Data as a Language: A Novel Approach to Data Integration

Christos Koutras
supervised by Asterios Katsifodimos, Christoph Lofi and Geert-Jan Houben
Delft University of Technology
c.koutras@tudelft.nl

ABSTRACT

In modern enterprises, both operational and organizational data is typically spread across multiple heterogeneous systems, databases and file systems. Recognizing the value of their data assets, companies and institutions construct *data lakes*, storing disparate datasets from different departments and systems. However, for those datasets to become useful, they need to be cleaned and integrated. Data can be well documented, structured and encoded in different schemata, but also unstructured with implicit, human-understandable semantics. Due to the sheer scale of the data itself but also the multitude of representations and schemata, data integration techniques need to scale without relying heavily on human labor. Existing integration approaches fail to address hidden semantics without human input or some form of ontology, making large scale integration a daunting task.

The goal of my doctoral work is to devise scalable data integration methods, employing modern machine learning to exploit semantics and facilitate discovery of novel relationship types. In order to capture semantics with minimal human intervention, we propose a new approach which we call *Data as a Language* (DaaL). By leveraging *embeddings* from the *Natural Language Processing* (NLP) literature, DaaL aims at extracting semantics from structured and semi-structured data, allowing the exploration of relevance and similarity among different data sources. This paper discusses existing data integration mechanisms and elaborates on how NLP techniques can be used in data integration, alongside challenges and research directions.

1. INTRODUCTION

Enterprises encounter data-related problems, such as the integration of multiple databases, spreadsheet files, logs as well as semi-structured and unstructured documents; data is stored across multiple storage systems, generally dirty and modelled abstractly with respect to the corresponding source. For the most part, data integration has been a manual process. Traditionally, database administrators would create one federated schema integrating various databases, allowing data analysts to query all of them at the same time. To automate this process, administrators relied on *Schema Matching*, i.e. the process of capturing potential relationships between different data sources and models.

Nowadays such a task is nearly impossible to rely on humans, since the number of data sources has increased dramatically. Knowledge about relationships among data assets is an essential building block for integrating data to be used in larger scale applications. However, most of the existing integration methods [3, 6, 15] rely on syntax, i.e. the symbolic representation of data as found in a database without considering what they semantically represent (their underlying meaning), which constraints the quality and amount of matches. Furthermore, recent work that incorporates semantics [7], does not consider data instances and focuses only on the schema elements.

Departing from existing schema matching methods, we envision a novel approach, which we term *Data as a Language* (DaaL). Essentially, DaaL designates a method for transforming elements of structured (e.g. rows of relational tables) and semi-structured data (e.g. log entries) into a non-directed graph, whose traversal outputs a number of documents. These become the input to Natural Language Processing (NLP) techniques, which leverage recent advancements in Deep Neural Networks (DNNs). Research on NLP methods has proven that they are suitable for capturing semantics, when the training sample consists of *large quantities of text corpora* which provides also some *context information.* In a typical data lake, the abundance of (semi)structured data satisfies the requirement of quantity, but proper context is difficult to guarantee since non-textual data does not provide a standard sequence; in contrast, textual data adheres to a specific syntax. DaaL aims at filling this gap, by providing a way to strengthen context when dealing with raw data values, in order to enable leveraging of NLP approaches that capture semantics. DaaL will facilitate relationship discovery between different data schemata and instances, which, in turn, can improve the accuracy of demanding data discovery and integration tasks.

2. DATA-AS-A-LANGUAGE

In this section we describe the Data as a Language (DaaL) approach for capturing relationships among elements of structured and semi-structured data of various schemata, from different data sources.

Proceedings of the VLDB 2019 PhD Workshop, August 26th, 2019. Los Angeles, California. Copyright (C) 2019 for this paper by its authors. Copying permitted for private and academic purposes.

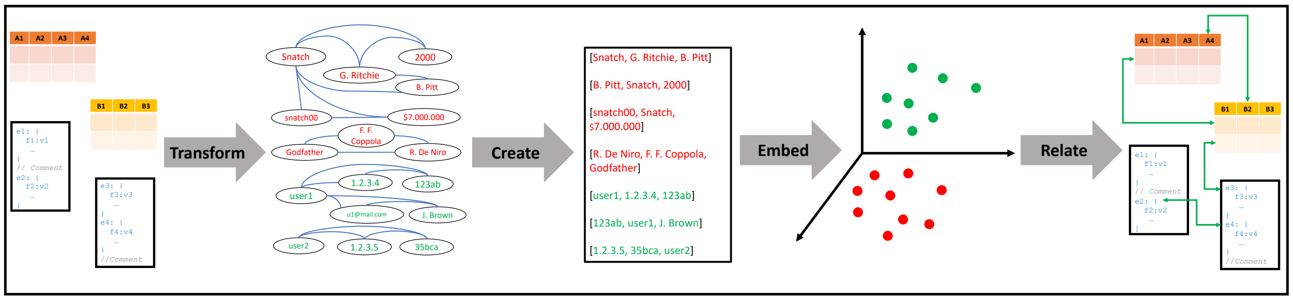


Figure 1: The DaaL pipeline. Data is transformed into a graph, from which documents are constructed to train vector representations. Data instance embeddings can then be used to discover relationships between disparate data elements.

2.1 DaaL in a Nutshell

The DaaL approach consists of the three stages, as depicted in Figure 1:

- **Transforming Data to a Graph.** Data from various sources, with complete or partial structure, go through the transformation stage. In that stage, DaaL processes data elements, such as tuples of relational tables, and outputs a non-directed graph connecting data elements with edges.

- **Creating Documents from the Graph.** Next, we create documents consisting of data element tokens by traversing the graph.

- **Producing Embeddings from Documents.** Consequently, the documents serve as input for training vector representations, commonly termed as *embeddings*, of data elements (e.g. entire relational tuples or individual attribute values, entries in semi-structured files) using existing learning and graph-based NLP techniques. These embeddings are constructed in such a way that elements which share the same context have similar vector representations.

- **Finding Semantic Relationships.** In the final stage, we use the embeddings in order to calculate similarity between different data assets. Since embeddings of semantically related data elements are close to each other, these similarities help us capture and materialize relationships among them.

The output of the DaaL pipeline will consist of our initial datasets enhanced with knowledge about semantically related content among data items (e.g. a relationship between two columns of different schemata). We can then leverage this knowledge to perform data discovery and integration tasks. In what follows, we delve into details about each of the above stages, discussing existing work and challenges.

2.2 Transforming Data to a Graph

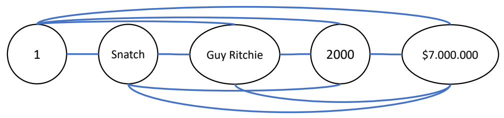
As a first step, we need to devise a method for transforming (semi)structured data into a non-directed graph. We do so in order to create some reasonable relevance between schema information and data values sharing some context, which will later be of high importance for producing accurate vector representations. In this section, we focus on how to produce such a graph from either i) *relational data*, or ii) *semi-structured files*.

Transforming Relational Data. Consider a relation R, and its set of m attributes $\{A_1, \ldots, A_m\}$. For each tuple t contained in the instance of R we want to create a connected component of the graph. One alternative would be, for each tuple, to create nodes representing each data value and edges between adjacent ones. However, we would relate only data elements that are adjacent, whereas we would like to relate them on a row basis.

Therefore, another approach would be to create a *clique* for each relational tuple in the input. More specifically, for each individual attribute value we create a node and connect it through an edge with all other values in the same row. This way, we manage to provide full context for relational data elements, since we incorporate row information.

EXAMPLE 1. *Consider the following tuple from a "Movies" relation with the attribute set {ID, Title, Director, Production_Year, Budget} and the clique created out of it:*

```
(1, Snatch, Guy Ritchie, 2000, $7.000.000)
```



Transforming Data from Semi-Structured Files. Consider a semi-structured file $\mathcal{F}$ which contains a variety of entries e referring to various entities. Each of these entries contains a number of fields f, each accompanied by its value. Transforming each such entry e to a connected component of the graph is similar to the approach used for relational data; in this case, we treat field values as attribute values.

Challenges. Transforming data into a graph is far from straightforward. Below we highlight three research challenges:

- **Graph Construction.** Creating a clique for each row or entry will be prohibitively costly with respect to storage, when having a massive amount of data as input. Therefore, in the case of relational data, if we know the primary key of a relation, we can create only an edge between the value of the primary key and each attribute in the row. For semi-structured data, we could identify fields that represent keys (e.g. if the field name is *ID*), and proceed in the same way. Hence, primary key values will act like hubs for relating attribute values of a single row.

- **Incorporating Schema Information.** We could also incorporate schema information into the graph, such as relation (entry) and attribute (field) names and their relationships with the corresponding attribute (field) values. In particular, we could create nodes for these entities too and connect them with an edge to the corresponding values and other schema-level elements they are found together with (e.g. an edge between attribute names of the same relation). An alternative approach would be to incorporate schema information inside the corresponding data values; this would also help distinguish same data values with different contextual meaning.

- **Capturing Entries.** Semi-structured datasets may also contain comments and system-generated messages, which we might not want to take into consideration when transforming to text. In order to extract e.g., log entries from the files, we could leverage approaches such as Datamaran [8], given some essential assumptions on the format of a file.

2.3 Creating Documents from the Graph

In order to train neural networks to produce vector representations for the data elements in the input, we need to create documents that provide some context. Towards this direction, we are going to use the graph of the previous step and construct documents through its traversal.

Based on [9], we propose for each node in the graph to perform a specified number of random walks of a given length, to explore diverse neighborhoods. In this fashion, each such random walk will represent a sequence of data values and will provide a different context for each of them. This way, we are able to provide a lot of useful context for each data element in the input, and bypass the shortcomings of syntax absence in non-textual data. The output of this stage consists of a number of such documents containing tokenized data values with respect to the random walk sequences.

2.4 Producing Embeddings from Documents

The idea of creating similar representations for words that appear in the same context has its roots in the distributional hypothesis [11], which states that such words tend to have a similar meaning. The recent progress made in neural networks facilitated the introduction of distributed representations called *embeddings*, which relate words to vectors of a given dimensionality. Towards this direction, numerous word embedding methods have been proposed in the literature, with the most popular ones being Word2Vec [13], GloVe [14], and fastText [2] which even produces character embeddings, making it possible to deal with out-of-vocabulary words. Apart from single word embeddings, there exist methods that try to produce vector representations for sentences or even paragraphs [12]. The authors in [7] propose the idea of *coherent groups* for incorporating single word embeddings into a similarity measure between two groups of words.

In [5], the authors introduce two approaches for composing distributed representations of relational tuples. The simplest one suggests that a tuple embedding is a concatenation of the embeddings of its attribute instances. They then propose using Recurrent Neural Networks (RNNs) with Long Short Term Memory (LSTM) in order to produce tuple embeddings out of single word ones, by taking into consideration the relationship and order between different attribute values inside a tuple. The authors also propose a method for handling unknown words, called *vocabulary retrofitting*, and two alternatives for learning embeddings when the data domain is too technical: i) assuming tuples are documents, and ii) training on a corpus of text with related semantics. We avoid these issues by presenting a general framework for producing word embeddings, by transforming data into a collection of documents comprising of related data values, which could serve as training input.

Training DaaL Embeddings. After transforming data into documents using the methods described previously, we make use of this newly crafted knowledge by training neural networks to produce vector representations that capture context and semantics. Towards this direction, we feed the tokenized documents in Word2Vec [13] or fastText [2] to receive individual word embeddings. When tuning the parameters of these methods, we need to pay attention to the window size around each word, which determines in what extent we take into consideration the surroundings to output the word embedding. In our case, we won't need a large window size, since we create a lot of different contexts for individual data elements, using the random walks for creating the documents; thus, a smaller window size will guarantee accurate and more distinct vector representations.

2.5 Finding Semantic Relationships

There has been a lot of prior work on capturing relationships between different datasets, or as previously defined, Schema Matching [15]. Data Tamer [16] has a *Schema Integration* module which allows each ingested attribute to be matched against a collection of existing ones, by making use of a variety of algorithms called *experts*. In [6] the authors focus on building Knowledge Graphs, where different datasets are correlated with respect to their content or schemata. [3] uses a Linkage Graph in order to support data discovery. All of these methods, rely only on syntax, based on similarity computation (e.g. *Jaccard Similarity*) between pairs of column signatures [1] . Moreover, provenance of datasets that are used within Google is explored in [10] by looking into production logs.

In an attempt to avoid considering only syntax, the authors in [17] propose an alternative algorithm for matching attributes of several relations, by clustering them based on the distribution of their values, whereas in [4] matching is performed with respect to a corpus of existing schema mappings, which serve as training samples for different training modules called *base learners*. [7] tries to build relationships between relations and columns of different databases with respect to a given ontology, by making use of both semantics and syntax; yet they avoid data instances. Therefore, DaaL is the first method to incorporate schema information and data instances to capture relationships between data elements with respect to the underlying meaning they share.

Our Approach. The trained embeddings could facilitate discovery of novel relationship types, other than just finding attributes that relate to the same entity. This is due to the fact that vector representations of attribute (field) values are affected by their context inside their relation (entry), which leads to capturing relationships with attributes (fields) of other relations (entries) that share the same context.

However, in order to materialize a relationship between two data elements (e.g. relational tuples, columns of relations) we need to devise a method that discovers it. One alternative could be calculating similarities between the vector representations of data, and if they are above a given threshold, signal a potential relationship between them. In addition, clustering similar embeddings could facilitate finding groups of semantically related data elements. Nonetheless, the challenge of proposing methods that take advantage of embeddings for accurately capturing semantic relevance is very demanding and open, since we want embeddings to be applicable to any given scenario.

3. APPLICATIONS

The output of the DaaL pipeline comprises the initial datasets together with numerous relationships among them. Thus, a Data Integration task could use this valuable information to explore any meaningful data correlation stemming from the various data sources. In addition, our proposed approach is the first to deal with finding correspondences among semi-structured files, without having to transform them into some structured format, which is something that the authors in [8] deal with.

Interestingly enough, DaaL gives also the opportunity to augment information available in a specific schema. For instance, we can enhance a relational table with extra attributes, found in other schemata, containing data values which have similar representations with the respective ones of the table. Most importantly, this could be done before getting the results of the time costly matching module, since we immediately can take advantage of the information we get from the embeddings (e.g., the most similar data elements to a given one in vector space).

4. RESEARCH PLAN

Experimental Evaluation Framework. We are currently creating a unified evaluation framework for comparing the schema matching methods proposed in the literature throughout the years. We aim at developing an open source framework for experimenting on Schema Matching and comparing the DaaL approach with previous ones based on a standardized set of evaluation techniques.

Implementing DaaL. The DaaL pipeline comprises four stages, namely *i*) transformation of data into a graph, *ii*) creation of documents that serve as input to the DNNs, *iii*) training DNNs and creating embeddings and subsequently *iv*) using those embeddings. Each of those stages poses its own challenges. We aim at building a streamlined system to enable plugging-in and experimenting with different techniques to generate text from data, to train vector representations and to create embeddings which we can leverage in different ways for data integration and discovery.

Humans in the Loop. Considering that data integration cannot be fully automated, we aim at developing a system which can assist and collaborate with human experts to perform large-scale data integration and discovery with minimal human intervention. In particular, users will naturally interact with the system and suggest suitable refinements to resolve uncertainties. For instance, humans can provide existing known relationships, or evaluate the quality of recommended matches detected by the system.

5. REFERENCES

[1] Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: a survey. *VLDBJ*, 24(4):557–581, 2015.

[2] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *TACL*, 5:135–146, 2017.

[3] D. Deng, R. C. Fernandez, Z. Abedjan, et al. The data civilizer system. In *CIDR*, 2017.

[4] A. Doan, P. Domingos, and A. Halevy. Learning to match the schemas of data sources: A multistrategy approach. *Machine Learning*, 50(3):279–301, 2003.

[5] M. Ebraheem, S. Thirumuruganathan, S. Joty, et al. Distributed representations of tuples for entity resolution. *PVLDB*, 11(11):1454–1467, 2018.

[6] R. C. Fernandez, Z. Abedjan, et al. Aurum: A data discovery system. In *ICDE*, pages 1001–1012. IEEE, 2018.

[7] R. C. Fernandez, E. Mansour, A. A. Qahtan, et al. Seeping semantics: Linking datasets using word embeddings for data discovery. In *ICDE*, pages 989–1000. IEEE, 2018.

[8] Y. Gao, S. Huang, and A. Parameswaran. Navigating the data lake with datamaran: Automatically extracting structure from log datasets. In *SIGMOD*, pages 943–958. ACM, 2018.

[9] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, pages 855–864. ACM, 2016.

[10] A. Halevy, F. Korn, N. F. Noy, et al. Goods: Organizing google's datasets. In *SIGMOD*, pages 795–806. ACM, 2016.

[11] Z. S. Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[12] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *ICML*, pages 1188–1196, 2014.

[13] T. Mikolov, I. Sutskever, K. Chen, et al. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[14] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

[15] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDBJ*, 10(4):334–350, 2001.

[16] M. Stonebraker, D. Bruckner, I. F. Ilyas, et al. Data curation at scale: The data tamer system. In *CIDR*, 2013.

[17] M. Zhang, M. Hadjieleftheriou, B. C. Ooi, et al. Automatic discovery of attributes in relational databases. In *SIGMOD*, pages 109–120. ACM, 2011.