Horizon 2020 Framework Programme

Grant Agreement: 732328 – FashionBrain

Document Information

Deliverable number: **D4.2**

Deliverable title: **Demo on text joins**

Deliverable description: **This deliverable present the demo we build for the text join techniques reported in D4.1**

Due date of deliverable: **Jun. 30, 2018**

Actual date of deliverable: **Jun. 30, 2018**

Authors: **Alexander Löser, Torsten Kilias,**

Ying Zhang, Martin Kersten, Richard Koopmanschap

Project partners: **Beuth, UNIFR, USFD, MDBS, Zalando**

Workpackage: **WP4**

Workpackage leader: **Beuth**

Dissemination Level: **Public**

Change Log

Version	Date	Status	Author (Partner)	Description/Approval Level
0.1	May 22, 2018	init	Ying Zhang (MDBS)	Initial version of the deliverable

0.8	June 20, 2018	first draft	Torsten Kilias (Beuth), Ying Zhang (MDBS)	First draft of the document

Table of Contents

1 Introduction

The FashionBrain project targets at consolidating and extending existing European technologies in the area of database management, data mining, machine learning, image processing, information retrieval, and crowdsourcing to strengthen the positions of European (fashion) retailers among their world-wide competitors.

Retailers, such as Zalando SE, often use a relational database management system (RDBMS) to manage their product catalogue, customer data, sales information, etc. At the same time, there is also an abundance of text information available online, e.g. news messages, fashion articles, customers reviews and public fashion discussions. So, to enrich relational data in a (fashion) data warehouse with text data from the web is therefore an important operation for retailers to learn about their users, monitoring trends and predicting new brands. Therefore, in *WP4 "In-Database-Mining and Deep Learning"*, we focus on developing deep learning empowered in-database text mining technology.

In the context of *T4.1 "Text joins between Fashion Entities in a RDBMS and Text"*, we have developed IDEL (In-Database Entity Linking), a novel architecture in which relational data, text data and entity linking are integrated into a single system. First, IDEL leverages two best of breed systems MonetDB[1] and INDREX [1] for relational and text data analysis, respectively. Then, we have developed novel entity linking technology using neural embeddings for IDEL. Finally, all components are integrated into MonetDB to realise an advanced in-database text-join system, in which deep learning techniques are used to achieve better entity linking results.

The results of our work in T4.1 are reported in two deliverables. In D4.1 "Report on text joins" [2], we have reported the design and implementation of IDEL. In *this deliverable D4.2 "Demo on text joins"*, we first give a brief summary of the architecture of IDEL (Section 2), then we present the demo we have built based on IDEL to showcase its usefulness for fashion data (Section 3).

[1] https://www.monetdb.org

2 Summary of IDEL

In this section, we briefly describe the architecture of IDEL and its implementation. More detailed information can be found in [2].

2.1 IDEL Architecture

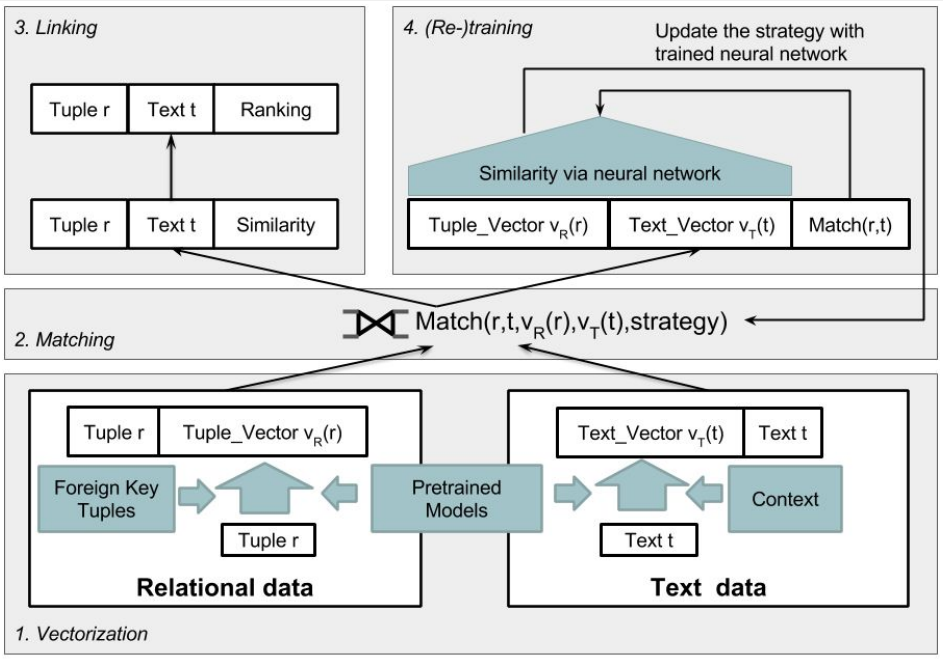

Figure 2.1: Architecture of IDEL. The workflow of the process contains four steps: vectorization, matching, linking and (re-)training

Figure 2.1 depicts the architecture of IDEL. It is designed in such a way that all involved data (i.e. relational data, text data and neural embeddings) is stored in an RDBMS. Relational data is stored according to its schema. Text data is simply stored as a collection of strings (e.g. a table with a single string column). In this way, users can manage and query relational data together with text data and neural embeddings. Our approach for entity linking addresses both mapping directions, i.e. text to relational data and vice versa. The process of entity linking is divided into four major steps:

Step 1: Vectorization

First, we compute the respective vector representations (i.e. `Tuple_Vector` $v_R(r)$ and `Text_Vector` $v_T(t)$) for the two data sets. To create these vectors, we choose SkipThought [5], an existing pre-trained neural network model from the machine learning community that is known to be particularly suitable for this kind of work. Further, we can enrich both vector representations with additional discriminative features we can derive from their respective data sets. For tuple vectors, we can use additional constraints in the relational data, such as foreign keys. For text vectors, we can use context from surrounding sentences.

Step 2: Finding matching candidates

The next step is to find matching candidates for entities in relational data with mentions in text data. Assume a user enters an SQL query such as the following to link relational data (`Building`) and entities mentioned in text data (`EntityMention.Mention`):

```
SELECT e.*, o.*
FROM EntityMention e, Building b
WHERE LINK_CONTAINS(e.Mention, b.name, $Strategy) = TRUE
```

This query joins `EntityMention` with tuples of `Building` and evaluates if the `name` of a `Building` is contained in the entity mentions. In addition to entity mentions and building names, the function `LINK_CONTAINS` takes a third parameter *`$Strategy`* so that different strategies can be passed. Currently, we support both an exact match and a semantic match strategy.

When computing the matches for the first time, there is often little knowledge about the data distribution. Therefore, we bootstrap an initial candidates pool by generating exact matches with a join between words in entity mentions and words in relational tuples describing those entities. The initial matches can be used in later steps, such as linking and retraining. We choose this strategy over using gold standards, because the later requires expensive domain experts for the labeling and hundreds or (preferably) even thousands of matchings.

Step 3: Linking

This step creates linkings of matching entities. We interpret entity linking as a ranking task and assume that an entity mention in the text data is given and we try to find the *k* most likely entities in the relational data (or vice versa). This step uses the matching candidates found in Step 2 and generates a ranking. If the matching function used in Step 2 returns similarity values, this steps will leverage that information to compute a ranking and select the top *N* best matchings for further use; otherwise (e.g. `LINK_CONTAINS`), all pairs of matching candidates are regarded to have equal similarity and hence will be included in the result of the linking.

Step 4: Retraining

An initial matching strategy like the aforementioned bootstrapping is often unable to detect difficult natural language features such as homonyms, hyponyms, synonyms and misspellings. To improve results of the initial matching step, IDEL conducts a retraining step to enhance the neural models of the semantic similarity function. Thereby, the system updates previous models with retrained neural networks and recomputes the matching step. This training - updating - matching circle can be also used to capture changes (i.e. inserts, deletes, updates) in the database. If those changes alter the distributions of the data, the systems triggers the neural network to be retrained with the new data so as to match the new entities reliably while using existing matching models for bootstrapping training data and reducing manual labeling efforts.

2.2 IDEL Implementation

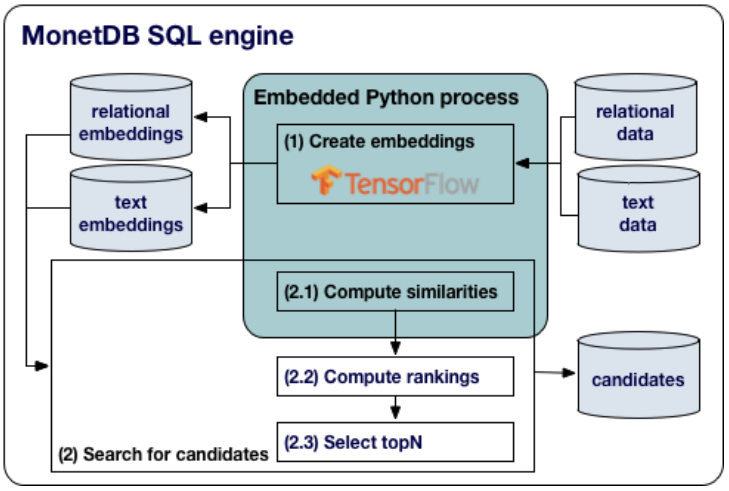

Figure 2.2: implementation architecture of IDEL.

We have integrated the entity linking process into a single RDBMS, MonetDB, as depicted in

Figure 2.2 depicts the implementation architecture of IDEL. Following our design criterion, the whole entity linking process has been implemented in a single RDBMS, MonetDB, an open-source columnar RDBMS optimised for in-memory processing of analytical workloads [6]. As shown in the figure, all data is stored in MonetDB, and all computation are implemented either as SQL queries or by *SQL Python UDFs*.

In recent years, MonetDB has enriched its support for in-database analytics by, among others, introducing MonetDB/Python integration through SQL UDFs [7]. As a result, MonetDB users can specify Python as the implementation language for their SQL UDFs. In principe, any Python libraries accessible by the MonetDB server can be imported. When an SQL Python UDF is called in an SQL query, MonetDB automatically starts a Python subprocess to execute the UDF. MonetDB exchanges data of relational tables between the SQL engine and the embedded Python process by means of NumPy arrays, because they both have the same binary structure. In this way, data transfer and conversion between these two environments are negligible.

In the implementation, we first store relational data in MonetDB according to their schemas and text data in a table with a single string-typed column. Then, we create embedding vectors for both relational and text data using two SQL Python UDFs, one for each input table. This step leverages the deep learning features of TensorFlow[2] to load the pre-trained neural networks and apply them on the input tables. We return embedding vectors as NumPy arrays and store them as BLOBs. Finally, we find matching candidates with the highest similarities among embeddings. We employ nearest neighbor search with Spotify Annoy[3] for a given embedding, compute a ranked list for each entity according to their similarities and finally return top *N* candidates. All steps are implemented in SQL.

A main advantage of this implementation is that changes inside this architecture, such as different embedding models or similarity functions, are transparent to upper layer applications.

[2] https://github.com/tensorflow
[3] https://github.com/spotify/annoy

3 IDEL Demonstration

Text joins can be used either for finding interesting tuples for entity mentions in text or finding interesting mentions in text for a given tuple. We show in our demo these two use cases. The linking from text to tuple is shown by an extension of the web-based editor Tasty (Tag as-you-Type) [3]. Tasty recognizes and links entities while the user is typing. With our extension Tasty now shows relevant information from the database to detected entities. This relevant information supports the writer with the creation of the text. If the writer needs further information about an entity, the demo is able to search for existing relevant text which contains a mention of this entity. Further, the demo enables the user to search a text collections for specific entity.

We integrate IDEL into the backend of Tasty and replace the existing Entity Linking system, so that Tasty now can link against a relational database. Further, we trained the Named Entity Recognizer Texoo [4] on Fashion related Named Entities. So, we can use Texoo and IDEL in combination.

3.1 Demo data set(s)

For our demo we need a structured data with fashion related entities, such as clothing, designer, models. Further we need a text collection which contains mentions of these fashion related entities. For the purpose of the demo we use wikipedia as the source for the entity mentions and wikidata as the source of the structured data. We use anchor links (i.e. links from one Wikipedia page to another Wikipedia page) for the generation of the entity mentions. Further, we transform the information from the less structured Wikidata into relational tables. Table 1 shows some statistics about these tables and their corresponding mentions. Later these tables can be replaced by data from a real-world FDWH.

Entity Type	Tuples	Mentions	Columns
Clothing	1685	56061	8
Designer	1899	18180	6
Model	11215	273130	5

Table 1: Statistics about the dataset derived from Wikipedia and Wikidata.

3.2 Demo scenarios

Our demo shows two main use cases. In the first scenario, we show related entities during text writing. The second scenarios shows related documents for an entity. Further, both scenarios complement each other. For example, if a user writes text and needs additional information about the topic or named entities, she/he can use the tuples from the relational database. However, sometimes the database does not contain the most fresh information. In this case, the user can find more fresh information in related documents.

Scenario I (Tag as-you-Type - Show related Entities to your Text)

In the first scenario we show the interactive editor Tasty with IDEL as backend. While the user writes text, the system annotates named entities in the text and links them to an relational database, such as an FDWH. It then shows additional information for the linked entities to the user which can help him/her to proceed writing.

Figure 3.1 shows a screenshot of the Tasty editor interface. At the left side, the input area with the currently written text is located. In the text fashion related named entities are marked. The user can add new annotations or correct the link of an existing annotation. If the user clicks on a marked entity

mention, the editor shows a selection of alternative entities for the currently linked entity. Further, the user can remove an incorrect annotation. The system can use such corrections to improve the learned model for future training iterations.

At the right side, Tasty shows an short overview of tuples which correspond to the named entities mentioned in the text. This gives the user additional information from the FDWH which can be useful for the further writing process. If the user is interested in more details, he/she can click on "Show full entry" to see the remaining columns of the Tuple (see Figure 3.2). In our demo, the full entry of a tuple can contain text, a number, a boolean or an image. Especially, images are important in the fashion domain, because they give visual impression of the article.

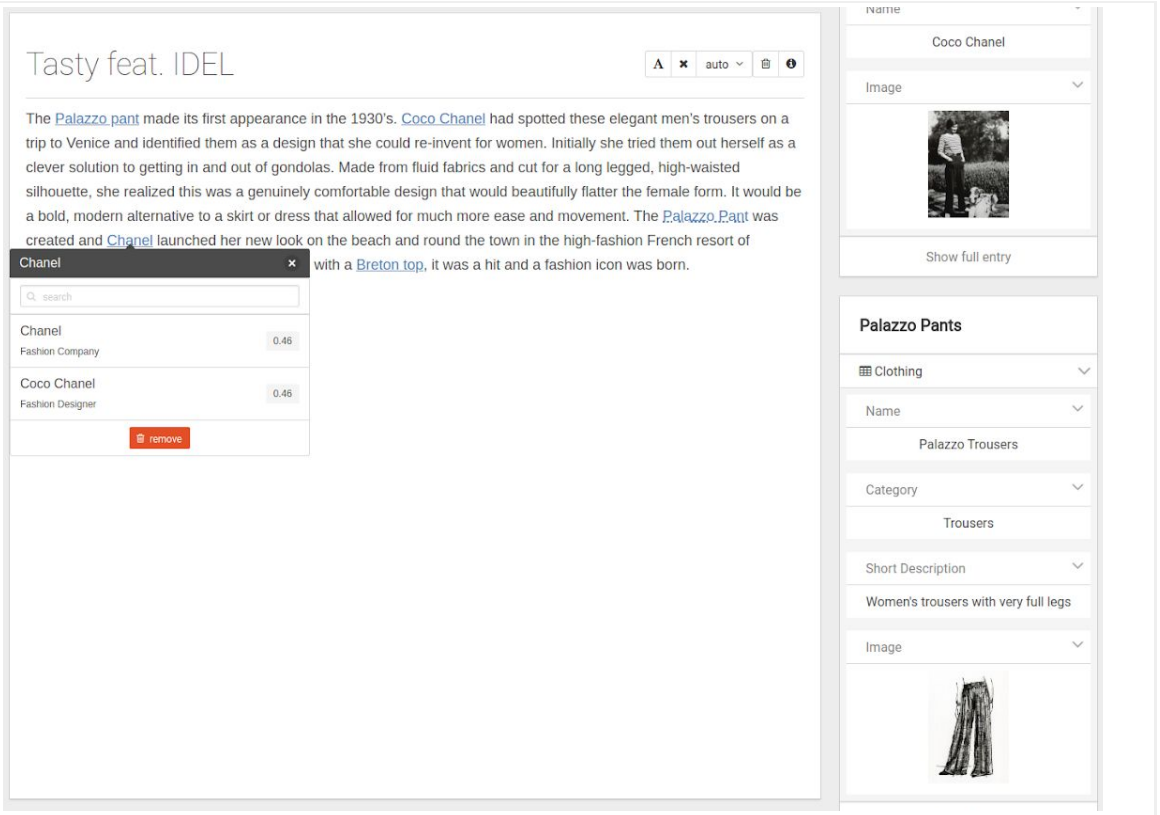

Figure 3.1: Screenshot of the Tasty editor interface

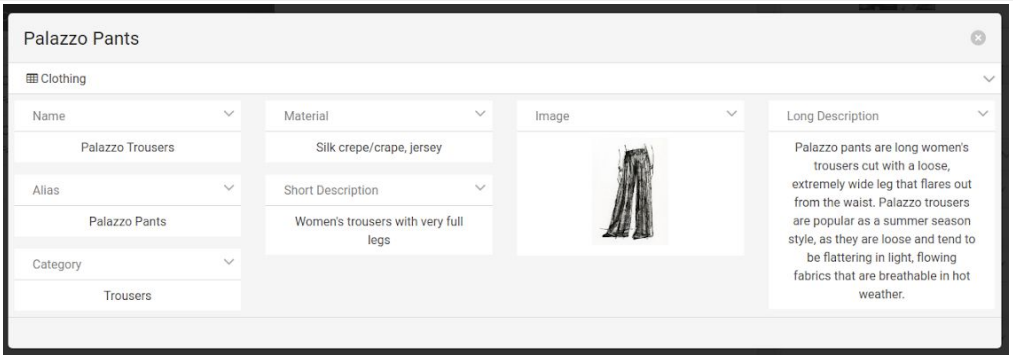

Figure 3.2: Screenshot of "Show full entry"

Scenario II (Searching for related Documents for an Entity)

Our second scenario is searching for related documents for an entity. In this scenario the user selects an entity and the system returns the related documents which contain a mention of this entity. This selection can be done either from a linked entity in the editor or by searching for the entity in the search bar. Figure 3.3 shows a screenshot of the second scenario. At the top the search bar is located, on the left the tuple for the selected entity and in the center the related documents. The related documents can give further and probably more fresh information about the selected entity then the tuple from the database.

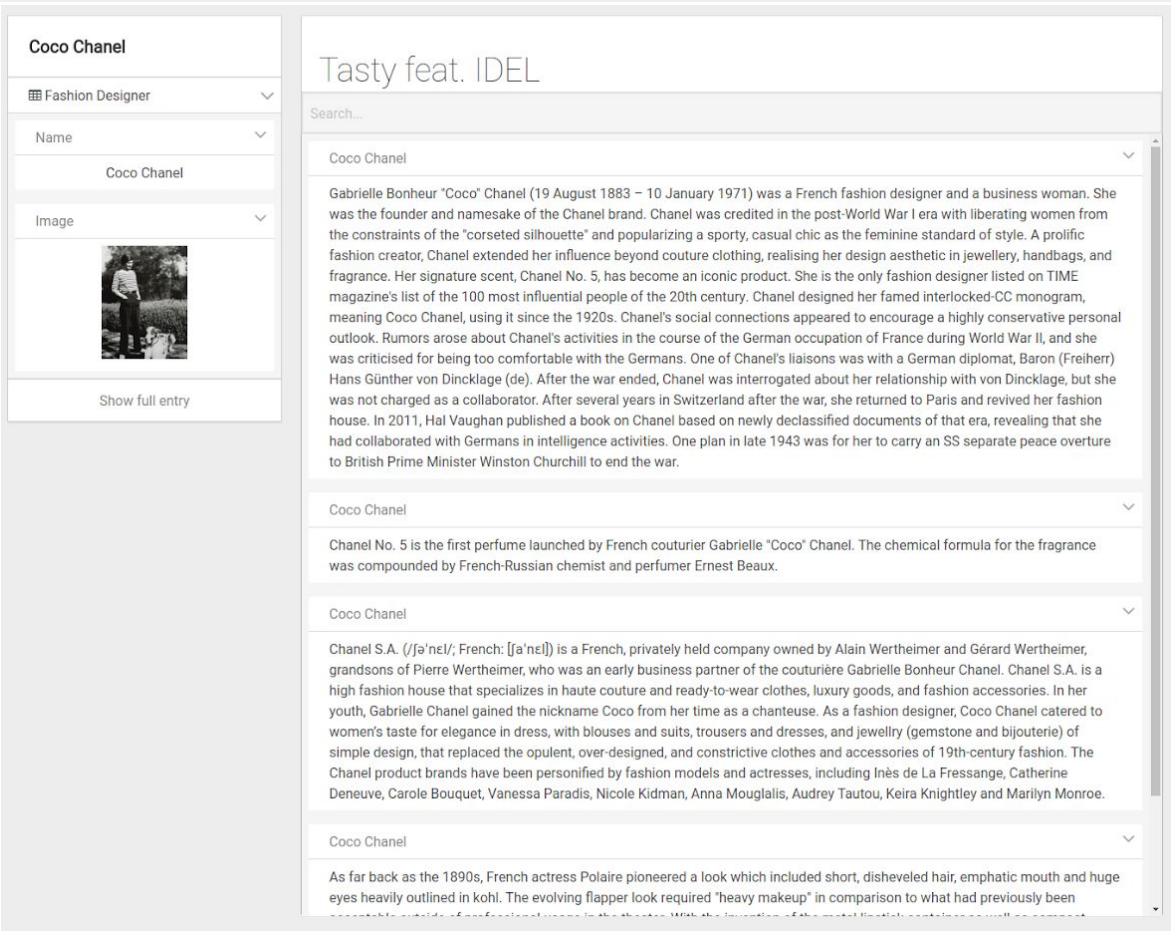

Figure 3.3: Screenshot of searching related documents for an entity

References

[1] Kilias, Torsten, Alexander Löser, and Periklis Andritsos. 2015. “INDREX: In-Database Relation Extraction.” *Information Systems* 53 (October): 124–44.

[2] Torsten Kilias, Alexander Löser, Felix A. Gers, Richard Koopmanschap, Ying Zhang, Martin Kersten. 2018. “IDEL: In-Database Entity Linking with Neural Embeddings”. FashionBrain project deliverable D4.1. Available online from: https://fashionbrain-project.eu/publicdeliverables/

[3] Sebastian Arnold, Robert Dziuba, Alexander Löser. “TASTY: Interactive Entity Linking As-You-Type.” COLING (Demos) 2016: 111-115

[4] Sebastian Arnold, Felix A. Gers, Torsten Kilias, Alexander Löser: “Robust Named Entity Recognition in Idiosyncratic Domains”. CoRR abs/1608.06757 (2016)

[5] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler. 2015. “Skip-thought vectors”. *In Advances in Neural Information Processing Systems*, pages 3276–3284.

[6] P. A. Boncz, M. L. Kersten, and S. Manegold. “Breaking the memory wall in MonetDB”. *Commun. ACM*, 51(12):77–85, 2008.

[7] M. Raasveldt and H. Mühleisen. 2016. “Vectorized UDFs in Column-Stores”. *SSDBM’16*. ACM.