TECHNISCHE UNIVERSITÄT DRESDEN

FACULTY OF COMPUTER SCIENCE
INSTITUTE OF ARTIFICIAL INTELLIGENCE
CHAIR OF COMPUTATIONAL LOGIC

Master of Science

Project Report

Predicting Relation Targets of DBpedia Properties Using Vector Representations from Word2vec

Happy Rani Das

Supervisor: Dr. Dagmar Gromann

Dresden, October 1, 2018

Abstract

Nowadays, the word2vec method is very popular in the field of machine learning to train vector representations of words, called word-embeddings. Vector representations of words have obtained enormous attention in natural language processing and information retrieval for word similarity detection, analogical reasoning and so on. They have been applied to predict relation targets and the word analogy task; however, only one pair of source and target has been used so far. Combining more than one pair of source and target in word-embeddings should theoretically strengthen the representative characteristic of the same category in a relation. This project's aim is to increase the accuracy of analogical reasoning by aggregating semantic information of DBpedia data stored in word embeddings to predict relation targets. Therefore, we introduced two super vectors which aggregate several vectors in the analogy task and can substantially increase the accuracy of predicting relation targets. The following four relations, namely capital-country, currency-country, person-party and company-headquarter, are considered from DBpedia for this study to evaluate the system's performance. With the increasing number of vectors, the performance has increased for all of the above-described relations and the biggest improvement has been achieved for the company-headquarter relation.

Contents

List of Figures

List of Tables

1 Introduction

Word2vec is a proficient method in natural language modeling to compute vector representations of words. It takes text corpus as input and produces a set of vectors as a result. At the beginning, word2vec build a vocabulary from the large text corpus and then generates a group of vectors. There are two ways to represent word2vec model architecture [Mikolov et al. 2013].

1. Continuous bag-of-words (predict a missing word from the given window of context words or word sequence) and

2. Skip-gram (predict the neighboring window of target context by using a word).

Nowadays, the Continuous bag-of-words (CBOW) and continuous skip-gram model architecture are very popular in machine learning, natural language processing and advance research areas.

Equivalent meaning of words will have analogous vectors. On the contrary, words that do not have similar meaning will have vectors that are further away from each other in vector space. It is quite surprising that word vectors follow the analogy rule. For instance, presume the analogy "man is to woman as king is to queen". It gives the following result-

$$v_{king} - v_{man} + v_{woman} = v_{queen} \tag{1.1}$$

where v_{king}, v_{man}, v_{woman} and v_{queen} are the word vectors for the words king, man, woman and queen respectively.

Word2vec has been applied in many areas with the objective of generating or extracting information to solve a specific problem from the specific knowledge base. Different types of knowledge bases are available and DBpedia is one of them. DBpedia ("DB" stand for "database") is a crowd-sourced community effort to extract structured information from

Wikipedia and to make this information available on the web. It has been used as a dataset for diverse purposes[1].

The intention of this project is to introduce a technique that can be used for learning DBpedia data and to find out corresponding relation targets of DBpedia data stored in word embeddings. If the user has two sentences like-

1. Berlin is the capital of Germany.

2. Paris is the capital of France.

The results of the pre-trained embeddings which have been trained with the word2vec method, yield words of similar meaning. Suppose, if we calculate the cosine similarity between vectors with (source:Berlin,target:Germany) and (source:Paris,target:France) this will eventually give insight the model to understand that both vector pairs are connected to capital. Thus they are closely related in the pre-trained embeddings that have been trained with the word2vec method.

The main goal of this project is to increase the accuracy of the analogical task by accumulating semantic information stored in word embeddings to predict relation targets of DBpedia data. Therefore, we implemented two super vectors which aggregate several vectors in the analogy task and can considerably increase the accuracy of predicting relation targets. Increasing number of vectors should work better than single vector because vector addition of same relation type should strengthen the representative characteristic. As an example, if one pair of source and target such as man and woman in Equation 1.1 for the category male and female in relation gender provide a good basis, then theoretically combining more than one pair of source and target in a relation should improve the results. For instance, man, boy, husband, father etc. should provide stronger representation for the characteristic “male” than just man.

Vector representations of words have not been applied for DBpedia data to aggregate semantic information stored in word embeddings. In this project, we introduced a technique to construct super vectors for information accumulation.

[1] http://wiki.dbpedia.org/about

The prediction of relation targets from DBpedia properties using vector representations is developed in Python. This project's target is to take DBpedia data as input and returns nearest words or relation targets as output.

The rest of this report is structured as follows. An overview of preliminaries and related work is discussed in Chapter 2. In the following Chapter, the methodology of the project is described. The results of the project is summarized in Chapter 4 and they are discussed in Chapter 5 subsequently. Finally, the report is concluded with Chapter 6 along with future work.

2 Preliminaries and Related Work

2.1 Preliminaries

2.1.1 SPARQL Query Language

SPARQL is a semantic query language to query RDF graph. It is pronounced as "sparkle", a recursive acronym stands for SPARQL protocol and capable to retrieve and manipulate information stored in Resource Description Framework (RDF) format. Different types of output format are available for SPARQL such as result sets, JSON, RDF/XML, CSV etc. SPARQL is mainly based on the basic graph/ triple pattern matching stored in the RDF graph where it tries to find out the set of triples from the RDF graph [Morsey et al. 2012]. The following example shows how a SPARQL query looks like.

Example: A SPARQL query-

```
PREFIX dbo: <http://dbpedia.org/ontology/>

PREFIX dbr: <http://dbpedia.org/resource/>

PREFIX s: <http://schema.org/>

SELECT * WHERE {

?Hotel a s:Hotel.

?Hotel dbo:location dbr:Dresden.

}
```

The above query is the combination of prefixes and triples. Prefixes are the shorthand for long URIs. This SPARQL query returns all the name of hotels in Dresden as a result when it is executed against DBpedia. For example, Dresden has only two hotels under the dbo:location and they are-

1. "http://dbpedia.org/resource/Taschenbergpalais" and
2. "http://dbpedia.org/resource/Swissôtel_Dresden_Am_Schloss"

2.1.2 Resource Description Framework (RDF)

Resource Description Framework (RDF) is a general-purpose language and a standard model for data interchange on the Semantic Web [Lassila, Swick, et al. 1998]. It has URLs, URIs and IRIs to identify resources uniquely on the web. As an example, http://dbpedia.org/resource/Donald_Trump is globally unique. RDF is specified by W3C recommendation where a simple syntax for RDF is Turtle (Terse RDF Triple Language). Moreover, RDF is the building block of the Semantic Web, made up of triple of the form where a triple consists of subject(resource or blank node), predicate(resource) and object(resource, literal or blank node). An example of RDF triple is-

dbr:Barack_Obama dbp:spouse "Michelle_Obama"

where dbr:Barack_Obama is the subject, dbp:spouse is the predicate/property and Michelle_Obama is the object.

2.1.3 Word Embeddings

Word embeddings are central to natural language processing and efficient method to capture inner words semantics [Levy and Goldberg 2014]. A word embedding is a vector representation of a word where word from a vocabulary is mapped to a vector of real number. Vector representation of word plays an increasingly essential role in predicting missing information by capturing semantic and syntactic information of words. Moreover, this representation can be useful in many areas such as question answering, information retrieval, text classification, text summarization and so on [Teofili 2017]. To get vector representation of a word we used word vectors that have been pre-trained on the Google News corpus using the word2vec method [Mikolov et al. 2013]. Work2vec takes text corpus as input and produces a set of vectors as output. Other methods like GloVe is also available to retrieve vector representations of words. It is a log-bilinear regression model for the unsupervised learning method to obtain vector representation of a word [Pennington et al. 2014].

2.2 Related Work

Vector representations of words have gained enormous attention in the field of machine learning. Several techniques are introduced to get word vector. Word2vec and GloVe have achieved tremendous popularity to predict semantic similarity.

Chen et al. [2017] proposed a method which is useful to detect similarity in a syntactic way for a set of objects and their relationship. One pair of objects can be similar to another pair of objects if they are connected by the similar relation, e.g., object pairs (boy, girl) and (man, woman) are similar because they are connected by the relation gender (Figure 2.1). This was done by Parallelogram model of analogy [2017]. This model represents object oriented data which is effective to presume relationship between objects (see Figure 2.1), where objects are represented as points and relation between objects are represented by the difference between two vectors (i. e. vector1 - vector2). Based on the parallelogram model, two word pairs (s_1, t_1) and (s_2, t_2) are relationally similar if the difference between two vectors ($vect_1 - vecs_1$ and $vect_2 - vecs_2$) are similar. Different types of methods are available to measure the difference between two vectors and cosine similarity is one of them. Cosine similarity is a measure of two non-zero vectors that computes the cosine of the angle between them[1]. Suppose, *a*= $vect_1 - vecs_1$ and *b* = $vect_2 - vecs_2$. The calculation of the cosine similarity is-

$$cos(a, b) = \frac{a \cdot b}{||a|| ||b||} \tag{2.1}$$

[1] https://en.wikipedia.org/wiki/Cosine_similarity

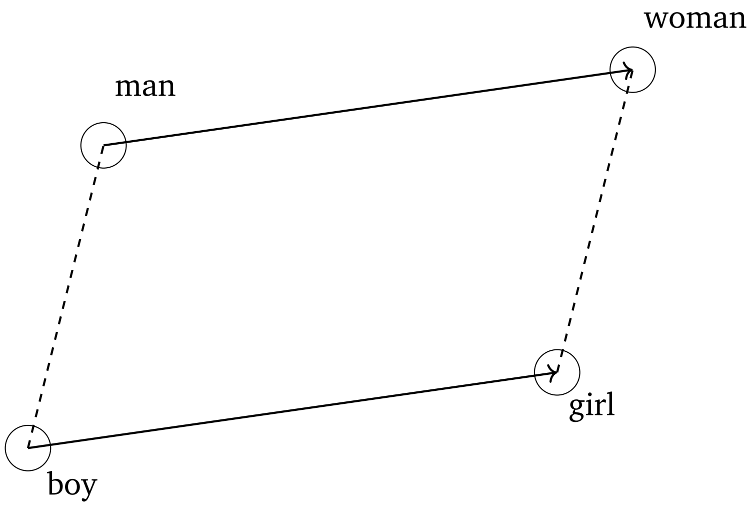

Figure 2.1: The parallelogram model for the analogy boy : girl :: man : ?

Yoshua Bengio et al. [2003] introduced a technique for learning the distributed representation of words which allows to make semantically new sentences from each training sentence. In their approach, they represented word as a distributed feature vector and a new sentence was possible to make from the training data set. For example, the sentence " A tiger is running in the jungle" helps to make another sentence "The fox was walking in a forest".

Andriy Mnih and Koray Kavukcuoglu [2013] proposed an approach to learning word embeddings based on training lightweight language models. In their paper, they focused on the analogy-based questions sets which had the form “x is to y as z is to ?”, defined as x : $y \rightarrow z$: ? . The idea was if x is related to y, then in the same way z is related to what? Their target was how efficiently it could detect the fourth word and they proved their approach produced better performance.

Zhiwei Chen et al. [2017] proposed a methodology to deduce the semantic relations in word embeddings from WordNet and Unified Medical Language System (UMLS). They trained multiple word embeddings from Wikipedia. In the field of text mining and Natural Language Processing (NLP), word embeddings have been exhibited efficiently for capturing syntactic and semantic relations in the past few years. To get the word embeddings, they used three tools; word2vec, dependency-based word embeddings, and GloVe. The nine semantic relations synonym, antonym, hypernym, hyponym, holonym, meronym, sibling, derivationally related forms and pertainym are explored by them.

WordNet unites nouns, verbs, adjectives, and adverbs into sets of cognitive synsets. Synsets are the combination of lexical and semantic relations. Unified Medical Language System is a synopsis of numerous controlled vocabularies in the biomedical sciences. They constructed a standard database with WordNet and UMLS to evaluate the performance of nine semantic relations. Pertainym relation acquired the maximum retrieved ratio to find nearest neighbors. They received better performance with Word2Vec than GloVe for almost all of the semantic relations. On the other hand, dependency-based word embeddings obtained poor performance than Word2Vec and GloVe for the relations pertainym, derivation, meronym and holonym.

The four linguistic relations inflectional, derivational, lexicographic and encyclopedic semantics were introduced by Gladkova et al. [2016]. An analogy is a successful word to detect diverse types of semantic similarity. Their target was to know which types of analogies will be able to work perfectly and which are not. Therefore, they tested 99,200 questions in 40 relations. For word embeddings, they used GloVe and Singular Value Decomposition (SVD) based method. In their experiments, some relation gave excellent accuracy, some didn't and the accuracy varied between around 10% to about 98%. On the one hand, they received around 98% accuracy for the relation capital-country. On the other hand, the accuracy was around 10% for the relation meronyms–member. This proved that a model is more effective with some categories than others. The reason is that some relations could not capture nicely with word embeddings or vector offset. They had shown that the accuracy was also varied with window-size, word-category and vector dimensionality. GloVe gave better performance than SVD. Out of 40 relations, 13 relations achieved more than 40% accuracy for GloVe and SVD acquired more than 40% accuracy only for six relations.

3 Methodology

Figure 3.1 shows the flowchart diagram of this project. DBpedia data was considered as input and the repository of word-embeddings pre-trained on the Google News corpus with the word2vec method was used to learn the vector representation of the given DBpedia data. Thereafter we tried to find out the nearest word based on the given data. After that, relation targets were predicted for the given DBpedia data, e.g., if relation source s_1 is related to relation target t_1 in some way, then relation source s_2 is related to relation target t_2? with the same process. For instance, if "Donald Trump" is to "Republican" as "Barack Obama" is to what? And the predicted relation target is "Democratic".

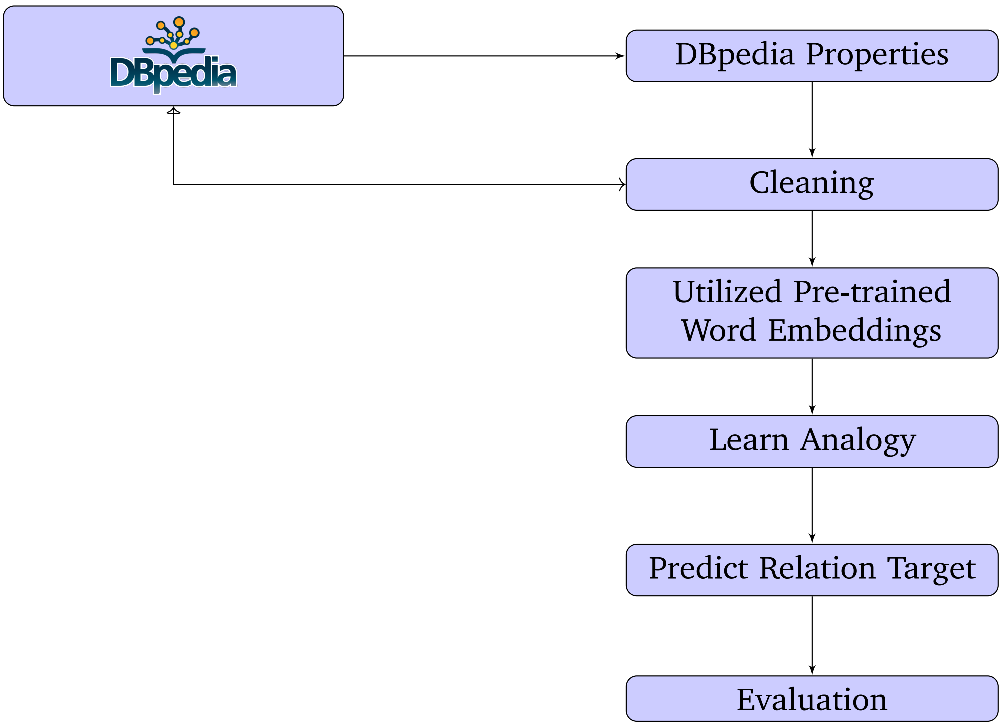

Figure 3.1: The general scheme of predicting relation targets of DBpedia properties using vector representations from word2vec

3.1 Query DBpedia Properties for Relations

DBpedia has huge amounts of data. It is the combination of resources, ontologies, properties and many more. In the RDF preliminaries, we have seen predicate represent property or relation for RDF triple. Property is used to link entities together. To extract all of the resources for specified properties from DBpedia we used SPARQL query language. Four SPARQL queries are considered in this project to evaluate the performance of the system.

Query 1:

The following SPARQL query is executed against DBpedia for the properties 'country' and 'capital' in order to get all of the country and capital name list-

```
SELECT DISTINCT ?country ?capital

WHERE

 {

 ?city rdf:type dbo:City;

 rdfs:label ?label;

 dbo:country ?country.

 ?country dbo:capital ?capital.

 } order by ?country
```

Table 3.1 displayed the result of the query 1.

country	capital
http://dbpedia.org/resource/Afghanistan	http://dbpedia.org/resource/Kabul
http://dbpedia.org/resource/Algeria	http://dbpedia.org/resource/Algiers
http://dbpedia.org/resource/Angola	http://dbpedia.org/resource/Luanda
http://dbpedia.org/resource/Argentina	http://dbpedia.org/resource/Buenos_Aires
http://dbpedia.org/resource/Armenia	http://dbpedia.org/resource/Yerevan
http://dbpedia.org/resource/Australia	http://dbpedia.org/resource/Canberra
http://dbpedia.org/resource/Austria	http://dbpedia.org/resource/Vienna
http://dbpedia.org/resource/Azerbaijan	http://dbpedia.org/resource/Baku
http://dbpedia.org/resource/Bahrain	http://dbpedia.org/resource/Manama
http://dbpedia.org/resource/Bangladesh	http://dbpedia.org/resource/Dhaka
http://dbpedia.org/resource/Barbados	http://dbpedia.org/resource/Bridgetown
http://dbpedia.org/resource/Belarus	http://dbpedia.org/resource/Minsk
http://dbpedia.org/resource/Belgium	http://dbpedia.org/resource/City_of_Brussels
http://dbpedia.org/resource/Belize	http://dbpedia.org/resource/Belmopan
http://dbpedia.org/resource/Benin	http://dbpedia.org/resource/Porto-Novo
http://dbpedia.org/resource/Bolivia	http://dbpedia.org/resource/Sucre
http://dbpedia.org/resource/Bosnia_and_Herzegovina	http://dbpedia.org/resource/Sarajevo
http://dbpedia.org/resource/Brazil	http://dbpedia.org/resource/Brasília
http://dbpedia.org/resource/Bulgaria	http://dbpedia.org/resource/Sofia
http://dbpedia.org/resource/Burkina_Faso	http://dbpedia.org/resource/Ouagadougou
http://dbpedia.org/resource/Burundi	http://dbpedia.org/resource/Bujumbura
http://dbpedia.org/resource/Cambodia	http://dbpedia.org/resource/Phnom_Penh
http://dbpedia.org/resource/Cameroon	http://dbpedia.org/resource/Yaoundé

Table 3.1: Output for the input properties country and capital.

Query: 2

The subsequent query is applied for the properties ‘country’ and ‘currency’ and it returns all of the results against those properties-

```
SELECT DISTINCT ?country ?currency

WHERE {

?city rdf:type dbo:City;

rdfs:label ?label;

dbo:country ?country.
```

```
?country dbo:currency ?currency.

} order by ?country
```

Query: 3

Here is a query which returns all of the persons and their corresponding party name under the properties ‘person’ and ‘party’-

```
SELECT DISTINCT ?person ?party WHERE {

?city rdf:type dbo:City;

?person rdf:type dbo:Person.

?person dbo:party ?party.

} order by ?person
```

For SPARQL, if the results are within 40,000 it is possible to display them on one page once at a time otherwise not; because the DBpedia SPARQL endpoint is configured in the following way-

$$\text{MaxSortedTopRows} = 40{,}000$$

There are huge amounts of data in DBpedia under dbo:Person and dbo:party. In order to retrieve the rest of the data, we used offset which allows getting the next results from the offset index. Therefore, we have considered the following query and appended the results with the previous results.

```
SELECT DISTINCT ?person ?party WHERE {

?person rdf:type dbo:Person.

?person dbo:party ?party.

} order by ?person.

 offset 43880
```

Query: 4

The following query is executed against DBpedia to return all of the results for the properties 'company' and 'headquarter'-

```
select DISTINCT ?x ?y where
{

?x a dbo:Company.

?y a dbo:City.

?x dbo:headquarter ?y.

} order by ?x
```

3.2 Cleaning

After extracting all data from DBpedia, cleaning was done. Initially, we eliminated "http://dbpedia.org/resource/" from all of the data. If the result contains "-" it was replaced with "_". In case of data which are inside the parentheses () and if the line ends with "_", we removed them to make the data meaningful for further processing. Thereafter word-embeddings pre-trained on the Google News corpus with the word2vec method is used to retrieve those data which are in that model. For instance, if the DBpedia data are in the word-embeddings pre-trained on the Google News corpus we considered them for further processing (filtered data) and ignored the rest of the DBpedia data. We used word-embeddings pre-trained model to get vector representation of a word so we had to make sure that the DBpedia data are in the word-embeddings pre-trained on the Google News corpus model.

3.3 Utilized Pre-Trained Word Embeddings

We considered pre-trained word-embeddings to represent word as a vector. Word-embeddings is published by Mikolov et al (2013). It contains three million words and phrases from Google News dataset and trained around 100 billion words[1]. We retrieved vectors for words from word-embeddings pre-trained on the Google News corpus with the word2vec method for our project experiment.

Representation of vector for word relies on many techniques such as GloVe, Word2vec and so on. We used word2vec in order to get vector representation of the word. The concept behind word to vector has following advantages-

1. Semantic likeliness (similarity) of word is represented by vector.
2. Analogical task is also represented by vector.

Vector for word is retrieved from the pre-trained word-embeddings. The number of dimensions for a vector is fixed and it is usually 300.

3.4 Learn Analogy

One of the advantages of vector representations of words is similarity calculation. It is possible to find out the most similar words and their distance by using word vector. For example, if we query vector space for vectors close to man it returns the words and corresponding cosine similarity values in the vector space created by the pre-trained word-embeddings as shown in Figure 3.2. Analogy has been playing a significant role to detect similarity in a syntactic and semantic way for a set of words and their relationship. Word analogy is the idea that word pairs with similar relation can be used to learn new word analogies. For a given word pair source:target the intention of learn analogy is to predict the target word pair which has the same relation. As an example for a word pair Berlin:Germany, the target word pair would be Paris:France, Venice:Austria, Madrid:Spain and so on.

[1] https://code.google.com/archive/p/word2vec/

```
[('woman', 0.7664012312889099),
('boy', 0.6824870109558105),
('teenager', 0.6586930155754089),
('teenage_girl', 0.6147903800010681),
('girl', 0.5921714305877686)]
```

Figure 3.2: Most similar words of man

3.5 Predict Relation Target

It is also possible to predict relation target by using vector representation from word-embeddings pre-trained on the Google News corpus with the word2vec method. For instance, consider the following equations-

$$w_1 \rightarrow w_2 \tag{3.1}$$

$$w_3 \rightarrow w_4? \tag{3.2}$$

If relation source w_1 is related to relation target w_2 in some way then relation source w_3 is related to relation target $w_4?$ with the same exact process. So, we need to predict the relation target $w_4?$. For our approach, in order to predict the relation target and to get more accurate results, we implemented two super vectors:- Super Vector Source (SVS) and Super Vector Target (SVT). SVS and SVT are the amalgamation of one to twenty relation sources and relation targets respectively from the training data.

Based on the equation 1.1, we can write-

$$1 \times testing\_source - vec([SVS]) + vec([SVT]) = vec[Result] \tag{3.3}$$

Where $vec([SVS])$ and $vec([SVT])$ are the combination of relation sources and relation targets respectively from the training data. $vec[Result]$ is the prediction of testing target for a respective testing source. For obtaining relation target of testing data, the relation source of testing data is composed with the SVS and SVT. We retrieved vectors for all words in the test and training set and tested all of the relation source vectors from the testing set against the training dataset vectors in word-embeddings pre-trained on the Google News corpus with the word2vec method to get relation target vectors of the testing set.

Assume, if Kigali is related to Rwanda, Paris is to France, Vienna is to Austria, Lima is to Peru, then Kabul is related to what? To obtain the result of Kabul is related to what, we aggregated all of the capital name vectors together to construct SVS and country name vectors together to build SVT. For example, consider the following representation-

vec["Kabul"] - (vec["Kigali"] + vec["Paris"] + vec["Vienna"] + vec["Lima"]) + (vec["Rwanda"] + vec["France"]+ vec["Austria"] + vec["Peru"]) = ?

Where vec["Kabul"] is the testing source vector. SVS and SVT are the composition of all the capital name and country name respectively in vector form. Output would be the predicted results in vector form against the testing source Kabul.

3.6 Evaluation

For this project, the following four relations capital-country, currency-country, person-party and company-headquarter were chosen from DBpedia to evaluate the system's performance. At the beginning, data were extracted from DBpedia by using SPARQL for the above four relations. Thereafter, cleaning was done to obtain filtered data by ensuring that the data are meaningful for further processing. We used those filtered data for the next steps. The (filtered) data were divided into two parts-

1. For training
2. For testing

In every DBpedia properties we took randomly half or more than half filtered data for testing and the rest are for training. Table 3.2 is displayed below and shows how many data were extracted from DBpedia for the above four relations. Moreover, it captured the filtered data too which are divided into training and testing set.

We retrieved vectors of all words for the test and training set and tested all of the relation sources in vector form from the testing set against the training dataset vectors in Google's pre-trained model to get relation targets in vector form of the testing set. From the training set, we considered maximum 20 pairs of source and target which are randomly selected. First, one pair of source and target was chosen randomly to generate

Table 3.2: Extracted and filtered data for four relations

Properties	Extracted Data	Filtered Data	
		Testing	**Training**
Person-Party	86,851	3,400	1,430
Country-Capital	182	73	73
Country-Currency	182	33	32
Company-Headquarter	2,330	122	83

SVS and SVT and used their vector representations in the word analogy task to predict the relation targets of all test set sources. For one pair of source and target we repeated the same process till ten times randomly to avoid inconsistency. Every time we counted how many correct predictions were there to calculate accuracy. Correct predictions is the sum of all the predicted correct testing targets for testing sources. Then we divided the correct predictions with the total number of testing targets to acquire the accuracy. Accuracy is the ratio of the number of the correct predictions to the total number of testing targets. We calculated the accuracy ten times in the same way. Afterward the average of ten accuracies was taken to achieve the average accuracy.

$$Average\_Accuracy = average(accuracy1, accuracy2, accuracy3, ..., accuracy10) \tag{3.4}$$

Secondly, two pairs of source and target were chosen randomly and applied the same procedure to get average accuracy. Then we proceed consecutively till twenty in order to evaluate the performance of the project and to observe which pair of source and target is giving the highest accuracy. The accuracy and average accuracy are calculated by using the following formulas in Equations 3.5 and 3.6 respectively.

$$Accuracy = \frac{Number\_of\_Correct\_Predictions}{Total\_Number\_of\_Test\_Targets} \tag{3.5}$$

$$Average\_Accuracy = \frac{Accumulated\_Accuracies}{Number\_of\_Obtained\_Accuracies} \tag{3.6}$$

4 Results

After constructing super vectors, correct predictions, vector number, accuracy and average accuracy were calculated, we evaluated the system's performance. At the beginning, correct predictions was set to zero to calculate the accuracy. Correct predictions is the sum of all the items, which predict the correct test targets for testing data. If the first predicted relation target is correct, the correct predictions is incremented from zero to one. We repeated the process for the entire testing set and calculated correct predictions. Vector number is the number of (source, target) pair from the training dataset. Accuracy is the ratio of the number of the correct predictions to the total number of test targets whereas average accuracy is the ratio of the accumulated accuracies to the number of obtained accuracies.

Capital-Country

For properties capital-country, the following average accuracy was acquired for the consecutive vector numbers as shown in Table 4.1. The accuracy was obtained by calculating the number of correct predictions to the total number of test targets. We considered top three results. When the correct answer was within the top three results, the correct prediction has incremented. For the relation capital-country, the testing dataset contains in total 73 items. The relation sources and relation targets were capital and country respectively. The dataset was tested against the training dataset. When the super vectors combined a total of 18 vectors from the training set, the correct predictions were 70 for the first iteration and the following accuracy was found.

1. Accuracy = (70/73) = 0.958904109589041

It has been already discussed in the evaluation part that for every pair of source and target we took data randomly 10 times from the training dataset. As the data were chosen randomly, so accuracy was changing at every time and results are follows.

Table 4.1: Average accuracy for properties country-capital

Number of Vectors	Average Accuracy	Percentage
1	0.797260274	79.7260274
2	0.884931507	88.49315068
3	0.9	90
4	0.915068493	91.50684932
5	0.902739726	90.2739726
6	0.923287671	92.32876712
7	0.923287671	92.32876712
8	0.905479452	90.54794521
9	0.938356164	93.83561644
10	0.92739726	92.73972603
11	0.932876712	93.28767123
12	0.932876712	93.28767123
13	0.924657534	92.46575342
14	0.92739726	92.73972603
15	0.938356164	93.83561644
16	0.930136986	93.01369863
17	0.926027397	92.60273973
18	0.94109589	94.10958904
19	0.934246575	93.42465753
20	0.926027397	92.60273973

2. Accuracy = (68/73) = 0.9315068493

3. Accuracy = (68/73) = 0.9315068493

4. Accuracy = (69/73) = 0.9452054795

5. Accuracy = (69/73) = 0.9452054795

6. Accuracy = (68/73) = 0.9315068493

7. Accuracy = (68/73) = 0.9315068493

8. Accuracy = (69/73) = 0.9452054795

9. Accuracy = (69/73) = 0.9452054795

10. Accuracy = (69/73) = 0.9452054795

The accumulated accuracies were calculated by aggregating the 10 accuracies, which was around 9.410958904109588. The average accuracy is given below.

Average_Accuracy = (9.410958904109588/10) = 94.10958904109

In Table 4.1, the average accuracy for country-capital relation was tabulated from which we can see that the highest average accuracy is around 94%. It is observed that when the super vectors are for one vector, the average accuracy is 79.726% and when we increased the vector number we received higher accuracy. The average accuracy for two vectors is 88.493% and for 3 vectors is 90%. So, with the increasing number of vectors average accuracy is increasing. For 6 and 7 vectors, the average accuracy is same and it is 92.329% and for 8 vectors the average accuracy is 90.548%, which is a little bit less compared to vectors 6 and 7. The average accuracy is 93.836% for 9 vectors. We received the highest average accuracy for 18 vectors and it is 94.110%. It is also noticeable that with the increasing number of vectors the average accuracy is increased or stabilized for consecutively one or two vectors and it goes down around 1% for the next vector. But after composition of 18 vectors in the super vectors, the accuracy started to decrease again. As we are looking for higher accuracy and noticed to have lower accuracy after addition of 18 vectors; so we decided to make the super vectors with the number of 20 vectors. The graphical visualization of the relation capital-country is shown in Figure 4.1.

Currency-Country

For properties currency and country, we received the following average accuracy for the successive number of vectors, shown in Table 4.2. The accuracy and average accuracy was calculated in the same way as we calculated for the properties capital-country. Properties pair country-currency has 33 total items in the testing dataset. Countries name were considered as relation sources and currencies name were the relation targets. For the correct predictions 21, the accuracy is-

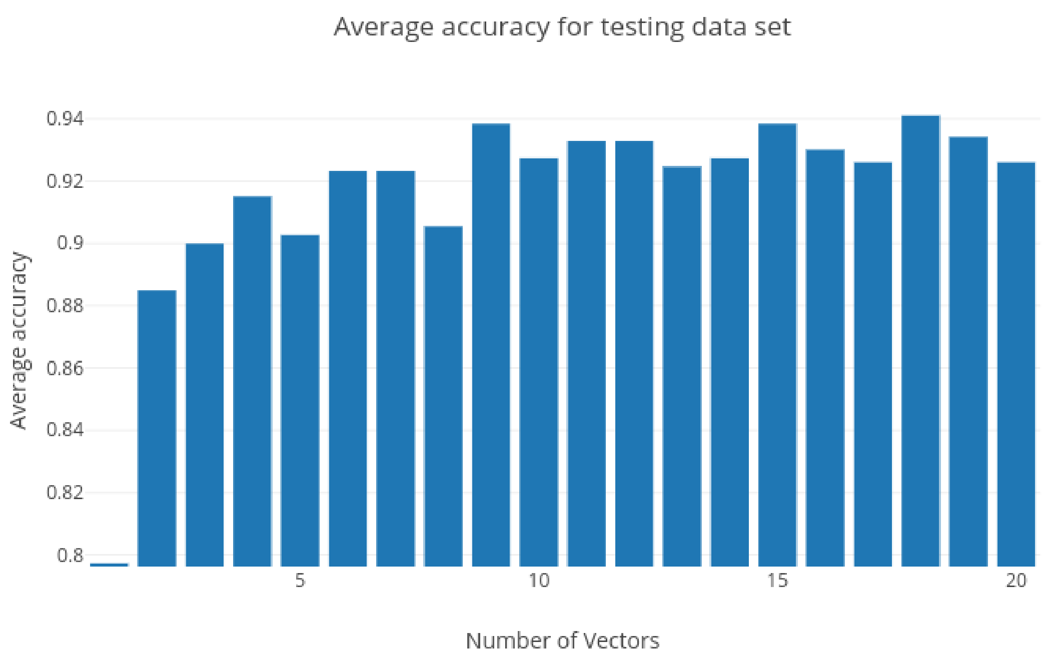

Figure 4.1: Average accuracy against vector number for capital and country

$$\text{Accuracy} = (21/33) = 0.6363636364$$

The highest average accuracy is about 62% for the properties currency and country. We have marked from the Table 4.2 that the average accuracy for one vector is 26.667%. We observed that the average accuracy is significantly increased for 2 and 3 vectors and they are 43.939% and 50.909% respectively. After that, the average accuracy is getting up and a little bit down with the increasing number of vectors. The average accuracy for 7 vectors is 61.212%. Like the properties pair capital-country, the highest average accuracy is acquired by 18 vectors for this relation and it is 62.121%. The visualization of the relation currency-country is shown in Figure 4.2.

Person-Party

The following average accuracy is obtained by the consecutive vector numbers presented in Table 4.3 for the relation person-party.

The system tested 3,400 data items against the training dataset for the properties pair person-party. The relation sources and relation targets were person and party name

Table 4.2: Average accuracy for properties country-currency

Vector Number	Average Accuracy	Percentage
1	0.266666667	26.66666667
2	0.439393939	43.93939394
3	0.509090909	50.90909091
4	0.460606061	46.06060606
5	0.527272727	52.72727273
6	0.545454545	54.54545455
7	0.612121212	61.21212121
8	0.572727273	57.27272727
9	0.6	60
10	0.551515152	55.15151515
11	0.533333333	53.33333333
12	0.590909091	59.09090909
13	0.606060606	60.60606061
14	0.56969697	56.96969697
15	0.581818182	58.18181818
16	0.593939394	59.39393939
17	0.603030303	60.3030303
18	0.621212121	62.12121212
19	0.621212121	62.12121212
20	0.603030303	60.3030303

respectively. As the training data were taken randomly, so we got different number of correct predictions at every iteration. When the correct predictions were 1,663, the accuracy was-

$$\text{Accuracy} = (1{,}663/3{,}400) = 0.4891176471$$

For calculating average accuracy, we also followed the same instruction here as we considered for the properties pair capital-country.

For properties person and party, we achieved the highest accuracy for 9 vectors and it was almost 45%. The average accuracy for one vector was 14.885% and the average accuracy was dramatically increased from 14.885% to 33.724% for two vectors, which is a huge difference. The graphical visualization of the relation person-party is shown in the Figure 4.3.

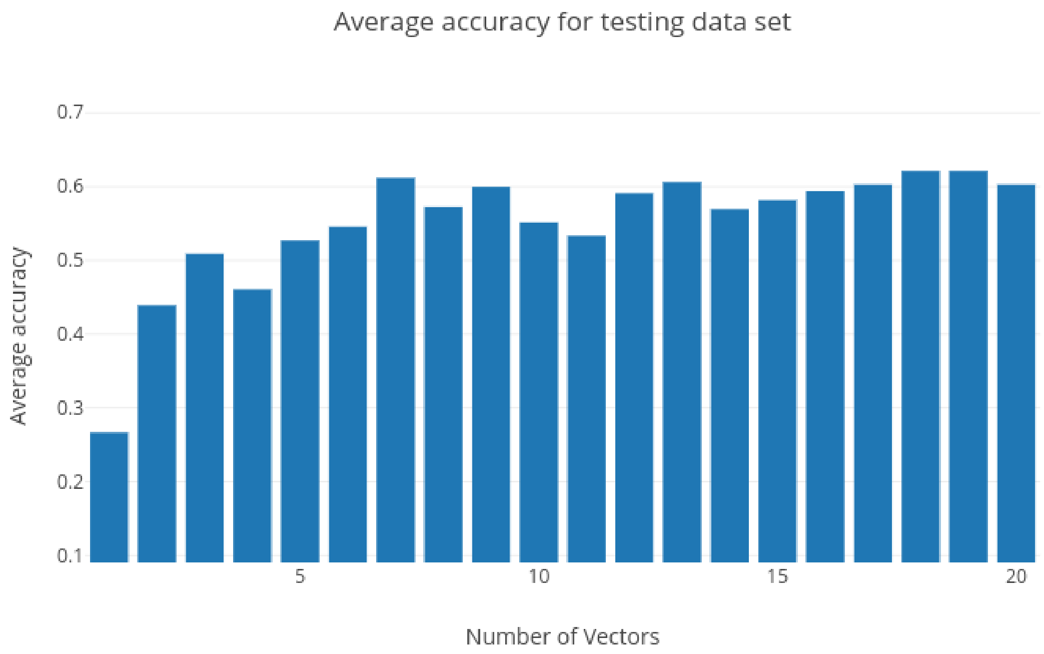

Figure 4.2: Average accuracy against vector number for currency and country

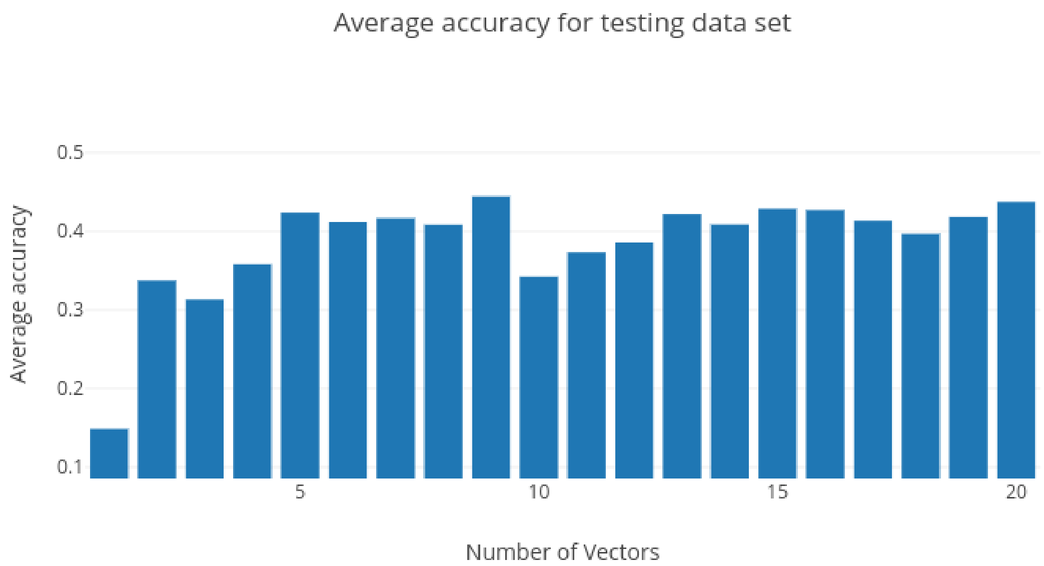

Figure 4.3: Average accuracy against vector number for person and party

Table 4.3: Average accuracy for properties person-party

Vector Number	Average Accuracy	Percentage
1	0.148852941	14.88529412
2	0.337235294	33.72352941
3	0.313264706	31.32647059
4	0.358264706	35.82647059
5	0.423647059	42.36470588
6	0.411882353	41.18823529
7	0.416647059	41.66470588
8	0.408264706	40.82647059
9	0.444441176	44.44411765
10	0.342352941	34.23529412
11	0.373441176	37.34411765
12	0.385882353	38.58823529
13	0.421705882	42.17058824
14	0.408558824	40.85588235
15	0.428882353	42.88823529
16	0.427058824	42.70588235
17	0.413558824	41.35588235
18	0.396735294	39.67352941
19	0.418058824	41.80588235
20	0.437294118	43.72941176

Company-Headquarter

For properties company-headquarter, we got the following average accuracy for the corresponding number of vectors, shown in Table 4.4.

The lowest average accuracy was received by the properties company-headquarter compared to other. For one vector it was around 2.869%. Although the accuracy value was very small for one vector, the highest performance was improved with the increasing number of vectors. The average accuracy was dramatically increased for 2 and 3 vectors and they were 10.0820% and 14.83606557% respectively. After that, the average accuracy was gradually increased from 4 to 13 vectors. The highest average accuracy was achieved by 13 vectors and it was 28.0328%. Thereafter, it started to walk towards a lower position.

Table 4.4: Average accuracy for properties company-headquarter

Vector Number	Average Accuracy	Percentage
1	0.028688525	2.868852459
2	0.100819672	10.08196721
3	0.148360656	14.83606557
4	0.183606557	18.36065574
5	0.21557377	21.55737705
6	0.222131148	22.21311475
7	0.225409836	22.54098361
8	0.239344262	23.93442623
9	0.259016393	25.90163934
10	0.250819672	25.08196721
11	0.254918033	25.49180328
12	0.27704918	27.70491803
13	0.280327869	28.03278689
14	0.260655738	26.06557377
15	0.268852459	26.8852459
16	0.26557377	26.55737705
17	0.267213115	26.72131148
18	0.274590164	27.45901639
19	0.279508197	27.95081967
20	0.276229508	27.62295082

We followed the same procedure here also to calculate the accuracy and average accuracy for the properties company-headquarter as we followed it for the properties capital-country. We tested 122 data items for the properties company-headquarter. The graphical visualization of relation company-headquarter is shown in the Figure 4.4.

The average accuracy for the four relations are graphically captured on one frame in Figure 4.5. From Figure 4.5, it is visible that the accuracy varies from properties to properties. For properties capital-country, the highest average accuracy is around 94% whereas for the properties country-currency highest average accuracy is roughly 62%. Also, the highest average accuracy for the properties person-party and company-headquarter are about 44% and 28% respectively.

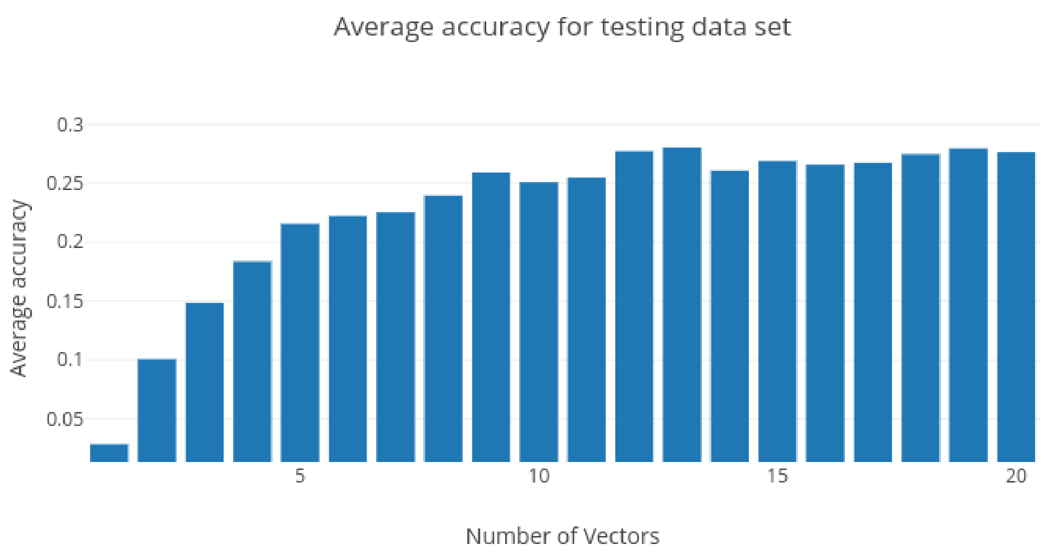

Figure 4.4: Average accuracy against vector number for company and headquarter

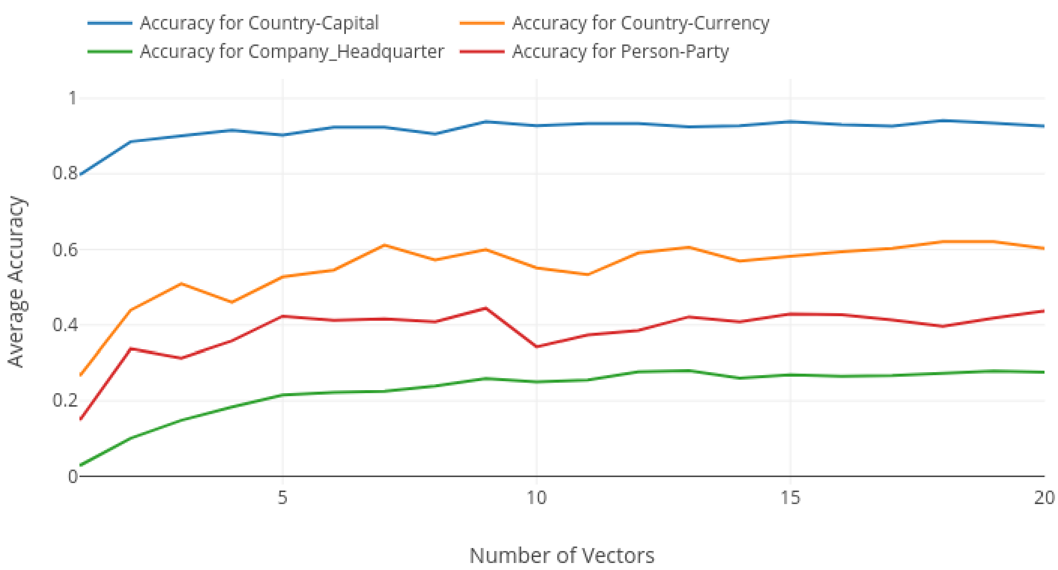

Figure 4.5: Average accuracy against vector number for the four relations

5 Discussion

All of the resources were extracted from DBpedia for the four relations country-capital, currency-country, person-party and company-headquarter by using SPARQL. But it was not possible to think about all of them for further processing. For the above relations, if both the source and target were in the Google pre-trained word vector model only then these data were considered. For example in the country-capital relation for the source United_States and target Washington_D.C. from both one was missing. That's why it was not considered for the next step. Since word-embeddings pre-trained on the Google News corpus model was used to obtain vector of the word, only words covered in the pre-trained repository were considered and words that were not in the repository were omitted.

As it was mentioned in the result section, accuracy varied from properties to properties. We tried to figure out why the accuracy was less for some properties. The following directions were observed during the evaluation of the system-

1. For properties company-headquarter, there were some duplicate data for the company in the filtered dataset. As an example, Buddle_Findlay appeared three times in the company and Safi_Airways appeared twice. If there were no duplicate we might have obtained better performance.

2. For properties country-currency, some of the currencies were combined with their country name. For instance Iraqi_dinar, Bangladeshi_taka, Ghana_cedi, Hungarian_forint, Malaysian_ringgit etc. The super vector was also constructed with these currency name from the training dataset. Sometimes Google's pertained model considers only the currency name without the country. For this reason, the accuracy was less here as compared to capital-country properties.

But for the testing dataset, if the currency is the combination of two words i.e two words are combined with "_" we divided them with "_" and compared the desired results with relation target and also the second part of the divided data. As an example, for the relation target Malaysian_ringgit, if the correct answer is Malaysian_ringgit or only ringgit it increments the correct result. Moreover, case-sensitivity is considered here.

Another reason is that, as we discussed in the related work, two-word pairs are relationally similar if the difference between two vectors are similar. In relation person-party vector differences are not much similar. That's why the average accuracy is not so high for this relation. The reason is that some relations may not be captured nicely with word-embeddings.

We noticed one more thing is that the average accuracies were fluctuating. The reason is that we took random vectors for the combinations and we averaged over 10 iterations with each number of vectors in the super vectors. This random selection and the averaging can cause some minor changes to the overall accuracy.

In word2vec tool, people have been using one pair of source and target to predict relation target. We aggregated more than one vector pair for better performance. Our highest vector pairs were twenty. We observed that the performance is really good. The accuracy was increasing with the elevated number of vectors. As an example, for properties company and headquarter, the average accuracy was around 2.9% for one vector and about 28% for thirteen vectors, which achieved an improvement compared to the analogy task.

6 Conclusion and Outlook

In this report, we proposed a method to aggregate semantic information stored in word-embeddings. The goal of this project is to increase the accuracy of the analogical reasoning by accumulating semantic information stored in word embeddings to predict relation targets of DBpedia properties. The system shows promising results in predicting relation targets of DBpedia data with the increasing number of vectors.

In this project, super vectors have been implemented to get more accurate results of predicting relation targets. This project has some limitations. We only used word2vec method to get vector representations of words even though there are some other techniques, e.g., GloVe, dependency-based word embeddings and so on. Another observation is that we considered only four SPARQL queries against DBpedia to obtain relation (source, target). In the future, we have planned to explore more DBpedia data by using SPARQL and will apply other methods for training word embeddings to predict the relation targets. Moreover, we will consider other languages such as German, Spanish other than English.

Bibliography

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, Christian Jauvin (2003). "A neural probabilistic language model." In: *Journal of machine learning research* 3.Feb, pp. 1137–1155.

Chen, Dawn, Joshua C Peterson, Thomas L Griffiths (2017). "Evaluating vector-space models of analogy." In: *arXiv preprint arXiv:1705.04416*.

Chen, Zhiwei, Zhe He, Xiuwen Liu, Jiang Bian (2017). "An exploration of semantic relations in neural word embeddings using extrinsic knowledge." In: *Bioinformatics and Biomedicine (BIBM), 2017 IEEE International Conference on*. IEEE, pp. 1246–1251.

Gladkova, Anna, Aleksandr Drozd, Satoshi Matsuoka (2016). "Analogy-based detection of morphological and semantic relations with word embeddings: what works and what doesn't." In: *Proceedings of the NAACL Student Research Workshop*, pp. 8–15.

Lassila, Ora, Ralph R Swick, et al. (1998). *Resource description framework (RDF) model and syntax specification*.

Levy, Omer, Yoav Goldberg (2014). "Dependency-based word embeddings." In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vol. 2, pp. 302–308.

Mikolov, Tomas, Kai Chen, Greg Corrado, Jeffrey Dean (2013). "Efficient estimation of word representations in vector space." In: *arXiv preprint arXiv:1301.3781*.

Mnih, Andriy, Koray Kavukcuoglu (2013). "Learning word embeddings efficiently with noise-contrastive estimation." In: *Advances in neural information processing systems*, pp. 2265–2273.

Morsey, Mohamed, Jens Lehmann, Sören Auer, Claus Stadler, Sebastian Hellmann (2012). "Dbpedia and the live extraction of structured data from wikipedia." In: *Program* 46.2, pp. 157–181.

Pennington, Jeffrey, Richard Socher, Christopher Manning (2014). “Glove: Global vectors for word representation.” In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Teofili, Tommaso (2017). “par2hier: towards vector representations for hierarchical content.” In: *Procedia Computer Science* 108, pp. 2343–2347.