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, May 23, 2018

Abstract

Vector representation of words has been learned by many methods, word2vec is one of them and used in many natural language processing and information retrieval. Though vector representation of words has been applied for DBpedia data to predict DBpedia properties, the surprising fact is that it has not been applied for DBpedia data to aggregate semantic information stored in word embeddings. This project targets to introduce a new system called super vector, which is capable to accumulate several vectors in the analogy task to predict the relation target of DBpedia and which can incredibly increase the accuracy determination of the vector analogous characteristics. The following four relations namely capital-country, currency-country, person-party and spouse are considered from DBpedia for this study. Although vector numbers is also used to evaluate the system's performance, average accuracy have to be determined. The highest accuracy is obtained by capital-country relationship and the amount is around 94%. This result shows that the method super vector is much more efficient and promising to do more research in enhancing performance.

Contents

List of Figures

List of Tables

1 Introduction

Word2vec is a neural network, which takes text corpus as input and produces a set of vectors as a result. In the beginning, word2vec build a vocabulary from a large text corpus and then produces a group of vectors. There are two ways to represent word2vec model architecture [Mikolov et al. 2013b].

1. Continuous bag-of-words (predict a missing word from a given window of context words or word sequence) and
2. Skip-gram (predict the neighboring window of target context by using a word)

Continuous bag-of-words (CBOW) and continuous skip-gram model architecture are very popular nowadays in the machine learning areas, natural language processing and advance research areas. We are using skip gram model architecture for our project to learn vector representation of a word. Skip- gram is a proficient method to represent a word as a vector from a huge text corpus. First, it retrieves word as a vector and then learn analogy. Finally, it is used to predict the relation target [Mikolov et al. 2013a].

Words that have equivalent meaning will have analogous vectors and the words whose doesn't have equivalent explanation will have unalike vectors. It is quite surprising that, word vectors follow the analogy rule. For instance, presume the analogy "Berlin is to Germany as Paris is to France". It gives the following results.

$$v_{Germany} - v_{Berlin} + v_{Paris} = v_{France}$$

where $v_{Germany}$, v_{Berlin}, v_{Paris} and v_{France} are the word vectors for Germany, Berlin, Paris and France 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. DBpedia is one of them. DBpedia ("DB" stand for "database") is a crowd-sourced community effort to extract structured information from Wikipedia, make this information available on the Web and has been used as a dataset for diverse purposes[1].

The main aim of this project is the implementation of a technique that can aggregate semantic information stored in word embeddings to predict DBpedia properties. The idea behind the technique is based on a new system; introduced to increase the accuracy which is capable to amalgamate more than one vector. Because the analogy tasks work perfectly with one vector. So theoretically and practically it should work better with the increasing number of vectors. For this purpose, the data are divided into training and testing set. Thereafter Super Vector is introduced. A Super Vector is made by aggregating all of the relation targets from the training set with the relation sources from the testing set. In order to get the desired result (relation target), subtraction of relation sources of training data is done from the Super Vector.

As an example if we have information like:-
Paris is to France, Vienna is to Austria, Lima is to Peru, Berlin is related to what? In order to get answer of Berlin is related to what, we aggregate all of the country name with Berlin in vector form. After that we subtract all of the capital name from super vector to get desire result.

Vector representation of words have not been applied for DBpedia data to aggregate semantic information stored in word embeddings. In this project we introduce a technique to aggregate semantic information. This project is focused on "Predicting relation targets of DBpedia properties using vector representations from word2vec". Vector representation has been applied in several areas with the purpose to detect similarity and the nearest word.

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

1. Berlin is the capital

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

2. Paris is the capital

The result of the Word2vec similarity will be the words ending up near to one another. Suppose, if we train a model with (input:Berlin,output:Capital) and (input:Paris,output:Capital) this will eventually gives insight the model to understand that, Berlin and Paris both are connected to capital, thus Berlin and Paris closely related in the Word2Vec similarity.

The Prediction of relation targets from DBpedia properties using vector representations is developed in python which takes DBPedia properties as input and returns nearest words or relation target as output.

The rest of this report is methodized 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 and Chapter **??** is dedicated to implementation. 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 possibility of the 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. SPARQL is capable to retrieve and manipulate information which is stored in a Resource Description Framework (RDF) format. There are diverse types of output format available for SPARQL such as result sets, JSON, RDF/XML, CSV etc. A 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 SPARQL query looks like.

Example: SPARQL query -

1. PREFIX dbo: ⟨ http://dbpedia.org/ontology/⟩

2. PREFIX dbr: ⟨ http://dbpedia.org/resource/⟩

3. PREFIX s: ⟨ http://schema.org/⟩

4. SELECT * WHERE {

5. ?Hotel a s:Hotel.

6. ?Hotel dbo:location dbr:Dresden.

7.}

The above query is a combination of prefixes and triples. Prefixes are shorthand for long URIs. The SPARQL query returns all the hotels in Dresden as a result when it executed against DBPedia. For example, Dresden has only two hotel under 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)

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

2.1.3 Word embedding

Word embedding is the heart of natural language processing and efficient to capture inner words semantics. A word embedding is a vector representation of a word where each word from a vocabulary is mapped to a vector. Vector representation of word plays an increasingly essential role in predicting missing information by capturing semantic and syntactic information of words, and also 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 are using Word2Vec method.

2.2 Related Work

2.2.1 Word Embeddings

Vector representation of word has gained enormous attention in the field of machine learning. Several techniques are introduced to get word vector. Word2vec and GloVe has gained 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. The relationship between one pair of objects is to that of another pair of objects. This is done by Parallelogram model of analogy [2017]. Parallelogram model of analogy is the representations of objects that contain data which is essential to presume relationship between objects (see fig. 2.1), where objects are represented as points and relations between objects are represented by their difference vectors.

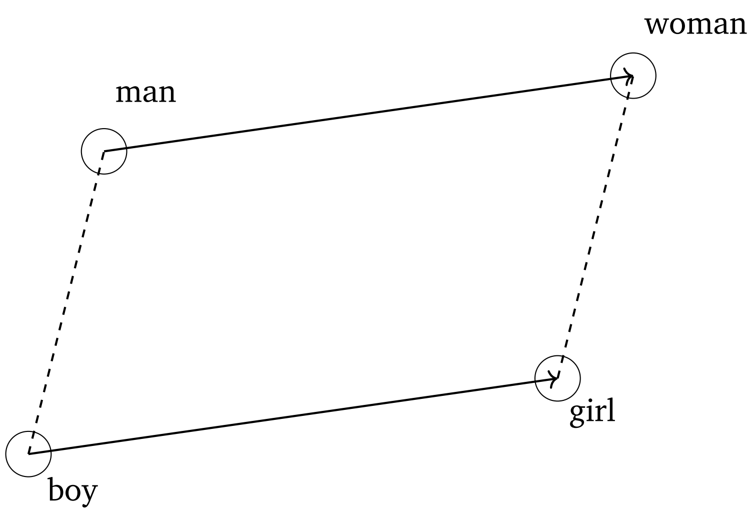


Figure 2.1: The parallelogram model for the analogy boy : girl :: man : ?

Based on the parallelogram model, two words pairs (s_1, t_1) and (s_2, t_2) are syntactically similar if their vector differences ($vect_1 - vecs_1$ and $vect_2 - vecs_2$) are similar. Different types of methods are available to measure the vector difference similarity. Cosine similarity is one of them. The 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 cosine similarity is given in the following.

$$\frac{a.b}{||a||||b||} \tag{2.1}$$

Yoshua Bengio et al. [2003] introduced a technique for learning the distributed representations 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 a training data set. For example, a 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 Learning word embeddings efficiently with noise-contrastive estimation. In their paper, they focused on the analogy-based questions sets, which has the form “x is to y as z is to ”, defined as x : $y \to z$: ? . It outperformed to detect the fourth word, which is syntactically and semantically correct.

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 embedding from Wikipedia. In the field of text mining and Natural Language Processing (NLP), word embedding has 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.

In their approach, they introduced two research questions-

Q1: By using vector representation what kind of semantic relation is possible.

Q2: By using vector representation how it is possible to detect semantic relations.

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

For that they used WordNet and Unified Medical Language System (UMLS) resources. Wordnet unite nouns, verbs, adjectives, and adverbs into sets of cognitive synsets. Synsets are a 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 and to answer the above two research questions.

The four linguistic relations inflectional, derivational, lexicographic and encyclopedic semantics are 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. For that they have tested 99200 questions in 40 relations. For word embedding, they used GloVe and SVD method. In their experiments, some relation has given excellent accuracy and some didn't and the accuracy varied between around 10% to about 98%. In one hand, they received around 98% accuracy for the relation capital-country. On the other hand, the accuracy is around 10% for the relation meronyms–member. This proves that a model may be more effective with some categories rather than others. The reason is that some relations may be not captured nicely with word embedding or vector offset. They have shown that the accuracy is also varied with window-size, word-category and vector dimensionality. GloVe gives 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 the project. DBpedia data is given as input and we have used pre-trained model GoogleNews-vectors-negative300.bin to get the vector representation of the given data. Thereafter we tried to find out the nearest word based on the given data. After that, the relation target was figured out as if one word is related to another word, the same type of word would be related to another word which was unknown to us. For instance, if Donald Trump is to republican as Barack Obama is to what? And the output would be Democratic.

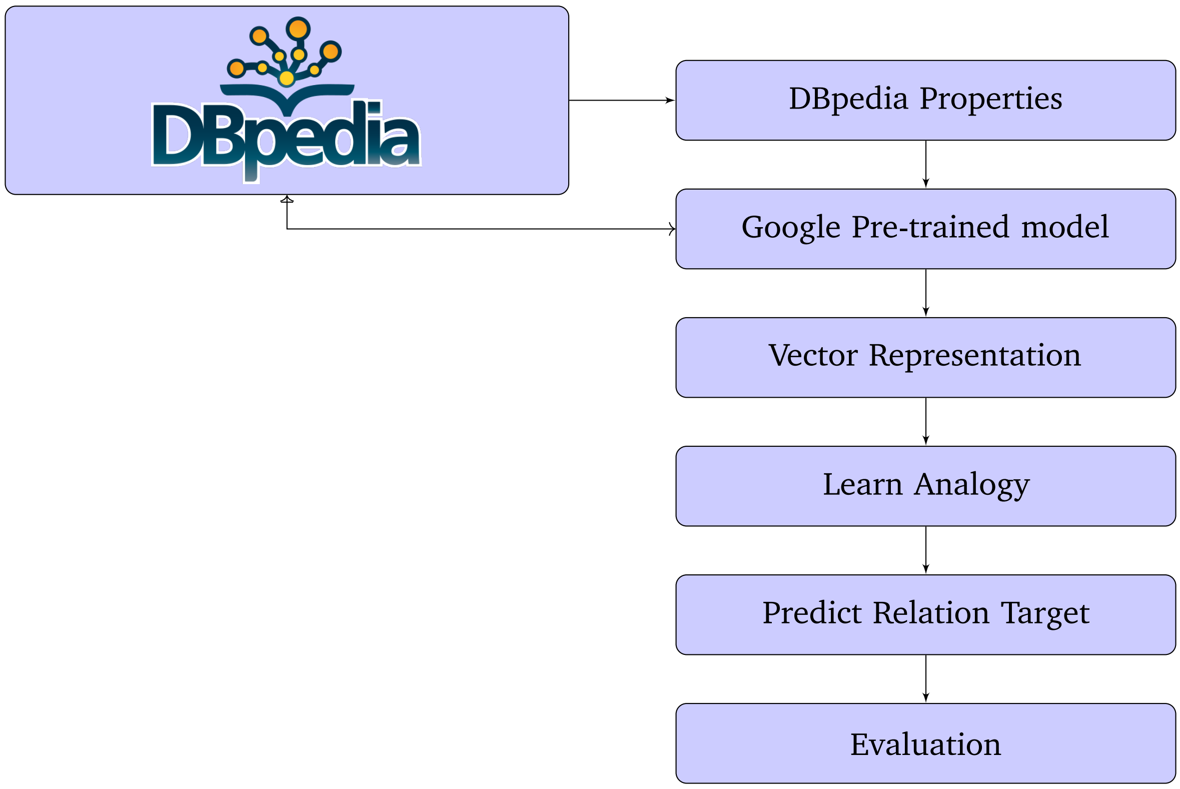


Figure 3.1: The General Scheme of Extracting Vector Representation of DBpedia Properties from Word2Vec

3.1 Query DBPedia Properties for relations

DBpedia has huge amounts of data. It is a combination of resources, ontology, properties and many more. Properties is a connection between objects or resources in DBpedia. To get all of the objects for specified properties we use SPARQL query language. For this project, it may possible to think about four SPARQL query against DBPedia.

Query 1:

The following code shows how to execute SPARQL query against DBpedia for properties 'country' and 'capital' in order to get all of the country and capital list:

1. SELECT DISTINCT ?country ?capital

WHERE

2. {

3. ?city rdf:type dbo:City;

4. rdfs:label ?label;

5. dbo:country ?country.

6. ?country dbo:capital ?capital.

7. } order by ?country

Table3.1 displayed the result of query 1.

Query: 2

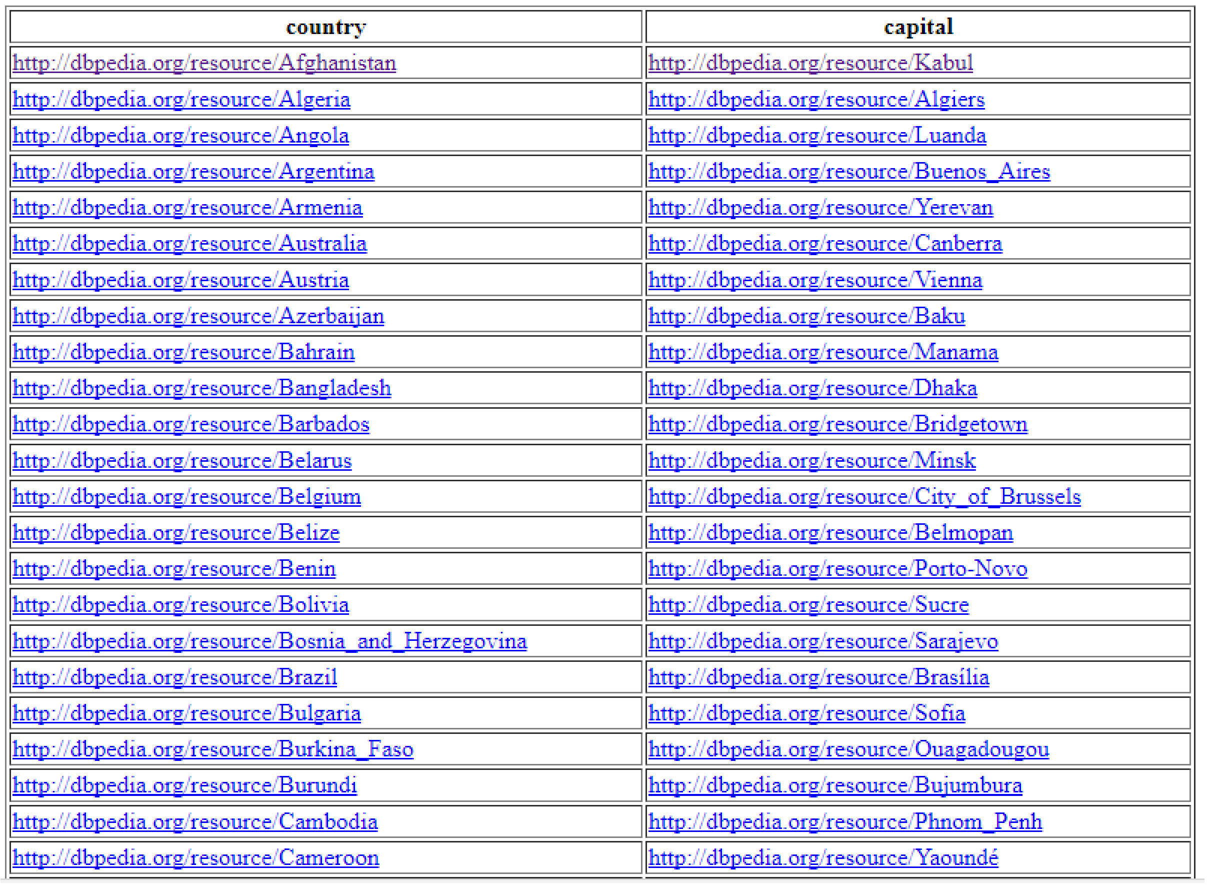
The subsequent query applies for properties country and currency and returns all of the results against those properties:

1. SELECT DISTINCT ?country ?currency WHERE

2. {

3. ?city rdf:type dbo:City;

4. rdfs:label ?label;



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.

5. dbo:country ?country.

6. ?country dbo:currency ?currency.

7. } 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

1. SELECT DISTINCT ?person ?party WHERE

2. {

3. ?city rdf:type dbo:City;

4. ?person rdf:type dbo:Person.

5. ?person dbo:party ?party.

6. } order by ?person

For SPARQL if the result is within 40,000 it is possible to show once at a time. But if the result is more than 40000 it's not possible to display in one page once at a time. Because The DBpedia SPARQL endpoint is configured in the following way:

$$\text{MaxSortedTopRows} = 40000.$$

In DBpedia for dbo:Person and dbo:party there are huge amounts of data. In order to get the rest of the data we use offset which allows to get the next results from the offset index. For instance, we consider the following query and append it into the results:

1. SELECT DISTINCT ?person ?party WHERE

2. {

3. ?person rdf:type dbo:Person.

4. ?person dbo:party ?party.

5. } order by ?person.

6. offset 43880.

Query: 4

The subsequent query return all of the results for the property spouse.

1. SELECT DISTINCT ?x ?y WHERE

2. {

3. ?x dbo:spouse ?y.

4. ?y dbo:spouse ?x.

5. ?x dbo:spouse |^dbo:spouse ?y.

6. FILTER (?x < ?y)

7. } order by ?x

3.2 Cleaning

After retrieving all data from DBpedia we tried to clean them. For instance, we removed "http://dbpedia.org/resource/" from all of the data. If the result contain "-" we replace it with "_". When the data are inside the parentheses (()) we take aside them with parenthesis and at the end of line if there is "_", we carry away them from the line in order to make it meaningful for further processing.

3.3 Google's pre-trained model

We are considering google pre-trained word vector model (GoogleNews-vectors-negative300.bin) to represents word vector. Google pre-trained word vector model is published by Mikolov et al. It contains three million words and phrases and from a Google News dataset they trained around 100 billion words[1]. We retrieved data from Google's pre-trained model. For example,if the DBpedia data is in the pre-trained model it returns the output (filtered data) otherwise it does not return the data which are not in the model. The output we are using them for our project experiment.

3.4 Vector Representation

Representation of vector for word rely on many technique such as GloVe, Word2Vec and so on. We consider Word2Vec in order to get vector representation of word. The concept behind word to vector has many advantages. For example-

1. it is possible to do matrix addition and subtractions

[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 word of man

2. semantic likeliness (similarity) of words is represented by vector too

We retrieved vectors for words from the Google's pre-trained model.The number of dimensions for a vector is fixed and it is usually 300.

3.5 Learn Analogy

One of the advantages of the vector representation of word is similarity calculation. It's possible to find out the most similar word and their distance by using word vector. Suppose, if we enter the man as an input it returns the following words and corresponding distance from Google's pre-trained model 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. The analogy is an influential cognitive system which provides the ability to compare two words or system of words, by exploring the relationships in which they are considered to be similar. Since analogies have been playing an indispensable role in guiding reasoning methods and in generating conjectures for new domains, it is attracting the attention of AI researchers and cognitive psychologists as an interesting topic [Bartha 2013].

3.6 Predict Relation Target

It is also possible to predict relation target by using vector representation from Word2Vec model. The relationship between the two words is like $w_1 \rightarrow w_2$. And the target word would be $w_3 \rightarrow w_4?$. So, we need to predict the relation target $w_4?$. For our approach, in order to predict the relation target we implemented Super Vector. A Super Vector is a manner, which takes the relation sources from the testing set and add it with the relation targets from the training set. To get the relation target of testing data we took the relation sources from the training set and subtracted it from the Super Vector.

Presume, if Kigali is related to Rwanda, Paris is to France, Vienna is to Austria, Lima is to Peru, then Kabul is related to what? And the representation looks like:

Super_Vector = vec["Kabul"] + vec["Rwanda"] + vec["France"]+ vec["Austria"] + vec["Peru"]

Target_Vector = vec.similar_by_vector((Super_Vector - (vec["Kigali"] + vec["Paris"] + vec["Vienna"] + vec["Lima"])), topn=3)

3.7 Evaluation

After retrieving cleaned data we use Google's pre-trained model to retrieved data (filtered data), which are in the Google's pre-trained model. We are using those filtered data for our next steps. We divided the retrieved (filtered data) data into two parts:

1. For training
2. For testing

In every DBPedia properties we have taken randomly half or more than half of the filtered data for testing and the rest are for training. We have tasted all of the relation sources from the testing set against the training dataset in Word2Vec model to get relation targets of the testing set. From training set, we took 20 vector(source, target) pairs. We took one vector pair(source, target) at first to generate Super Vector, tested against all of the relation sources from testing set and get relation targets. For one vector pair, we took the data randomly 10 times from the training set. In every time we counted how many correct results were there. Then we divided the correct results with the total number of testing data and got the accuracy. We repeated it 10 times randomly in the same way and calculated accuracy respectively. Afterward, we divided the accuracy with the random choice number 10 and achieved the average accuracy.

Secondly, two vector pairs(source, target) were chosen, and applied the same procedure to get average accuracy. After that, we proceed consecutively till twenty in order to evaluate the performance of the project.

4 Results

As we mentioned before we implemented Super Vector by adding the relation sources of testing data with the relation targets of training data. To predict the relation targets of testing data we subtracted the relation sources of training data from the Super Vector. We tested all of the relation sources from the testing set one after another against the training set. At the beginning, we set correct item to zero. A correct item is the sum of all the items, which predict the correct answer for testing data. If the first predicted relation target is correct, the correct item is incremented from zero to one. We repeated the process for all of the testing set and calculated correct item. Vector number, accuracy and average accuracy are used to evaluate the performance of the project. Vector number is the number of (source, target) pair from the training dataset. Accuracy is the ratio of the number of the correct items to the number of total items whereas average accuracy is the ratio of the accumulated accuracies to the number of obtained accuracies. The accuracy and average accuracy are calculated using the formulas in Equations 5.1 and 5.2 respectively.

$$Accuracy = Number\_of\_Correct\_Items/Number\_of\_Total\_Items \tag{4.1}$$

$$Average\_Accuracy = Accumulated\_Accuracies/Number\_of\_Obtained\_Accuracies \tag{4.2}$$

For properties capital and country we got the following average accuracy against the vector number shown in Table 4.1. The accuracy is obtained by calculating the number of correct items against the number of total items of testing data. We consider the top three results. If the correct answer is in the top three results the correct item will be increased. For properties, capital-country the testing dataset contains 73 total items. The relation sources are capital here and it predicted country as a relation target. The dataset was tested against the training dataset. When the number of vectors was 18 the correct items were 70 for the first iteration and the accuracy is:-

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

1. Accuracy = (70/73) = 0.958904109589041

We have discussed in the evaluation that for every vector pair we have taken the data randomly 10 times from the training dataset. As the data were chosen randomly, so accuracy was changing in every time and results are follows.

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 is around 9.410958904109588. The average accuracy is:-

Average_Accuracy = (9.410958904109588/10) = 94.10958904109

The graphical visualization of relation(capital, country) is shown in the figure 4.1 for average accuracy against the vector number.

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

Accuracy = (21/33) = 0.6363636364

The visualization of relation(currency, country) is shown in the figure 4.2 for average accuracy against number of vectors.

For properties person and party we got the following average accuracy against number of vectors, shown in Table 4.3.

The system tested 3400 data items against the training dataset for the properties pair person-party. The relation sources were the person and the relation targets were the party name. As the training data were taken randomly, so we got a different number of correct items at every iteration. When the correct items are 1663, the accuracy is:-

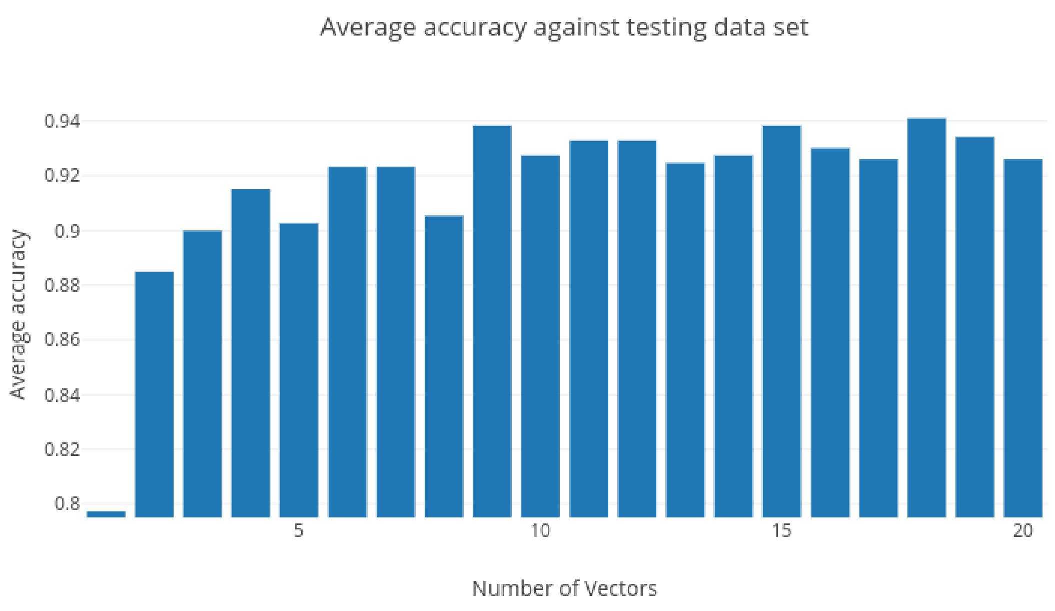


Figure 4.1: Average accuracy against vector number for capital and country

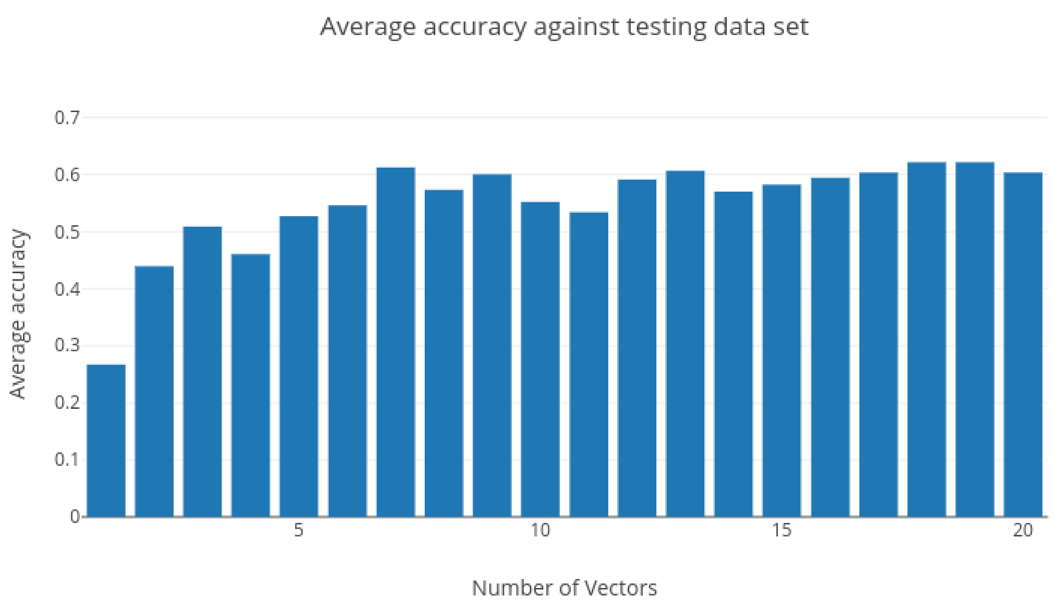


Figure 4.2: Average accuracy against vector number for currency and country

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

Accuracy = (1663/3400) = 0.4891176471

For calculating average accuracy we also follow the same instruction from the properties pair capital-country for the properties pair person-party.

The visualization of relation(person, party) is shown in the figure 4.3 for average accuracy against the number of vectors. For properties spouse we got the following average accuracy against the number of vectors, shown in Table 4.4. We followed the same procedure here to calculate the accuracy and average accuracy for the properties spouse as we followed it for the property capital-country. We tested 503 data items for the properties spouse.

The visualization of relation(male_spouse, female_spouse) is shown in the figure 4.4 for average accuracy against the number of vectors.

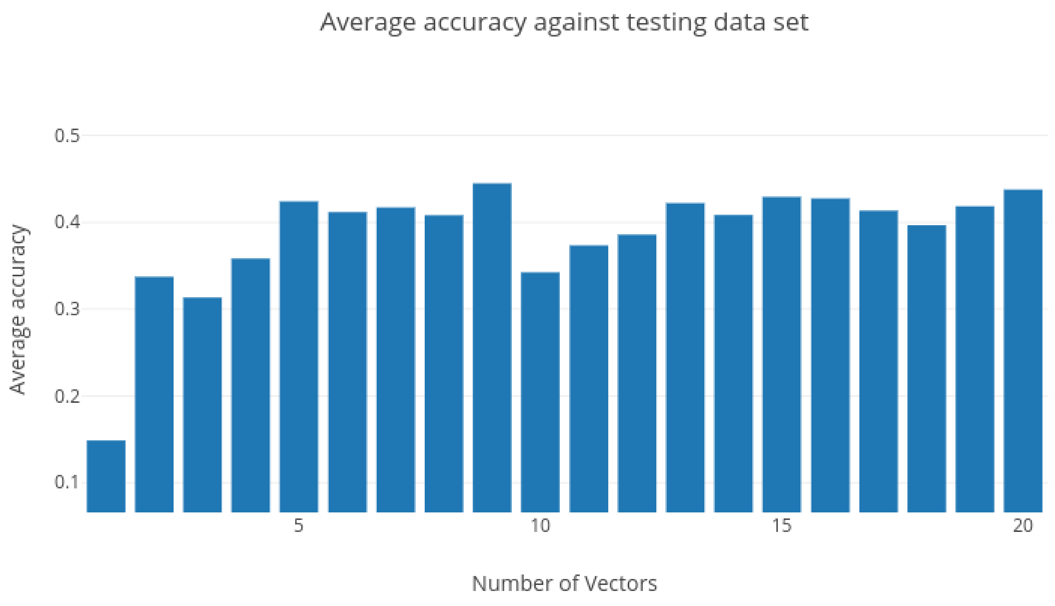


Figure 4.3: Average accuracy against vector number for person and party

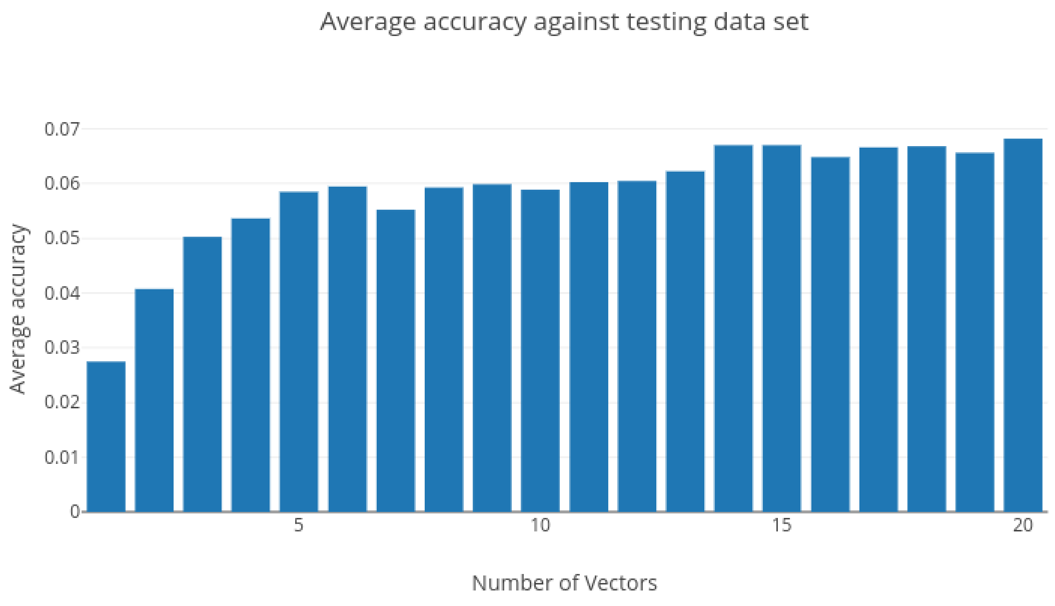


Figure 4.4: Average accuracy against vector number for spouse

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

Table 4.4: Average accuracy for properties spouse

Vector Number	Average Accuracy	Percentage
1	0.027435388	2.743538767
2	0.040755467	4.07554672
3	0.050298211	5.029821074
4	0.053677932	5.367793241
5	0.058449304	5.844930417
6	0.05944334	5.944333996
7	0.05526839	5.526838966
8	0.059244533	5.92445328
9	0.059840954	5.984095427
10	0.058846918	5.884691849
11	0.060238569	6.023856859
12	0.060437376	6.043737575
13	0.06222664	6.222664016
14	0.066998012	6.699801193
15	0.066998012	6.699801193
16	0.064811133	6.48111332
17	0.066600398	6.660039761
18	0.066799205	6.679920477
19	0.065606362	6.560636183
20	0.068190855	6.819085487

5 Discussion

Capital-Country

As we can see from the results for properties capital and country in table-4.1 the highest average accuracy is around 94%. We observed that when the super vector is a combination of one vector, the average accuracy is 79.7260274%. But when we increased the vector number we received higher accuracy. The Average accuracy for two vectors is 88.49315068% and for vector number three is 90%. So, with the increasing number of vectors average accuracy is increasing. For vector number 6 and 7, the average accuracy is same and it is 92.32876712% and for vector number 8 the average accuracy is 90.54794521%, which is a little bit less compared to vector numbers 6 and 7. The average accuracy is 93.83561644% when the number of vectors is 9. We receive the highest average for vector number 18 and it is 94.10958904%. We noticed with the increasing number of vectors the average accuracy is increased or be stabled for consecutively one or two vectors and after that, it goes down around 1% for the next vector. But after the vector number of 18, it has started to walk toward a lower position. Since we are looking for higher accuracy, so we decided to make the super vector with the number of 20 vectors.

Currency-Country

The highest average accuracy for properties currency and country is about 62%. We have marked from the table 4.2 that the average accuracy for the number of vector one is 26.66666667%. we observed that the average accuracy is significantly increased for the vector number 2 and 3 and they are 43.93939394% and 50.90909091% respectively. After that, the average accuracy is getting up and a little bit down with the increasing number of vectors. The average accuracy for the number of vector 7 is 61.21212121%.

Like the properties pair capital-country, the highest average accuracy for the properties pair currency-country is acquired by the vector number 18 and it is 62.12121212%.

Person-Party

For properties person and party, we achieved the highest accuracy for the number of vector 9 and it is almost 45%. The average accuracy for vector number one is 14.88529412% and the average accuracy is dramatically increased from 14.88529412% to 33.72352941% for the vector number two, which is a huge difference. We can confidently say that the goal of the project succeeds.

Spouse

The average accuracy is so less for the properties spouse. We received 2.743538767% average accuracy for the number of vector one. For the vector number 2 and 3, the average accuracy is 4.07554672% and 5.029821074% respectively. Even if it is so less, still it is increasing with the increasing number of vectors. The average accuracy is gradually increased from vector number 4 to 6. There is a bit by bit up and down till the vector number 10. Thereafter it started to rise again until vector number 14, from that it went up a little bit up and down till the vector number 19. The highest average accuracy is achieved by vector number 20 and it is 6.819085487%.

As we discussed in the related work, two words pairs and are syntactically similar if their vector differences are similar. Cosine similarity is used to measure vector difference similarity. For word pair(male_spouse, female_spouse) vector differences are not much similar. That's why the average accuracy is so low for the properties spouse. we can say that the vector difference for word pair(capital, country) is much more similar. For this reason, we obtained about 94.10958904% average accuracy. But, we acquire around 62.12121212% and 44.44411765% average accuracy for the word pair(currency, country) and word pair(person, party) respectively. We graphically captured all of the average accuracies on one frame in figure: 5.1

In Word2Vec tool people have been using one-word vector pair to predict similarity. We tried to aggregate more than one vector pair. Our highest vector pair was twenty. We

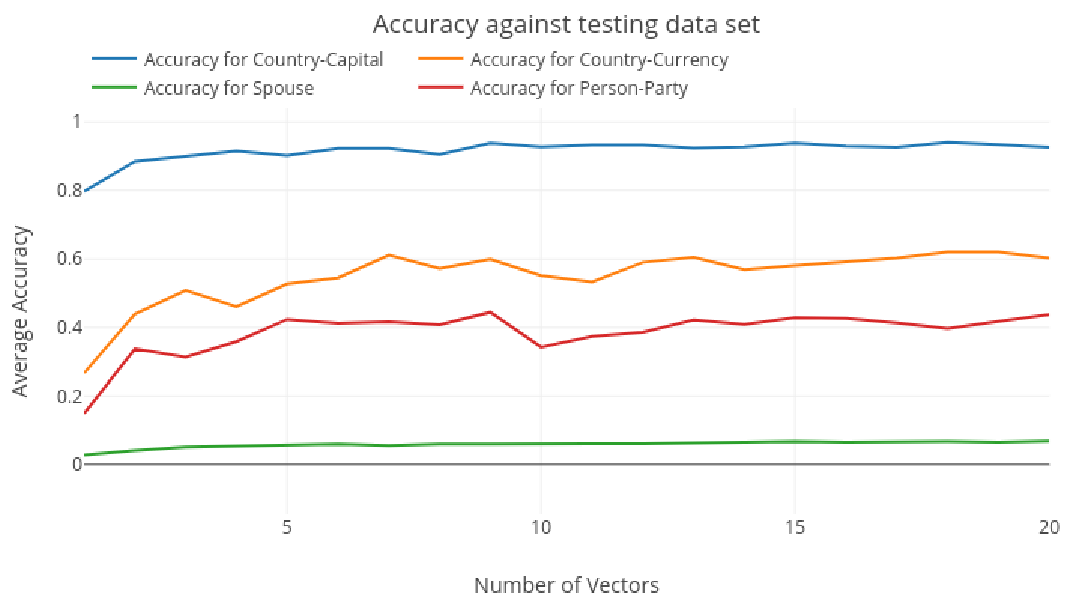


Figure 5.1: Average accuracy against vector number for spouse

observed that the performance is really outstanding. The accuracy was increasing with the elevated number of vectors. As an example, for properties currency and country, the average accuracy is around 26.66666667% for one vector and about 62.12121212% for the vector number nineteen, which is a immense achievement.

6 Conclusion and Outlook

In this project, Predicting relation targets of DBpedia properties using vector representations from word2vec is proposed and implemented. We used knowledge from DBpedia. The goal of this project is to aggregate semantic information stored in word embeddings to predict relation targets. We proposed a new system to evaluate the performance of DBpedia properties. The system shows promising results to detect relation target of DBpedia properties with the increasing number of vectors and pointing out for further research. The project is tested with 4 SPARQL queries against DBpedia and it gives a wonderful performance.

In this project, predicting relation target is explored for DBpedia data. This project has some limitations. We only used the Word2Vec tool to get vector representation even though there are some other techniques are available such as Glove, dependency-based word embeddings etc. Another thing is that we considered only four SPARQL query against DBpedia to obtain relation(source, target). The above limitations are pointed out so that this research project can be further pursued and enhanced.

Bibliography

Bartha, Paul (2013). “Analogy and analogical reasoning.” In:

Bengio, Yoshua et al. (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 et al. (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.” In:

Mikolov, Tomas et al. (2013a). “Distributed representations of words and phrases and their compositionality.” In: *Advances in neural information processing systems*, pp. 3111–3119.

Mikolov, Tomas et al. (2013b). “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 et al. (2012). “Dbpedia and the live extraction of structured data from wikipedia.” In: *Program* 46.2, pp. 157–181.

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

Appendix

A Eidesstattliche Erklärung

Name:	Das
Vorname:	Happy Rani
Matrikel-Nr.:	4576505

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht sind, und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Dresden, 22 May, 2018

Unterschrift