78	-26.11
	Ditto	71.58	74.10	68.80	18.51	63.72	68.79	57.14	56.01	50.39
	ΔF_1	-	2.52	-2.78	-53.07	-7.86	-2.79	-14.44	-15.57	-21.19
Beer	**DeepMatcher**	70.97	72.22	66.67	68.75	62.86	70.97	64.52	64.52	45.71
	ΔF_1	-	1.25	-4.30	-2.22	-8.11	0.00	-6.45	-6.45	-25.26
	Ditto	85.71	90.32	73.33	66.66	68.96	66.66	55.17	28.57	51.61
	ΔF_1	-	4.61	-12.38	-19.05	-16.75	-19.05	-30.54	-57.14	-34.10
DBLP-ACM	**DeepMatcher**	98.76	98.65	98.33	98.77	98.00	98.31	98.43	97.99	97.33
	ΔF_1	-	-0.11	-0.43	0.01	-0.76	-0.45	-0.33	-0.77	-1.43
	Ditto	98.00	98.75	98.10	98.52	97.83	98.53	98.42	98.29	98.29
	ΔF_1	-	0.75	0.10	0.52	-0.17	0.53	0.42	0.29	0.29
DBLP-Google	**DeepMatcher**	94.90	94.85	94.05	93.29	92.56	93.51	91.80	91.28	89.88
	ΔF_1	-	-0.05	-0.85	-1.61	-2.34	-1.39	-3.10	-3.62	-5.02
	Ditto	95.05	95.14	94.89	94.17	94.38	94.56	94.00	94.37	92.25
	ΔF_1	-	0.09	-0.16	-0.88	-0.67	-0.49	-1.05	-0.68	-2.80
Walmart-Amazon	**DeepMatcher**	63.66	62.18	64.15	62.05	61.28	62.32	60.57	59.57	63.16
	ΔF_1	-	-1.48	0.49	-1.61	-2.38	-1.34	-3.09	-4.09	-0.50
	Ditto	85.93	85.05	17.09	84.55	29.08	80.40	81.52	17.21	21.23
	ΔF_1	-	-0.88	-68.84	-1.38	-56.85	-5.53	-4.41	-68.72	-64.70

Table 4: F1 scores on textual datasets.

dataset/technique		Original	E1	E2	E3	E4	E5	E6	E7	E8
Abt-Buy	**DeepMatcher**	70.03	71.26	65.99	65.96	64.66	61.63	56.31	53.20	51.19
	ΔF_1	-	1.23	-4.04	-4.07	-5.37	-8.40	-13.72	-16.83	-18.84
	Ditto	89.15	90.51	82.72	84.93	78.46	88.05	24.74	86.93	74.87
	ΔF_1	-	1.36	-6.43	-4.22	-10.69	-1.10	-64.41	-2.22	-14.28

Table 5: F1 scores on dirty datasets.

dataset/technique		Original	E1	E2	E3	E4	E5	E6	E7	E8
DBLP-ACM	**DeepMatcher**	97.20	95.26	95.53	92.63	94.53	95.51	93.41	89.29	86.57
	ΔF_1	-	-1.94	-1.67	-4.57	-2.67	-1.69	-3.79	-7.91	-10.63
	Ditto	97.62	97.63	97.29	98.30	97.96	97.87	98.08	97.62	96.37
	ΔF_1	-	0.01	-0.33	0.68	0.34	0.25	0.46	0.00	-1.25
DBLP-Google	**DeepMatcher**	92.22	91.89	91.56	91.49	90.39	91.26	90.32	90.38	86.63
	ΔF_1	-	-0.33	-0.66	-0.73	-1.83	-0.96	-1.90	-1.84	-5.59
	Ditto	95.07	94.69	94.66	95.04	94.44	94.69	94.06	93.25	93.22
	ΔF_1	-	-0.38	-0.41	-0.03	-0.63	-0.38	-1.01	-1.82	-1.85
iTunes-Amazon	**DeepMatcher**	70.77	64.52	69.84	63.49	67.74	66.67	67.80	67.65	70.97
	ΔF_1	-	-6.25	-0.93	-7.28	-3.03	-4.1	-2.97	-3.12	0.2
	Ditto	96.15	94.11	83.87	88.13	72.22	86.20	90.56	81.35	90.90
	ΔF_1	-	-2.04	-12.28	-8.02	-23.93	-9.95	-5.59	-14.8	-5.25
Walmart-Amazon	**DeepMatcher**	38.73	53.23	41.29	47.77	33.49	42.39	42.61	33.43	36.16
	ΔF_1	-	14.5	2.56	9.04	-5.24	3.66	3.88	-5.3	-2.57
	Ditto	17.21	82.63	83.33	81.86	78.19	81.01	82.19	40.16	85.26
	ΔF_1	-	65.42	66.12	64.65	60.98	63.8	64.98	22.95	11.95

convergence, and the F1 score increased to 84.53%.

We now discuss the results obtained on the erroneous datasets. On the structured datasets, Ditto outperforms DeepMatcher on Amazon-Google and DBLP-Google; results are similar on the other three datasets. On the textual dataset, Ditto outperforms DeepMatcher by a significant margin, except for the outlier result on E6. On the dirty datasets, Ditto is superior to DeepMatcher on all erroneous datasets.

In general, both solutions experienced a decrease in accuracy in almost all cases. The largest drop in F1 scores occurred on Amazon-Google. This dataset is more challenging for the EM task because the distinction between matches and no-matches is more subtle; note that the two solutions also achieved their worst results on the original version of this dataset. On the other hand, the few cases in which accuracy increased or only slightly decreased occurred on DBLP-ACM and DBLP-Google. These datasets can be considered easier for EM because they exhibit a better separation between matches and no-matches. Moreover, DBLP-ACM and DBLP-Google are the largest datasets, thereby providing more training data; note that DeepMatcher and Ditto also obtained their best results on the original version of these datasets. Such results indicate that the effect of the textual modifications at the character level follows the characteristics of the original datasets in terms of the separation between matches and no matches: the impact of these modifications is negligible on easy datasets and significant on difficult ones.

Table 6 shows the training time of each model in the original datasets—note the hh:mm:ss format. DeepMatcher required a shorter training time on the three datasets types: the total training time of DeepMatcher corresponds to 36.69% of Ditto on the structured datasets (Table 6(a)), 24.09% on the textual datasets (Table 6(b)), and 41.57% on the dirty datasets. Note that these values only indicate the training time needed in a shared processing environment such as Google Colab. Of course, an accurate comparison would require dedicated hardware.

4 RELATED WORK

The EM problem has been studied since the late 1950s, starting with the pioneering work in (Newcombe et al., 1959). Since then, various scientific communities, including Databases, Information Retrieval, Natural Language Processing (NLP), Machine Learning, Semantic Web, and Statistics, have addressed many aspects of the problem. EM is often referenced in these communities by a variety of different terms, such as entity resolution, record matching, deduplication, and reference reconciliation, among others. A review of the literature prior to the emergence of DL is presented in (Elmagarmid et al., 2007). Another review, more recent and focusing on DL-based techniques, is presented in (Barlaug and Gulla, 2021). DeepMatcher and Ditto are representative examples among the current set of DL-based solutions. For example, DeepER (Ebraheem et al., 2018) corresponds to a simpler instance of the architectural template of DeepMatcher, whereas the solution in (Brunner and Stockinger, 2020) corresponds to the basic version of Ditto without optimizations.

EM is closely related to other problems in NLP and data integration, often with interchangeable solutions. Some examples include entity linking (Shen et al., 2015), which aims to link mentions of entities in a document to an entity represented in a knowledge base. Entity alignment (Leone et al., 2022) refers to the problem of finding equivalences between entities from two different knowledge bases. Coreference resolution (Clark and Manning, 2016) identifies text segments in documents that refer to the same entity.

EM has an intrinsic quadratic complexity as it requires comparing each entity representation with one another. For this reason, EM is typically preceded by a blocking step, which divides the input data into (possibly overlapping) blocks and considers only entities within the same block for matching. In an approach similar to (Mudgal et al., 2018), the work in (Thirumuruganathan et al., 2021) defines a space of DL solutions for blocking from which a representative set was evaluated.

5 CONCLUSION

This paper presented a comparative evaluation of DeepMatcher and Ditto on data with textual patterns not considered in previous experiments. The results showed that the two solutions experienced a drop in accuracy in most of the analyzed scenarios, and Ditto presented, in general, greater effectiveness and robustness compared to DeepMatcher at the cost of requiring more training time. Furthermore, we observed that the effect of textual modifications on the classification accuracy is dictated by the characteristics of the original datasets in terms of the separation between matches and no-matches: the impact of these modifications is negligible on easy datasets and significant on difficult ones. Future works include studying improvements in the training process to capture the textual modifications considered in this paper and experiments with different data representations.

Table 6: Training time comparison between DeepMatcher and Ditto (hh:mm:ss format).

technique/dataset	Amazon-Google	Beer	DBLP-ACM	DBLP-Google	Walmart-Amazon	Total
DeepMatcher	0:07:08	00:00:28	00:12:12	00:27:08	00:10:30	0:57:26
Ditto	0:19:17	0:02:14	0:41:02	1:08:02	0:25:57	2:36:32

(a) Structured data.

technique/dataset	Abt-Buy
DeepMatcher	00:09:11
Ditto	00:38:07

(b) Textual data.

technique/dataset	DBLP-ACM	DBLP-Google	iTunes-Amazon	Walmart-Amazon	Total
DeepMatcher	0:14:28	0:30:41	0:01:22	0:11:04	0:57:35
Ditto	0:41:36	1:08:34	0:03:53	0:24:29	2:18:32

(c) Dirty data.

ACKNOWLEDGEMENTS

This work was partially supported by CNPq/Brazil.

REFERENCES

Abedjan, Z., Chu, X., Deng, D., Fernandez, R. C., Ilyas, I. F., Ouzzani, M., Papotti, P., Stonebraker, M., and Tang, N. (2016). Detecting Data Errors: Where Are We and What Needs to Be Done? *Proceedings of the VLDB Endowment*, 9(12):993–1004.

Barlaug, N. and Gulla, J. A. (2021). Neural Networks for Entity Matching: A Survey. *ACM Transactions on Knowledge Discovery from Data*, 15(3):52:1–52:37.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Brunner, U. and Stockinger, K. (2020). Entity Matching with Transformer Architectures – A Step Forward in Data Integration. In *Proceedings of the International Conference on Extending Database Technology*, pages 463–473.

Clark, K. and Manning, C. D. (2016). Improving Coreference Resolution by Learning Entity-Level Distributed Representations. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 643–653.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186.

Ebraheem, M., Thirumuruganathan, S., Joty, S. R., Ouzzani, M., and Tang, N. (2018). Distributed Representations of Tuples for Entity Resolution. *Proceedings of the VLDB Endowment*, 11(11):1454–1467.

Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. (2007). Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16.

Hernández, M. A. and Stolfo, S. J. (1998). Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery*, 2(1):9–37.

Konda, P., Das, S., C., P. S. G., Doan, A., Ardalan, A., Ballard, J. R., Li, H., Panahi, F., Zhang, H., Naughton, J. F., Prasad, S., Krishnan, G., Deep, R., and Raghavendra, V. (2016). Magellan: Toward Building Entity Matching Management Systems. *Proceedings of the VLDB Endowment*, 9(12):1197–1208.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 1106–1114.

Leone, M., Huber, S., Arora, A., García-Durán, A., and West, R. (2022). A Critical Re-evaluation of Neural Methods for Entity Alignment. *Proceedings of the VLDB Endowment*, 15(8):1712–1725.

Li, Y., Li, J., Suhara, Y., Doan, A., and Tan, W. (2020). Deep Entity Matching with Pre-Trained Language Models. *Proceedings of the VLDB Endowment*, 14(1):50–60.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*, abs/1907.11692.

Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., Deep, R., Arcaute, E., and Raghavendra, V. (2018). Deep Learning for Entity Matching: A Design Space Exploration. In *Proceedings of the SIGMOD Conference*, pages 19–34. ACM.

Newcombe, H., Kennedy, J., Axford, S., and James, A. (1959). Automatic Linkage of Vital Records. *Science*, 130(3381):954–959.

Shen, W., Wang, J., and Han, J. (2015). Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460.

Thirumuruganathan, S., Li, H., Tang, N., Ouzzani, M., Govind, Y., Paulsen, D., Fung, G., and Doan, A. (2021). Deep Learning for Blocking in Entity Matching: A Design Space Exploration. *Proceedings of the VLDB Endowment*, 14(11):2459–2472.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is All you Need. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 5998–6008.