Interpreting Deep Learning Models for Entity Resolution: An Experience Report Using LIME

Vincenzo Di Cicco
Università Roma Tre
vin.dicicco@stud.uniroma3.it

Donatella Firmani
Università Roma Tre
donatella.firmani@uniroma3.it

Nick Koudas
University of Toronto
koudas@cs.toronto.edu

Paolo Merialdo
Università Roma Tre
paolo.merialdo@uniroma3.it

Divesh Srivastava
AT&T Labs – Research
divesh@research.att.com

ABSTRACT

Entity Resolution (ER) seeks to understand which records refer to the same entity (e.g., matching products sold on multiple websites). The sheer number of ways humans represent and misrepresent information about real-world entities makes ER a challenging problem. Deep Learning (DL) has provided impressive results in the field of natural language processing, thus recent works started exploring DL approaches to the ER problem, with encouraging results. However, we are still far from understanding *why* and *when* these approaches work in the ER setting. We are developing a methodology, `Mojito`, to produce explainable interpretations of the output of DL models for the ER task. Our methodology is based on `LIME`, a popular tool for producing prediction explanations for generic classification tasks. In this paper we report our first experiences in interpreting recent DL models for the ER task. Our results demonstrate the importance of explanations in the DL space, and suggest that, when assessing performance of DL algorithms for ER, accuracy alone may not be sufficient to demonstrate generality and reproducibility in a production environment.

ACM Reference format:
Vincenzo Di Cicco, Donatella Firmani, Nick Koudas, Paolo Merialdo, and Divesh Srivastava. 2019. Interpreting Deep Learning Models for Entity Resolution: An Experience Report Using LIME. In *Proceedings of International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, Amsterdam, Netherlands, July 5, 2019 (aiDM'19),* 4 pages.
https://doi.org/10.1145/3329859.3329878

1 INTRODUCTION

Entity resolution (ER) aims at matching records that refer to the same real-world entity. Although widely studied for the last 50 years [6], ER still represents a challenging data management task. Several database researchers have recently started to investigate the opportunity of applying deep learning techniques (DL) to address this issue [5, 10]. In the DeepMatcher project [10], Mudgal *et al.* have conducted a seminal study aimed at analyzing benefits and limitations of DL to tackle ER. Their study has empirically compared the performance of four DL solutions with a more traditional yet state-of-the-art solution, Magellan [8], on several datasets. The analysis concentrates on the evaluation of the accuracy, in terms of precision (P), i.e. fraction of match predictions that are correct, recall (R), i.e. fraction of correct matches being predicted as matches, and F-measure, defined as $2PR/(P + R)$.

The results of the study show that DL represents a promising direction: on structured data, the F-measure of DL solutions are similar to those obtained by Magellan; on textual and dirty data they outperform Magellan. Based on these results, they advocate further research to improve the DL solutions focused on ER.

To explore the design space of DL solutions for ER, the DeepMatcher analysis has also investigated whether the complexity of the DL model and the available amount of training data influence the performance. According to the study, with limited amounts of training data, complex models achieve higher accuracy. But with large amounts of training data, the gap between complex and simpler DL models is smaller, and then simpler models could be preferred because they are faster to train.

In this paper, we extend the DeepMatcher analysis to the study of another aspect that plays a fundamental role in the design of a DL solution for ER: the *interpretability* of the model. Understanding the matching predictions of an ER solution is indeed crucial to assess the trustworthiness of the model and to discover its biases.

We have developed a methodological framework, described in Section 2, that leverages LIME [11], a model agnostic technique that produces effective and easily interpretable explanations on the predictions of any classifier. We have adapted LIME to the specific task of ER, and we have defined a procedure, called `Mojito`, to produce interpretable explanations of DL models applied to ER.

We have applied the `Mojito` methodology to evaluate different DL models of the DeepMatcher analysis. Our results have raised interesting issues on the models, demonstrating the importance of developing solutions for the interpretability of DL models in the ER context. As we discuss in Section 3, our method discovered that on some datasets the DL models rely on untrustworthy attributes. In a dataset with a small training set, by means of `Mojito`, we realized that the most complex DL model, which according to the DeepMatcher outcomes should be preferred, is sensitive to a hidden and counterintuitive bias.

aiDM'19, July 5, 2019, Amsterdam, Netherlands
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6802-5/19/07…$15.00
https://doi.org/10.1145/3329859.3329878

$u \in (U)$ (LEFT)

ABV	beer_name	brewery	style
5.60%	Sanibel Red Island Ale	Point Ybel Brewing Company	American Amber / Red Ale

$v \in (V)$ (RIGHT)

ABV	beer_name	brewery	style
5.60%	Point Ybel Sanibel Red Island Ale	Point Ybel Brewing Company	Irish Ale

$T^{u,v}$

```
L0_5.60% L1_Sanibel L1_Red L1_Island L1_Ale L2_Point L2_Ybel L2_Brewing L2_Company L3_American L3_Amber L3_/ L3_Red L3_Ale
R0_5.60% R1_Point R1_Ybel R1_Sanibel R1_Red R1_Island R1_Ale R2_Point R2_Ybel R2_Brewing R2_Company R3_Irish R3_Ale
```

Figure 1: Two tuples that refer to the same entity (a beer), and the corresponding *representative text sequence*, $T^{u,v}$.

2 THE MOJITO METHODOLOGY

We follow the same problem setting defined in DeepMatcher. An *entity* represents a real-world object (such as, a product, an organization), and an *entity mention* is a description of a real-world entity. In this paper, we concentrate on *structured* descriptions, i.e., tuples. Given two sets of tuples, U and V, that share the same schema with attributes $A_0, \ldots, A_{N-1}$, the goal of ER is to find all pairs of tuples $(u \in U, v \in V)$ that refer to the same real-world entity.

ER can thus be considered as a binary classification problem: given a pair of tuples $(u \in U, v \in V)$, the classifier predicts *match* if u and v refer to the same entity, *noMatch* otherwise. In this perspective, a supervised approach to ER corresponds to the definition of a classifier that relies on a training set $\mathcal{T}$, whose elements are labeled pairs of tuples $\langle (u \in U, v \in V), l \rangle$, where l is a label to indicate whether the pair of tuples matches or not.

We aim at "explaining an ER prediction" by providing quantitative information about the relationship between components of the record (e.g., attribute values, or words in text) and the model's predictions. As we see ER as a pairwise binary *classification* task, we can rely on the LIME technique [11], which is designed to explain the predictions of any classifier in an interpretable manner. In the following, first we introduce LIME, then we describe how we have tailored it to design the Mojito methodology.

LIME. LIME comes in a few variants, depending on the classification task, e.g., text or images. For our goals, we focus on text data. Given a text classifier C and a text sequence T, LIME produces a *set of scores*, denoted E, whose elements represent the relevance $r(t) \in [-1, 1]$ of the *word tokens* (i.e., separated by whitespaces) $t \in T$ with respect to a given class c of interest. LIME assigns a positive score to the tokens in T that push the prediction of C towards c, and a negative score to those pushing to any other class $c' \neq c$. The set of scores E comes from the weights of a linear regression classifier on the token space that approximates C's predictions in the proximity of T, and thus can be thought of as a *surrogate* of C for the given T. LIME uses different surrogates for different T values, each trained on a different set of $\langle text, label \rangle$ pairs. The training set for computing scores of the tokens in T w.r.t. c includes T itself (i.e., $\langle T, C(T) \rangle$) and all the "perturbed" neighbours of T, $\langle T_1, C(T_1) \rangle$, $\langle T_2, C(T_2) \rangle, \ldots,$ each generated by dropping few random tokens from T. If labels $C(T_i)$ span uniformly c and the other classes, then the weights learned by the surrogate can provide an accurate relevance score.

Mojito. Given a pair of tuples (u, v), we consider as instance of the ER classification task a special *representative text sequence*, denoted $T^{u,v}$, which is built as follows. Every token in the original tuples is extended with a prefix referring to the position (right or left) of the tuple in the pair, and to the corresponding attribute (for simplicity, we use an ordinal number). $T^{u,v}$ is the concatenation of these extended tokens. Figure 1 shows two tuples, u and v, and the corresponding *representative text sequence*, $T^{u,v}$; note, for instance, the word token "Sanibel" in the second attribute of the left tuple becomes the word "L1_Sanibel" in our classification instance.

In our setting, the input class c is always the class *match*, thus positive and negative scores represent relevant features for the *match* and *noMatch* class, respectively.[1] It is worth noticing that, in the ER context, besides the standard LIME perturbation which drops words, making tuples less similar, it is possible to introduce another form of perturbation by forcing non matching pair of tuples to be more similar (i.e., closer to the *match* class). We can easily achieve this goal by copying attribute values from left to right (or viceversa). Let $T^{u,v}$ be the representative text sequence corresponding to a non-matching pair of tuples (u, v), and let $C(T^{u,v})$ be correctly classified *noMatch*. Many of its neighbours $T_i^{u,v}$ generated by dropping tokens are likely to lead to the same prediction $C(T_i^{u,v}) = noMatch$, yielding poorly informative training data for the surrogate. To this end, in addition to the original version of LIME, that we refer to as LIME_DROP, we implement a variant of LIME that generates neighbours by copying values of random attributes from one tuple to the other, rather than dropping tokens. We refer to this method as LIME_COPY. For example consider again the tuples in Figure 1: LIME_COPY could evaluate the classification by copying the value `American Amber / Red Ale` of attribute `style` from the left tuple to the right one. In this way, we have more chances of obtaining also *match* labels for the neighbors of T, thus learning the relevance of different attributes also for the *noMatch* prediction.

Our methodology takes as input an ER classifier C, such as any of those in [10], and a selected subset of tuple pairs from its training set, with *match* (M) and *noMatch* (NM) labels. For every representative text sequence, we run LIME_DROP and LIME_COPY to compute the set of scores of the attributes and tokens to explain the matching tuples. To interpret the DL model of the classifier, a user can consider scores returned by LIME_DROP and LIME_COPY either at the attribute level, or at the finer token granularity. The *aggregated scores* (e.g., with average) by each token prefix let the user quantify the relevance of each schema's attribute in the classification process, and make an assessment based on domain knowledge. For instance, in the domain of Figure 1 it would be surprising to realize that the beer name attribute has little relevance for the *match* class. The *token scores* can be useful when discovering such anomalies, to investigate further.

[1]Intuitively, we expect many positive scores for matching record pairs and many negative scores otherwise.

dataset	size	train	#match-#noMatch	ℓ
itunes-amazon	539	321	78-100	79.84
beer	450	268	48-100	30.42
walmart-amazon	10,242	6,144	100-100	31.20

Table 1: Dataset in our experiments, along with all the available labelled pairs, number of pairs used to train DL models, number of match-noMatch instances used with `Mojito`, and average record length (ℓ) expressed as the number of tokens.

3 EXPERIENCES WITH `MOJITO`

In this section, we describe our first experiences with `Mojito` by explaining the results of state-of-the art algorithms for ER, and show advantages and limitations of our approach. We consider the space of DL solutions for ER described in [10], which reviews many DL solutions that have been developed for ER tasks. We consider the top two best performing solutions in [10], dubbed `hybrid` and `RNN`, described in the following.

- **`hybrid`** uses a bidirectional recurrent neural network (RNN) with decomposable attention, and a vector concatenation augmented with element-wise absolute difference to form the input to the classifier.
- **`RNN`** consists of a forward RNN processing the input word embedding sequence in its regular order, and a backwards RNN processing the input in reverse. The final representation corresponds to the concatenation of the two RNN outputs.

To the best of our knowledge, `hybrid` is the model with the highest representational power in literature for the ER task. Both `hybrid` and `RNN` represent text tokens using pre-trained *word embedding* [2]. Word embeddings map words from a vocabulary to vectors of real numbers and are a de-facto standard in language modeling. We refer the reader to [10] for more details about the above models.

Experimental setup. Our experiments were performed on the Google Colaboratory environment using a CPU Intel Xeon running at 2.3GHz, 12GB RAM and a GPU NVIDIA Tesla K80. The operating system was Linux, kernel version 4.14, and Python version 3.6.7 (64-Bit). We use datasets from a diverse array of domains and different sizes, as summarized in Table 1. We used the implementation of LIME available at the project site,[2] and the `hybrid` and `RNN` models of the DeepMatcher project.[3]

Attributes. Figures 2–4 show aggregated attribute scores returned by `Mojito` for each of the considered datasets. The plots provide the *average score* both for the matching and the non-matching class (upper and lower plots, respectively), and the corresponding standard deviation (vertical lines). Scores for `hybrid` and `RNN` are shown on the left and right plots, respectively.

In Figure 2 we observe that both `RNN` and `hybrid` heavily rely on the `time` attribute, for both the *match* and *noMatch* class. More than that, attributes that seem more identifying for humans, such as `song_name`, `album_name` and `author_name`, seem to have little or no relevance. In order to check the explanation, we observed that given a pair of songs correctly predicted as *noMatch* by both models, we could typically flip their prediction by simply setting the same time on both songs (using `LIME_COPY`), irrespective to the huge differences in the song and author name.

[2]https://github.com/marcotcr/lime
[3]https://github.com/anhaidgroup/deepmatcher

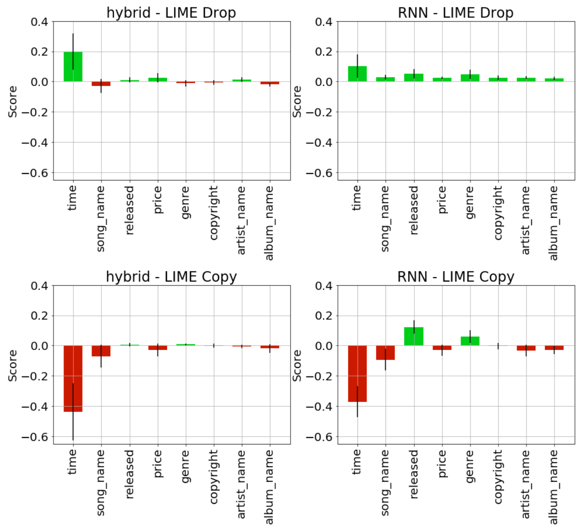

Figure 2: `itunes-amazon` dataset.

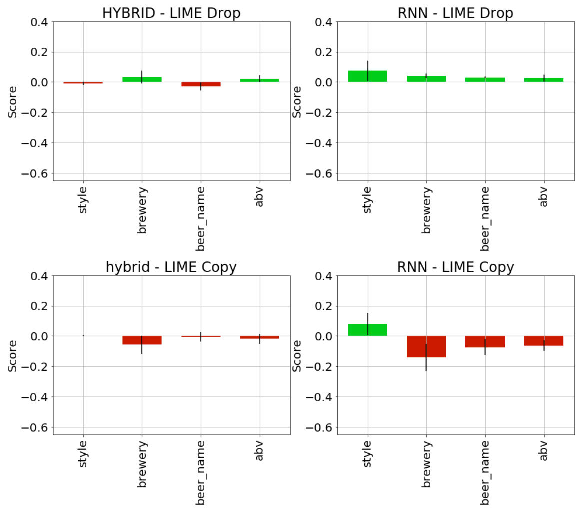

Figure 3: `beers` dataset.

Figure 3 shows that `RNN` seems to reasonably use `style`, `brewery`, and `beer_name` for matching. However, it is surprising that no attribute seems to be relevant for `hybrid`'s *noMatch* predictions. Further experiments with `LIME_DROP` revealed that the top three tokens by relevance for `hybrid`'s *noMatch* class are "Imperial", "Red" and "Ale", with significantly higher relevance than the others (see Figure 5). Also, it turned out that most *noMatch* pairs in the training set are about imperial red ales. Again, given a pair of beers correctly predicted as *match* by both models, we could typically flip `hybrid`'s prediction by simply appending the sequence "Imperial Red Ale" to any of the two beer names. We note also that – with the smaller `beers` training set – the most complex DL model (`hybrid`)

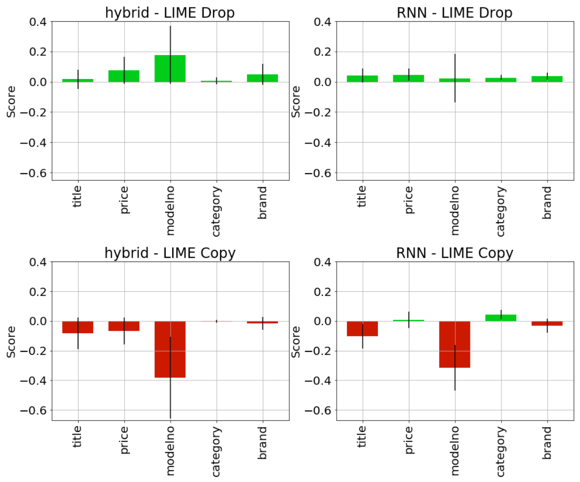

Figure 4: `walmart-amazon` dataset.

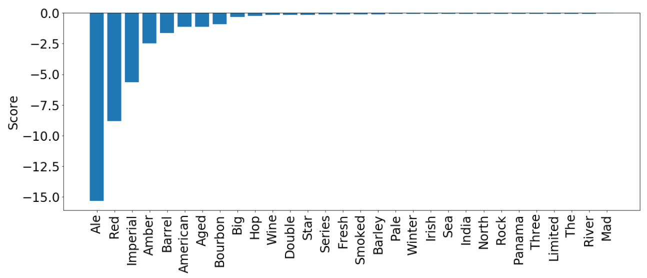

Figure 5: Token scores for the attribute `beer_name` of the beer dataset.

is more sensitive to bias, in contrast to the observations made in the DeepMatcher work [10].

Figure 4 suggests that both `hybrid` and `RNN` are using `ModelNo`, for both the *match* and *noMatch* class.

4 RELATED WORK

To the best of our knowledge, this is the first work reporting experiences using `LIME` over recent DL methods for ER. Therefore, `LIME` [11] and [10] are the works most related to us. Nonetheless, there are some works that indirectly relate to our:

- works that study explainable AI methods, other than `LIME`, recognizing the utility of explanations;
- papers on the benefits of DL for ER, motivating our study.

Explainable AI. `LIME` [11] is a flexible method for explaining different models for text (e.g. random forests) and image classification (e.g. neural networks) by learning a transparent *surrogate* – such as linear regression – for a given instance on an interpretable feature space (e.g., text tokens or *superpixels*). Before `LIME`, others have proposed the use of interpretable models [15]. More specifically, other works providing model-agnostic explanation by learning an approximate interpretable model on the predictions of the original model including [1, 3, 12]. While these works try to approximate the original model as a whole, `LIME` [11] solves the more feasible task of finding a model that approximates the model locally.

DL for ER. A variety of DL solutions have been proposed for matching tasks in the field of natural language processing, such as entity linking and coreference resolution [9]. More recently, there has been extensive interest in applying DL in data cleaning [14], and the first DL solution for the specific ER task, called `DeepER`, is provided in [5]. The work of [10] describes a categorization of these solutions, factoring out their commonalities, and identifying a few solutions as "representative points" in the design space.

Other works. For sake of completeness, we observe that literature is rich with classical *machine learning* methods for ER, from the seminal Fellegi-Sunter model [6], to active learning methods [13], and to the recent `Magellan` [8]. Finally, we mention that ER using crowdsourcing has also become popular [4, 7].

5 FUTURE WORK

Deep learning techniques can achieve high accuracy and outperform classical machine learning solutions for ER. However, our experience shows the importance of the explanations to understand the underlying decisions of DL models, which can learn to solve ER by leveraging features that would hardly work in a real application scenario. We plan to develop further our `Mojito` methodology, extending the analysis to recognize more sophisticated explanations, such as the *joint relevance* of attributes. Another direction is to use explanations to produce suggestions in order to jointly assess bias in the model and improve the training data.

REFERENCES

[1] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K.-R. MÃžller. How to explain individual classification decisions. *Journal of Machine Learning Research*, 11(Jun):1803–1831, 2010.
[2] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.
[3] M. Craven and J. W. Shavlik. Extracting tree-structured representations of trained networks. In *Advances in neural information processing systems*, 1996.
[4] S. Das, P. S. GC, A. Doan, J. F. Naughton, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, and Y. Park. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *SIGMOD*, 2017.
[5] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang. Distributed representations of tuples for entity resolution. *PVLDB*, 11(11), 2018.
[6] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.
[7] S. Galhotra, D. Firmani, B. Saha, and D. Srivastava. Robust entity resolution using random graphs. In *SIGMOD*, 2018.
[8] P. Konda, S. Das, P. Suganthan GC, A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. Naughton, et al. Magellan: Toward building entity matching management systems. *PVLDB*, 9(12), 2016.
[9] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.
[10] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, and V. Raghavendra. Deep learning for entity matching: A design space exploration. In *SIGMOD*, 2018.
[11] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *KDD*, 2016.
[12] I. Sanchez, T. Rocktaschel, S. Riedel, and S. Singh. Towards extracting faithful and descriptive representations of latent variable models. *AAAI Spring Syposium on Knowledge Representation and Reasoning (KRR): Integrating Symbolic and Neural Approaches*, 2015.
[13] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *KDD*, 2002.
[14] S. Thirumuruganathan, N. Tang, and M. Ouzzani. Data curation with deep learning [vision]: Towards self driving data curation. *arXiv preprint arXiv:1803.01384*, 2018.
[15] F. Wang and C. Rudin. Falling rule lists. In *Artificial Intelligence and Statistics*, pages 1013–1022, 2015.