EmbDI: Generating Embeddings for Relational Data Integration

(Discussion Paper)

Riccardo Cappuzzo[1], Paolo Papotti[1] and Saravanan Thirumuruganathan[2]

[1]EURECOM, France
[1]EURECOM, France
[2]QCRI, Qatar

Abstract

Deep learning techniques have been used with promising results for data integration problems. Some methods use *pre-trained* embeddings that were trained on a large corpus such as Wikipedia. However, they may not always be an appropriate choice for enterprise datasets with custom vocabulary. Other methods adapt techniques from natural language processing to obtain embeddings for the enterprise's relational data. However, this approach blindly treats a tuple as a sentence, thus losing a large amount of contextual information present in the tuple. We propose algorithms for obtaining *local embeddings* that are effective for data integration tasks on relational databases. We describe a graph-based representation that allows the specification of a rich set of relationships inherent in the relational world. Then, we propose how to derive sentences from such a graph that effectively "describe" the similarity across elements (tokens, attributes, rows) in the datasets. The embeddings are learned based on such sentences. Our experiments show that our framework, EmbDI, produces promising results for data integration tasks such as entity resolution, both in supervised and unsupervised settings.

Keywords

Data Integration, Word Embeddings

1. Introduction

The problem of data integration concerns the combination of information from heterogeneous relational data sources, which is recognized as an expensive task for humans [1]. While traditional approaches require substantial effort from domain scientists to generate features and labeled data or domain specific rules, there has been increasing interest in achieving accurate data integration with deep learning methods to reduce the human effort. Embeddings have been successfully used for this goal in data integration tasks such as entity resolution [2, 3, 4, 5, 6, 7], schema matching [8, 9], identification of related concepts [10], and data curation in general [1]. Typically, these works fall into two dominant paradigms based on how they obtain word embeddings. The first is to reuse *pre-trained* word embeddings computed on a generic corpus

SEBD 2021: The 29th Italian Symposium on Advanced Database Systems, September 5-9, 2021, Pizzo Calabro (VV), Italy
riccardo.cappuzzo@eurecom.fr (R. Cappuzzo); paolo.papotti@eurecom.fr (P. Papotti); sthirumuruganathan@hbku.edu.qa (S. Thirumuruganathan)
0000-0003-0651-4128 (P. Papotti)

for a given task. The second is to build *local* word embeddings that are specific to the dataset. These methods treat each tuple as a sentence by reusing the same techniques for learning word embeddings employed in natural language processing.

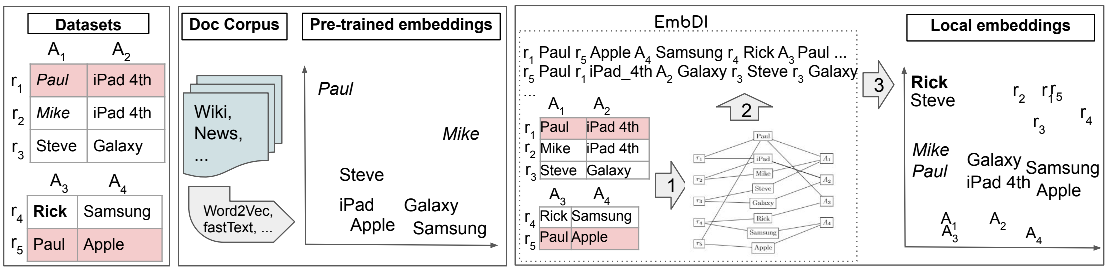

Figure 1: A vector space learned from text (prior methods) and from data (EmbDI).

However, both approaches fall short in some circumstances. Enterprise datasets contain custom vocabulary, as in the small datasets in the left-hand side of Figure 1. The pre-trained embeddings do not capture the semantics expressed by these datasets and do not contain embeddings for the word "Rick". Approaches that treat a tuple as a sentence miss a number of signals such as attribute boundaries, integrity constraints, and so on. Moreover, existing approaches do not consider the generation of embeddings from heterogeneous datasets, with different attributes and alternative value formats. These observations motivate the generation of *local* embeddings for the *relational* datasets at hand. We advocate for the design of such local embeddings that leverage both the relational nature of the data and the downstream task of data integration.

Tuples are not sentences. Simply adapting embedding techniques originally developed for textual data ignores the richer set of semantics inherent in *relational* data. Consider a cell value $t[A_i]$ of an attribute A_i in tuple t, e.g., "Mike" (in italic) in the first relation from the top. Conceptually, it has a semantic connections with both other attributes of tuple t (such as "iPad 4th") and other values from the domain of attribute A_i (such as "Paul", also in italic in the figure).
Embedding generation must span different datasets. Embeddings must be trained using heterogeneous datasets, so that they can meaningfully leverage and surface similarity across data sources. A notion of similarity between different types of entities, such as tuples and attributes, must be developed. Tuple-tuple and attribute-attribute similarity are important features for data integration.

There are multiple challenges to overcome. First, it is not clear how to encode the semantics of the relational datasets in the embedding learning process. Second, datasets may share limited amount of information, have different schemas, and contain a different number of tuples. Finally, datasets are often incomplete and noisy. The learning process is affected by low information quality, generating embeddings that do not correctly represent the semantics of the data.

We introduce EmbDI, a framework for building relational, local embeddings for data integration that introduces a number of innovations to overcome the challenges above. We identify crucial components and propose effective algorithms for instantiating each of them. EmbDI is designed to be modular so that anyone can customize it by plugging in other algorithms and benefit from the continuing improvements from the deep learning and database communities. The two main contributions in our solution are the following.

1. Graph Construction. We use a compact tripartite graph-based representation of relational datasets that effectively represents syntactic and semantic data relationships. Specifically, we use three types of nodes. *Token* nodes correspond to the unique values found in the dataset. *Record Id* nodes (RIDs) represent a unique token for each tuple. *Column Id* nodes (CIDs) represent a unique token for each column/attribute. These nodes are connected by edges based on the structural relationships in the schema. This graph is a compact representation of the original datasets that highlights overlap and explicitly represent the primitives for data integration tasks, i.e., records and attributes.

2. Embedding Construction. We formulate the problem of obtaining local embeddings for relational data as a graph embeddings generation problem. We use random walks to quantify the similarity between neighboring nodes and to exploit metadata such as tuple and attribute IDs. This method ensures that nodes that share similar neighborhoods will be in close proximity in the final embeddings space. The corpus that is used to train our local embeddings is generated by materializing these random walks.

In this discussion paper, we report results for the entity resolution task and refer the reader to the extended version for more experiments [11].

Outline. Section 2 introduces background about embeddings. Section 3 highlights the main challenges and details the major components of the framework. Section 4 concludes the paper by reporting experiments validating our approach.

2. Background

Embeddings. Embeddings map an entity to a high dimensional real valued vector. The mapping is performed in such a way that the geometric relation between the vectors of two entities represents their co-occurrence/semantic relationship. Algorithms used to learn embeddings rely on the notion of "neighborhood": if two entities are similar, they frequently belong to the same contextually-defined neighborhood. When this occurs, the algorithm forces the vectors that represent the two entities to be close to each other in the vector space.

Word Embeddings [12] are trained on a large corpus of text and produce as output a vector space where each word in the corpus is represented by a vector. The vectors for words that occur in similar context – such as SIGMOD and VLDB – are in proximity to each other. Popular architectures for learning embeddings include continuous bag-of-words (CBOW) or skip-gram (SG).

Node embeddings [13] map graph nodes to a high dimensional vector space so that the likelihood of preserving node neighborhoods is maximized. One way to achieve this is by performing random walks starting from each node. Node embeddings are often based on the SG model, as it maximizes the probability of observing a node's neighborhood given its embedding. By varying the type of random walks used, one obtains diverse types of embeddings.

Embeddings for Relational Datasets. Termite [10] projects tokens from structured and unstructured data into a common representational space that could then be used for identifying related concepts. RetroLive [14] produces embeddings that combine relational and semantic information through a retrofitting strategy. There has been prior work that adopt embeddings for specific tasks like entity matching [2, 3] and schema matching [9]. Our goal is to learn relational embeddings tailored for data integration that can be used for multiple tasks.

3. Challenges and Proposed Solution

Consider the scenario where one utilizes pre-trained embeddings, such as word2vec, for the tokens in two small datasets, as reported in Figure 1. Pre-trained embeddings suffer from a number of issues when we use them to model the relations.

1. A number of words, such as “Rick”, in the dataset are not in the pre-trained embedding. This is especially problematic for enterprise datasets where tokens are often unique and not found in pre-trained embeddings.
2. Embeddings might contain geometric relationships that exist in the corpus they were trained on, but that are missing in the relational data. For example, the embedding for token “Steve” is closer to tokens “iPad” and “Apple” even though it is not implied in the data.
3. Relationships that do occur in the data, such as between tokens “Paul” and “Mike”, are not observed in the pre-trained vector space.

Learning local embeddings from the relational data often produces better results. However, computing embeddings for non integrated data sources is a non trivial task. This becomes especially challenging in settings where data is scattered over different datasets with heterogeneous structures, different formats, and only partially overlapping content. Prior approaches express such datasets as sentences to be consumed by word embedding methods. However, we find that these solutions are still sub-optimal for downstream data integration tasks.

3.1. Constructing Local Relational Embeddings

Our framework, EmbDI, consists of three major components, as depicted in the right-hand side of Figure 1.

1. In the *Graph Construction* stage, we transform the relational dataset in a compact tripartite graph that encodes various relationships inherent in it. Tuple and attribute ids are treated as first class citizens.
2. Given this graph, the next step is *Sentence Construction* through the use of biased random walks. These walks are carefully constructed to avoid common issues such as rare words and imbalance in vocabulary sizes. This produces as output a series of sentences.
3. In *Embedding Construction*, the corpus of sentences is passed to an algorithm for learning word embeddings. Depending on available external information, we optimize the graph and the workflow to improve the embeddings’ quality.

Why construct a Graph? Prior approaches for local embeddings seek to directly apply an existing word embedding algorithm on the relational dataset. Intuitively, all tuples in a relation are modeled as sentences by breaking the attribute boundaries. The corpus of sentences for each tuple in the relation is then used to train the embedding. This approach produces embeddings that are customized to that dataset, but it also ignores signals that are inherent in relational data. We represent the relational data as a graph, thus enabling a more expressive representation with a number of advantages. First, it elegantly handles many of the various relationships between entities that are common in relational datasets. Second, it provides a straightforward way to incorporate external information such as "two tokens are synonyms of each other". Finally, a graph representation enables a unified view over different datasets that is invaluable for learning embeddings for data integration.

Simple Approaches. Consider a relation R with attributes $\{A_1, A_2, \ldots, A_m\}$. Let t be an arbitrary tuple and $t[A_i]$ the value of attribute A_i for tuple t. A naive approach is to create a chain graph where tokens corresponding to adjacent attributes such as $t[A_i]$ and $t[A_{i+1}]$ are connected. This will result in m edges for each tuple. Of course, if two different tuples share the same token, then they will reuse the same node. However, relational algebra is based on set semantics, where the attributes do not have an inherent order. So, simplistically connecting adjacent attributes is doomed to fail. Another extreme is to create a complete subgraph, where an edge exists between all possible pairs of $t[A_i]$ and $t[A_{i+1}]$. Clearly, this will result in $\binom{m}{2}$ edges per tuple. This approach results in the number of edges is quadratic in the number of attributes and ignores other token relationships such as "token t_1 and token t_2 belong to the same attribute".

Relational Data as Heterogeneous Graph. We propose a graph with three types of nodes. *Token* nodes correspond to the content of each cell in the relation. Multi-word tokens may be represented as a single entity, get split over multiple nodes or use a mix of the two strategies. *Record Id* nodes (RIDs) represent tuples, *Column Id* nodes (CIDs) represent columns/attributes. These nodes are connected by edges according to the structural relationships in the schema.

Consider a tuple t with RID r_t. Then, nodes for tokens corresponding to $t[A_1], \ldots, t[A_m]$ are connected to the node r_t. Similarly, all the tokens belonging to a specific attribute A_i are connected to the corresponding CID, say c_i. This construction is generic enough to be augmented with other types of relationships. Also, if we know that two tokens are synonyms (e.g. via wordnet), this information could be incorporated by reusing the same node for both tokens. Note that a token could belong to different record ids and column ids when two different tuples/attributes share the same token. Numerical values are rounded to a number of significant figures decided by the user, then they are assigned a node like regular categorical values; null values are not represented in the graph.

Graph Traversal by Random Walks. To generate the distributed representation of every node, we produce a large number of random walks and gather them in a training corpus where each random walk corresponds to a sentence. Random walks allows a richer and more diverse set of neighborhoods than the encoding of a tuple as a *single* sentence. For example, a walk starting from node 'Paul' could go to node A_3, and then to node 'Rick'. This walk implicitly defines the neighborhood based on attribute co-occurrence. Similarly, the walk from 'Paul' could go to 'r_5' and then to 'Apple', incorporating the row level relationships. Our approach is agnostic to the specific type of random walk used. To better represent all nodes, we assign a "budget" of random walks to each of them and guarantee that all nodes will be the starting point of at least as many random walks as their budget. After choosing the starting point T_i, the random walk is generated by choosing a neighboring RID of T_i, R_j. The next step in the random walk will then be chosen at random among all neighbors of node R_j, for example by moving on C_a. Then, a new neighbor of C_a will be chosen and the process will continue until the random walk has reached the target length. We use uniform random walks in most of our experiments to guarantee good execution times on large datasets, while providing high quality results.

Embedding Construction. The generated sentences are then pooled together and used to train the embeddings algorithm. Our approach is agnostic to the actual word embedding algorithm used. We piggyback on the plethora of effective embeddings algorithms such as word2vec, GloVe, fastText, and so on. We discuss the hyperparameters for embedding algorithms such as learning method (either CBOW or Skip-Gram), dimensionality of the embeddings, and size of context window in the full version of the paper.

Using Embeddings for Integration. Once the embeddings are trained, they can be used for common data integration tasks. We describe *unsupervised* algorithms that employ the embeddings produced by EmbDI to perform tasks widely studied in data integration. The algorithms exploit the distance between embeddings of Column and Record IDs for schema matching and entity resolution, respectively; details are reported in the full version of the paper [11].

4. Experiments

We show the positive impact of our embeddings for entity resolution, more results on multiple data integration tasks are reported in the full version of the paper. Experiments have been conducted on a laptop with a CPU Intel i7-8550U, 8x1.8GHz cores and 32GB RAM.

Datasets and Pre-trained Embeddings. We used 8 datasets from the literature [15, 2, 3, 16] and a dataset with a larger schema (IM) that we created starting from open data (https://www.imdb.com/interfaces/, https://grouplens.org/datasets/movielens/). For the majority of the scenarios, less than 10% of the distinct data values are overlapping across the two datasets.

Pre-trained word embeddings have been obtained from fastText [17]. We relied on state of the art methods to combine words in tuples and to obtain embeddings for words that are not in the pre-trained vocabulary [2].

Algorithms. We test four algorithms for the generation of local embeddings. All local methods make use of our tripartite graph and exploit record and column IDs in the integration tasks. The first method is Basic, which creates embeddings from permutations of row tokens and sentences with samples of attribute tokens. The second method is Node2Vec [13], a widely used algorithm for learning node representation on graphs. Given our graph as input, it learns vectors for all nodes. The third method is Harp [18], a state of the art algorithm that learns embeddings for graph nodes by preserving higher-order structural features. This method represents general meta-strategies that build on top of existing neural algorithms to improve performance. The fourth method is EmbDI, as presented in Section 3.1 (https://gitlab.eurecom.fr/cappuzzo/embdi), with walks (sentences) of size 60, 300 dimensions for the embeddings space, the Skip-Gram model in word2vec with a window size of 3, and different tokenization strategies to convert cell values in nodes (details reported in the full paper).

We also test our local embeddings in the *supervised* setting with a state of the art ER system (DeER$_L$), comparing its results to the ones obtained with pre-trained embeddings (DeER$_P$). As baseline for the *unsupervised* case, we use our matching algorithm with pre-trained embeddings (fastTxt).

Metrics. We measure the quality of the results w.r.t. hand crafted ground truth tuple pairs with precision, recall, and their combination (F-measure).

	Unsupervised						Supervised		Task specific	
	Pretrain	Local					(5% labelled)		(5% labelled)	
	fast Txt	EmbDI -S	EmbDI -F	EmbDI -O	Node 2Vec	Harp	DeER$_P$	DeER$_L$	DeER$_P$	DeER$_L$
BB	0.59	0.50	0.82	**0.86**	**0.86**	**0.86**	0.51	0.53	0.54	0.58
WA	0.58	0.59	0.75	**0.81**	mem	0.78	0.58	0.62	0.62	0.63
AG	0.18	0.14	0.57	0.59	0.70	**0.71**	0.53	0.56	0.58	0.62
FZ	0.99	0.98	0.99	0.99	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
IA	0.10	0.09	0.09	0.11	mem	0.14	0.76	0.81	0.77	**0.82**
DA	0.72	0.95	0.94	0.95	0.87	**0.97**	0.84	0.89	0.86	0.90
DS	0.80	0.85	0.75	**0.92**	mem	0.81	0.80	0.87	0.82	0.91
IM	0.31	0.90	0.64	0.94	mem	**0.95**	0.82	0.88	0.84	0.91

Table 1
F-Measure results for Entity Resolution (ER).

ER Results. We study both unsupervised and supervised settings. To enable baselines to execute these datasets, we aligned the attributes with the ground truth. EmbDI can handle the original scenario where the schemas have not been aligned with a limited decrease in ER quality.

Results in Table 1 for unsupervised settings show that EmbDI-O embeddings obtain the best quality results in three scenarios and second to the best in four cases. In every case, local embeddings obtained from our graph outperform pre-trained ones. For supervised settings, using local embeddings instead of pre-trained ones increases the quality of an existing system. In this case, supervised DeER shows an average 5% absolute improvement in F-measure with 5% of the ground truth passed as training data. The improvements decrease to 4% with more training data (10%). Local embeddings obtained with the Basic method lead to 0 rows matched.

Compared to Node2Vec and Harp, the execution of EmbDI is much faster and is able to compute local embeddings for all small and medium size datasets in minutes on a commodity laptop. For example, it takes 2 minutes for 7.4k tuples and 19 minutes for 25k tuples versus 40 and 12 minutes with Harp, respectively. EmbDI embedding creation takes on average about 80% of the total execution time, while graph generation takes less than 1%, and sentence creation the remaining 19%.

Acknowledgement This work has been partially supported by the ANR grant *ANR-18-CE23-0019* and by the IMT *Futur & Ruptures* program “AutoClean”.

References

[1] S. Thirumuruganathan, N. Tang, M. Ouzzani, A. Doan, Data curation with deep learning, EDBT (2020).

[2] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, N. Tang, Distributed representations of tuples for entity resolution, PVLDB 11 (2018) 1454–1467.

[3] S. Mudgal, H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, Deep learning for entity matching: A design space exploration, in: SIGMOD, 2018.

[4] C. Zhao, Y. He, Auto-em: End-to-end fuzzy entity-matching using pre-trained deep models and transfer learning, in: WWW, 2019, pp. 2413–2424.

[5] Ö. Ö. Çakal, M. Mahdavi, Z. Abedjan, CLRL: feature engineering for cross-language record linkage, in: EDBT, 2019, pp. 678–681.

[6] J. Kasai, K. Qian, S. Gurajada, Y. Li, L. Popa, Low-resource deep entity resolution with transfer and active learning, arXiv preprint arXiv:1906.08042 (2019).

[7] R. Singh, V. V. Meduri, A. K. Elmagarmid, S. Madden, P. Papotti, J. Quiané-Ruiz, A. Solar-Lezama, N. Tang, Synthesizing entity matching rules by examples, Proc. VLDB Endow. 11 (2017) 189–202.

[8] R. C. Fernandez, E. Mansour, A. A. Qahtan, A. Elmagarmid, I. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, N. Tang, Seeping semantics: Linking datasets using word embeddings for data discovery, in: ICDE, 2018.

[9] C. Koutras, M. Fragkoulis, A. Katsifodimos, C. Lofi, Rema: Graph embeddings-based relational schema matching, SEA Data workshop (2020).

[10] R. C. Fernandez, S. Madden, Termite: a system for tunneling through heterogeneous data, arXiv preprint arXiv:1903.05008 (2019).

[11] R. Cappuzzo, P. Papotti, S. Thirumuruganathan, Creating embeddings of heterogeneous relational datasets for data integration tasks, in: SIGMOD, ACM, 2020.

[12] J. Turian, L. Ratinov, Y. Bengio, Word representations: a simple and general method for semi-supervised learning, in: ACL, ACL, 2010, pp. 384–394.

[13] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: SIGKDD, ACM, 2016, pp. 855–864.

[14] M. Günther, M. Thiele, E. Nikulski, W. Lehner, Retrolive: Analysis of relational retrofitted word embeddings, EDBT (2020).

[15] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, X. Zhu, Corleone: hands-off crowdsourcing for entity matching, in: SIGMOD, 2014.

[16] S. Das, P. S. G. C., A. Doan, J. F. Naughton, G. Krishnan, R. Deep, E. Arcaute, V. Raghavendra, Y. Park, Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services, in: SIGMOD, 2017.

[17] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching word vectors with subword information, TACL 5 (2017) 135–146.

[18] H. Chen, B. Perozzi, Y. Hu, S. Skiena, HARP: hierarchical representation learning for networks, CoRR abs/1706.07845 (2017). URL: http://arxiv.org/abs/1706.07845. `arXiv:1706.07845`.