Using Word Embeddings for Visual Data Exploration with Ontodia and Wikidata

Gerhard Wohlgenannt[1], Nikolay Klimov[1], Dmitry Mouromtsev[1], Daniil Razdyakonov[2], Dmitry Pavlov[2], and Yury Emelyanov[2]

[1] Intern. Lab. of Information Science and Semantic Technologies, ITMO University, St. Petersburg, Russia http://en.ifmo.ru/en
[2] Vismart Ltd., St. Petersburg, Russia https://vismart.biz

Abstract. One of the big challenges in Linked Data consumption is to create visual and natural language interfaces to the data usable for non-technical users. Ontodia provides support for diagrammatic data exploration, showcased in this publication in combination with the Wikidata dataset. We present improvements to the natural language interface regarding exploring and querying Linked Data entities. The method uses models of distributional semantics to find and rank entity properties related to user input in Ontodia. Various word embedding types and model settings are evaluated, and the results show that user experience in visual data exploration benefits from the proposed approach.

Keywords: Linked Data querying, word embeddings, Ontodia, Wikidata, natural language interface

1 Introduction

The gigantic data source of Linked Data (LD) is accessible both by machines and humans. Especially for end users, there are high barriers, such as finding relevant datasets, understanding the schema, or being familiar with query languages such as SPARQL [1]. One of the tools that provide an intuitive way to discover LD for non-technical users is Ontodia[3]. Ontodia is an open-source library for OWL and RDF diagramming and visual exploration. In its current version, natural language (NL) search in the properties of given entities will only find properties exactly matching in the its labels. Here, we investigate a method to make the search more flexible and abstracting users from the underlying data schemata by leveraging word embeddings to provide properties which are semantically related to a user query. Using Wikidata[4] as underlying dataset, we aim to i) investigate if word embeddings are useful for the given problem, ii) evaluate which types of pre-trained embedding models, and which parameters, are best suited for the task, and iii) provide a prototype to demonstrate the benefits of the method.

We do not aim at full-fledged question answering over LD with NL to SPARQL transformation, but at improving the search functionality in diagrammatic LD exploration.

[3] http://www.ontodia.org
[4] https://www.wikidata.org

2 Related Work

Query expansion for keyword queries is a classical problem in information retrieval. A traditional way of keyword expansion is the use of dictionaries such as WordNet to find synonyms or hypo- and hypernyms. This method suffers from sparse data regarding Named Entities and missing coverage of specialized domains. In the Semantic Web field, eg. Augenstein et al. [1] propose a method to map keywords to LD resources by finding the properties that are related to semantic similarity between resources. In contrast to our work, which searches in entity properties, Augenstein et al. [1] focus primarily on finding resources (entities). Freitas et al. [3] propose a complex system for querying heterogeneous, and distributed datasets, which abstracts users from the underlying data schemata. The system combines entity search, a Wikipedia-based semantic relatedness measure and spreading activation to answer NL queries.

Challenges and future directions in Question Answering on LD are presented in Shekarpour et al. [8]. The application of word embeddings and deep learning is listed prominently among the promising techniques for future investigation. In line with this recommendation, we apply distributional semantics for the natural language query interface of Ontodia. In general, word embeddings transform the vocabulary of a given corpus into a continuous low-dimensional vector space representation. They have been successfully applied, for example, for word similarity computations, but also more complex natural language tasks [4].

3 System Description

The work presented in this paper extends Ontodia with improved search capabilities. As mentioned, Ontodia is an open-source tool[5] for simple OWL and RDF visual data exploration. Ontodia is often integrated with metaphactory[6] as a semantic platform backend. In a typical data exploration scenario, the user starts querying the dataset at the system entry point[7]. At search result, the user can switch to using Ontodia to explore the data space. In the current version, search in the connections of an entity only finds literal matches of the search term in the property labels. This limits the ease-of-use with unfamiliar datasets. E.g, when looking for family relations of entity *Van Gogh*, the system will not find any matching properties due to missing exact lexical matches, see Figure 1.

The prototype presented here makes use of a) aliases for property labels defined in Wikidata, and it applies distributional semantics in the form of word embeddings to find suitable properties related to a user query. Figure 2 shows the results using the new search functionality, which are a combination of: (i) exact matches of the input term in the property labels, (ii) exact matches in property aliases, and (iii) related properties according to the word embedding model used, ordered descendingly by semantic similarity.

[5] `https://github.com/ontodia-org/ontodia`
[6] `http://www.metaphacts.com/product`
[7] `https://wikidata.metaphacts.com/resource/Start`

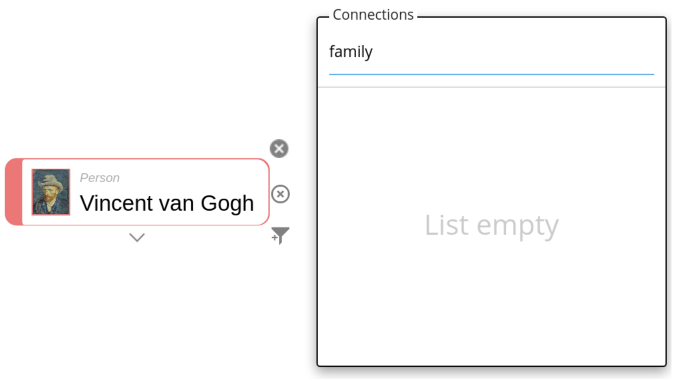
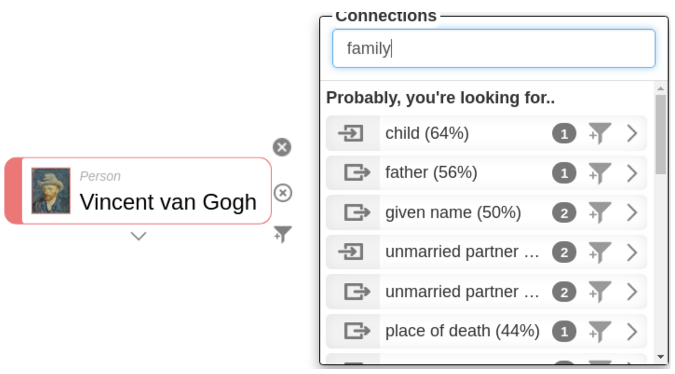


Fig. 1. Searching for "family" relations of entity *Van Gogh* in the original system.



Probably, you're looking for..

- child (64%) 1
- father (56%) 1
- given name (50%) 2
- unmarried partner ... 2
- unmarried partner ... 2
- place of death (44%) 1

Fig. 2. Searching for "family" relations of entity *Van Gogh* in the new prototype.

The updated search interface also allows for a new way of data exploration, where the user is interested in a certain topic, for example *family* or *politics*, and can then explore all entity properties (connections) related to the topic.
The prototype described here is available at:
`http://ontodia-prop-suggest.apps.vismart.biz/wikidata.html`.

3.1 The Method

A central ingredient to the method is the word embedding model. The models were trained on a Wikipedia corpus – and in some cases additional textual sources – and contain continuous vector space representations of the words from the corpora which capture the distributional semantics of the words.

First, the Wikidata properties need to be added to the model vector space. For every property we split the property label (`rdf:label`) into a list of words, and remove stopwords. The vector representation of a property is created as the vectorial sum of the words. A variant of the system also includes the words from the property descriptions to create the property vectors. At runtime, the same process is applied to the natural language user query provided in the search box. The query is split into single words, stopwords are removed, and the vectorial representation is the sum of the query word vectors. Finally, the system ranks the properties by cosine similarity between the query vector and all the property vectors to find the most relevant properties.

The method is simple and computationally efficient. In this publication, the focus is on the evaluation of the method, and especially on comparing the performance of various types of word embeddings.

3.2 Implementation

The presented method and the accompanying code was implemented in Python and can be found on GitHub[8]. The main modules include a preprocessing phase, where the vectors for the Wikidata properties are constructed and persisted, the module to rank properties according to user input, and the tools for the evaluation of the system. For integration with Ontodia, we created a webservice that takes the user input in JSON format, computes the property rankings, and returns them in JSON format to Ontodia for display to the user.

4 Evaluation

First, this section describes aspects of evaluation setup like the Wikidata dataset, the gold standard data used, system settings and the word embedding models. Then, a detailed presentation of the evaluation results, including a discussion of aspects like dataset quality and result interpretation, follow.

4.1 Evaluation Setup

Wikidata Dataset Wikidata is an open knowledge base, which can be exported and interlinked with other datasets on the Linked Data web. Wikidata is the central data storage for projects like Wikipedia.The dataset currently includes around 28 million items, and, more relevant for this work, there are 3323 properties defined to describe and connect the entities. The properties have labels for various languages, and aliases (called "also known as") for many of the labels. We focus on English language labels, for which currently 4603 aliases are defined. Additionally, properties usually have a short textual description, which we also use in our method to create property representations.

Gold Standard Dataset The aliases manually defined in Wikidata are an obvious source to be used as a gold standard dataset to evaluate our method. For this purpose, any of the 4603 English language aliases is used as an query term, and the system suggests a ranking of properties similar to the term. 1736 of 3323 properties actually have aliases defined.

[8] `https://github.com/gwohlgen/ontodia_search_properties`

Despite the varying quality of aliases (details in the *Discussion section*), we decided to use them as a gold standard dataset. Eventually the proposed method can even be applied to help detect questionable alias definitions in the future.

System Settings In the evaluations, we experimented with various system settings and word embedding models. The types of word embedding models are described below, the most important system settings include:

- **Use description text (Boolean):** For creating the representations of properties in vector space, we compared the results of using only the words from the property labels versus words from property labels and description texts.
- **Dimensions of vector model:** Some predefined vector models are available with different numbers of vector dimensions (for example 50 vs. 100 vs. 300 dimensions). A lower number of dimensions makes the model more computationally efficient, but it may loose semantic nuances.
- **Number of words in the model:** In the pre-trained models the word vectors are ordered descendingly by word frequency in the training corpus. Big models with hundreds of thousands of vectors occupy a lot of memory and take a long time load. Therefore, we compared the performance of models with 300.000 words with smaller models with the 10.000 most frequent words.

Word Embeddings One of the main goals was to evaluate which of the pre-trained word embedding models is best suited for the task at hand. The pre-trained models available are not trained on exactly the same corpus, but all include English Wikipedia. The following word embedding types were evaluated:

- **fastText**: FastText [2] is an extension of the original Word2vec [5] model which uses sub-word information. Words are represented as bag of character n-grams. FastText generates better word embeddings for rare words, and takes morphological information into account. Here, we applied a model trained on Wikipedia 2016[9]. Two variants were compared, a model with 300.000 words, and a small model with only the 10.000 most frequent words.
- **GloVe**: GloVe[6] factors the logarithm of the co-occurrence matrix that reflects the position of the context words in the word window. We used a model pre-trained on a Wikipedia 2014 and Gigaword 5 corpus (6B tokens)[10]. Variants include combinations of models with 300, 100 or 50 dimensions, and 300.000 versus only 10.000 word vectors.
- **LexVec**: LexVec [7] is a word embedding method which factorizes PPMI matrices and combines characteristics of techniques like Word2vec and GloVe. LexVec performs well on word similarity and semantic analogy tasks, but struggles on syntactic analogies. The model used was trained on a 7B token corpus of English Wikipedia 2015 and NewsCrawl[11]. Again, we evaluated variants of 300.000 versus 10.000 word vectors.

[9] `https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md`
[10] `https://github.com/stanfordnlp/GloVe`
[11] `https://github.com/alexandres/lexvec`

4.2 Evaluation Results

In the main evaluation which aims to judge the suitability of various word embedding types we experiment with different models and settings. As stated, the task is as follows: for any of the aliases defined for Wikidata properties, we create a ranking of related properties. The word vectors of the alias words are compared to the vectors representing the properties. Every alias is compared to all 3323 properties, which is much harder than the real-world task of searching only in the properties of a given entity. The later task is evaluated in the next section.

Table 1 presents an overview of the results. Column one states the embedding model type and the settings, namely the model size (either 300.000 or 10.000 words), and the dimensions of the vectors. The metrics *Top-N* reflect the ratio of system suggestions, where the correct property is in the *Top-N* of the generated ranking. MRR is the well-known *mean reciprocal rank*. The lower part of the table includes some results for models which only use the words from the property label to create the property vectors, but not from the description text (*WO-D*).

	Top 1	**Top 3**	**Top 10**	**MRR**
fastText 300.000 / 300d	**38.12%**	**55.13%**	**70.22%**	**0.493**
fastText 10.000 / 300d	31.49%	48.59%	66.29%	0.432
GloVe 300.000 / 300d	36.33%	52.82%	66.55%	0.469
GloVe 10.000 / 300d	30.54%	45.90%	61.78%	0.411
GloVe 300.000 / 100d	33.24%	47.85%	61.94%	0.429
GloVe 10.000 / 100d	28.39%	43.21%	58.97%	0.386
GloVe 300.000 / 50d	27.97%	41.88%	56.58%	0.376
GloVe 10.000 / 50d	24.42%	38.51%	54.24%	0.344
LexVec 300.000 / 300d	37.21%	53.45%	67.99%	0.479
LexVec 10.000 / 300d	30.59%	46.03%	62.55%	0.411
fastText WO-D 300.000 / 300d	36.10%	51.94%	65.25%	0.464
fastText WO-D 10.000 / 300d	29.99%	46.83%	60.74%	0.407
GloVe WO-D 300.000 / 300d	34.21%	48.88%	60.76%	0.437

Table 1. Evaluation of word embedding models and settings on aligning Wikidata aliases to the corresponding property.

The fastText model with 300.000 word vectors and 300 vector dimensions performs best over all metrics. We also experimented with a bigger fastText model with around 2.5m word vectors, but those additional rare words just increased memory consumption, the performance stayed almost the same. On the other hand, it is evident that reducing the model size to 10.000 words affects performance negatively. Over all model types reducing model size from 300.000 to 10.000 words led to a sharp drop in accuracy. Regarding model types, fastText is best suited for the task, followed by LexVec, and lastly GloVe. As seen in the last part of the table, using the words from the description text to represent property vectors is helpful. Finally, using fine grained word representations with larger vectors (50 versus 100 versus 300 dimensions) has a strong positive effect.

Property Search for Single Entities In the evaluations above, we measure the accuracy for matching aliases against all the 3323 properties in Wikidata. However, in an interactive scenario of visual data exploration with Ontodia, the user query is typically restricted to the properties defined for a specific entity. This scenario was simulated and evaluated by randomly choosing 1150 entities from the Wikidata dataset, and performing the evaluation with the their properties and aliases.In total, about 85% of the properties had one or more aliases defined, with an average of 5.9 aliases per property. Table 2 presents the evaluation using the fastText and LexVec models on the task of finding the corresponding entity property for all aliases defined for an entity.

	Top 1	**Top 3**	**Top 10**	**MRR**
fastText	68.63%	84.75%	94.59%	0.78
LexVec	62.08%	80.93%	93.18%	0.73

Table 2. Evaluation of system accuracy for matching aliases with properties of randomly-picked entities.

Again, fastText outperforms the LexVec embeddings. When ranking the entity properties for the alias term by similarity, in over 70% of cases the first ranked property is correct with respect to the gold standard. For the Top-3 results, the number is 87.49%, and the MRR is 0.80. The results make us confident that the new search feature has a very positive impact on user experience. The runtime of a query is typically under $10ms$ – well-suited for interactive systems.

4.3 Discussion

Dataset Quality During the evaluation and the inspection of the results we found various issues with Wikidata dataset quality, which (i) explain part of the misclassification of the method, and (ii) provide hints on improving dataset quality, esp. the quality of aliases. First of all, in 14 cases the alias was exactly the same term as the property label. More interestingly, many aliases are not proper synonyms. For example, property *P582* with label "end time", has alias such as "divorced", or simply "to". Or, *P150* with label "contains administrative territorial entity", has aliases such as "divides into", "contains", "has villages" – some of which make it hard for the system to link to the correct property.

4.4 System Performance

The experiments summarized in Table 1 indicate that the fastText algorithm is best suited for the task, followed by LexVec. System configuration, especially the model vocabulary size and the number of vector dimensions are crucial for system performance, and should only be compromised if decreasing memory footprint is inevitable. Furthermore, including the property descriptions in the vector provides better property representations. In our real-world use case (Section 4.2) the method demonstrates sufficient performance to improve user experience.

Regarding computational performance, using Python and the gensim library, the fastText model with 300.000 vectors and 300 dimensions consumes ca. 650M of memory, a 10.000 words model requires 130M. The runtime for a query against all 3323 properties is around 300ms, for the interactive use-case query time is usually below $10ms$.

5 Conclusions

In this publication we present a method for simple and powerful search in entity properties of Linked Data using natural language. A prototype of the method is integrated into the Ontodia tool using Wikidata as data source. The method applies models of distributed semantics to find properties related to user input. The contributions include (i) the presentation of a method for searching in Linked Data which applies word embeddings to the given task in an efficient way, (ii) an extensive evaluation of various types of word embedding models and parameters such as model size and dimensionality against a gold standard, (iii) the provision of the implementation and an online prototype. In future work we will apply the presented approach to other datasets, and investigate the integration with more powerful question answering for Linked Data techniques.

6 Acknowledgments

This work was supported by the Government of the Russian Federation (Grant 074-U01) through the ITMO Fellowship and Professorship Program.

References

1. Augenstein, I., Gentile, A.L., Norton, B., Zhang, Z., Ciravegna, F.: Mapping keywords to linked data resources for automatic query expansion. In: Cimiano, P.e.a. (ed.) ESWC 2013. pp. 101–112. Springer LNCS, Berlin, Heidelberg (2013)
2. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. arXiv preprint arXiv:1607.04606 (2016)
3. Freitas, A., Oliveira, J.G., O'Riain, S., da Silva, J.C., Curry, E.: Querying linked data graphs using semantic relatedness: A vocabulary independent approach. Data & Knowledge Engineering 88, 126 – 141 (2013)
4. Ghannay, S., Favre, B., Estve, Y., Camelin, N.: Word embedding evaluation and combination. In: Calzolari, N., al. (eds.) LREC 2016. ELRA, Paris, France (2016)
5. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
6. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: EMNLP. pp. 1532–1543 (2014)
7. Salle, A., Idiart, M., Villavicencio, A.: Enhancing the lexvec distributed word representation model using positional contexts and external memory. CoRR abs/1606.01283 (2016), http://arxiv.org/abs/1606.01283
8. Shekarpour, S., Lukovnikov, D., Kumar, A.J., Endris, K.M., Singh, K., Thakkar, H., Lange, C.: Question answering on linked data: Challenges and future directions. CoRR abs/1601.03541 (2016), http://arxiv.org/abs/1601.03541