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 2, 2018

Zusammenfassung

Die Vektordarstellung von Wörtern wurde mit vielen Methoden gelernt, word2vec ist eine von ihnen und wird in vielen natürlichen Sprachverarbeitungen und Informationsabfragen verwendet. Die überraschende Tatsache ist jedoch, dass die Vektordarstellung von Wörtern nicht für Dbpedia-Daten angewendet wurde, um in Worteinbettungen gespeicherte semantische Informationen zu aggregieren, um dbpedia-Eigenschaften vorherzusagen. In diesem Artikel konzentrieren wir uns darauf, wie die Vektordarstellung von Wörtern aus word2vec auf Dbpedia-Eigenschaften angewendet werden kann, um das Beziehungsziel zu erhalten. Die Hauptziele der Vorhersage von Beziehungszielen von DBPedia-Eigenschaften unter Verwendung von Vektordarstellungen aus word2vec bestehen darin, im Word2Vec-Modell enthaltene DBpedia-Eigenschaften zu aggregieren, entsprechende Ressourcen zu finden und die Word2Vec-Technik zur Vorhersage des Beziehungsziels anzuwenden.

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, the surprising fact is that Vector representation of words has not been applied for Dbpedia data to aggregate semantic information stored in word embeddings to predict dbpedia properties. In this paper, we are focusing on how Vector representation of words from word2vec can be applied to Dbpedia properties in order to get relation target. The main goals of Predicting relation targets of DBPedia properties using vector representations from word2vec are to aggregate DBpedia properties contained in Word2Vec model, find corresponding resources and to apply Word2Vec technique to predict relation target.

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 group of vectors. There are two ways to represent word2vec model architecture [Mikolov et al. 2013].

1. Continuous bag-of-words (predicts a missing word given a 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

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 us the result like following

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

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 is the largest open linked data and data source, extract structured information from the web and has been used as a dataset for diverse purposes.

The main aim of this project is to implement a technique that can aggregate semantic information stored in word embeddings to predict DBpedia properties and to apply Word2Vec technique to find relation target.

For this purpose, first, we divided our data into training and testing set. Thereafter we introduced Super Vector. 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), we subtract relation sources of the training data 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 of detect similarity and to find out the nearest word.

The intention of this project is to introduce techniques that can be used for learning DBpedia data and to find out corresponding relation target from DBPedia. DBpedia ("DB" stand for "database") is a crowd-sourced community effort to extract structured information from Wikipedia and make this information available on the Web[1] If the user has two sentences like-

1. Berlin is the capital.

2. Paris is the capital.

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

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 give insight the model to understand that, Berlin and Paris both as connected to capital, thus Berlin and Paris closely 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 Section 2 .Section 3 and 4 describe the methodology and implementation of the project. Section 5 discussed the results.In section 6 we discussed about project outcome. We conclude the report with conclusion where possible future work is also mentioned.

2 Preliminaries and Related Work

2.1 Preliminaries

2.1.1 SPARQL Query Language

SPARQL is a semantic query language to query RDF graph. and it's pronunciation is "sparkle", a recursive acronym stands for SPARQL protocol. SPARQL is capable to retrieve and manipulate information 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:locaton 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 these 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 Vector Representation of Words

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. [D. Chen et al. 2017]. proposed "Evaluating vector-space models of analogy", 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 to that of another pairs of objects. This is done by Parallelogram model of analogy [D. Chen et al. 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.

Based on the parallelogram model, two words pairs (s_1, t_1) and (s_2, t_2) are syntactically similar if their vector differences ($vecr_1 - vect_1$ = and ($vecr_2 - vect_2$ are similar. Different methods to measure 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

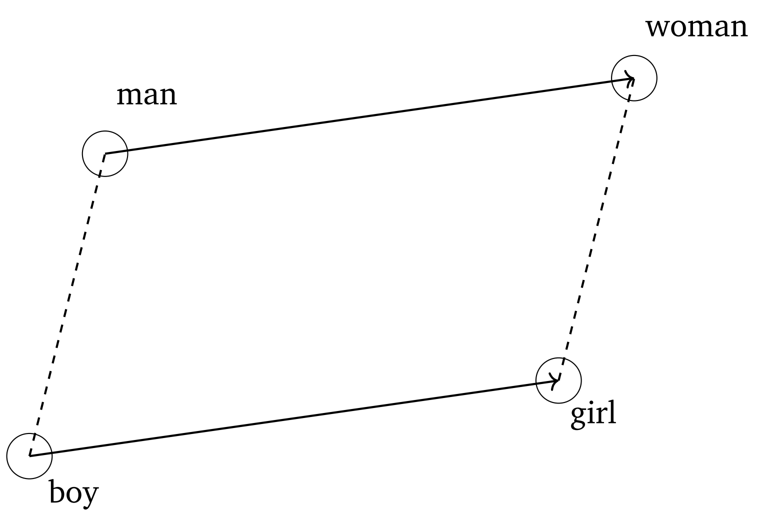


Figure 2.1: The parallelogram model for the analogy boy : girl :: man : ?

angle between them[1]. Suppose, a= $vecr_1 - vect_1$ and b = $vecr_2 - vect_2$. The cosine similarity is

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

Yoshua Bengio et al. [Bengio et al. 2003] introduced a "Neural Probabilistic Language Model" 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 [Mnih and 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 : $y \rightarrow z$: ? . It outperformed to detect the fourth word, which is syntactically and semantically correct.

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

Zhiwei Chen et al. [Z. 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 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, as, Word2vec, dependency based word embeddings, and GloVe. The following nine semantic relations are explored by them [Z. Chen et al. 2017, p1]:-

1. Synonym: a object that means exactly or nearly the same meaning as like another object.
2. Antonym: a object with an opposite meaning with another object, e.g., beautiful : ugly, long : short, and precede : follow.
3. Hypernym: a object with a wide meaning that more specific words fall under. For instance, color is a hypernym of green. 4. Hyponym: a object of more specific meaning than a general or superordinate object applicable to it. For instance, green is a hyponyms of color.
5. Holonym: a object that denotes a whole whose part is denoted by another term,, e.g., arm is the holonym of hand.
6. Meronym: a term that denotes part of something but which is used to refer to the whole. It is the opposite relationship of holonym.
7. Sibling: the relationship denoting that terms have the same hypernym. E.g., son and daughter are sibling terms, since they have the same hypernym child.
8. Derivationally related forms: terms in different syntactic categories that have the same root form and are semantically related, e.g., childhood is a derivationally related from of child.
9. Pertainym: adjectives that are pertainyms are usually defined by phrases such as "of or pertaining to" and do not have antonyms, e.g., America is a pertainym of American.

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 to detect semantic relations.

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.

3 Methodology

Figure 3.1 shows the flowchart diagram of our project. DBpedia data is given as input and we use pre-trained model GoogleNews-vectors-negative300.bin to get the vector representation of the given data. Thereafter we try to find out the nearest word based on the given data. After that we try to figure out the relation target like if one word is related to another word, the same type of word would be related to other word which we don't know about it. For instance, if Donald Trump is to republican as Barack Obama is to what? And the output would be Democratic.

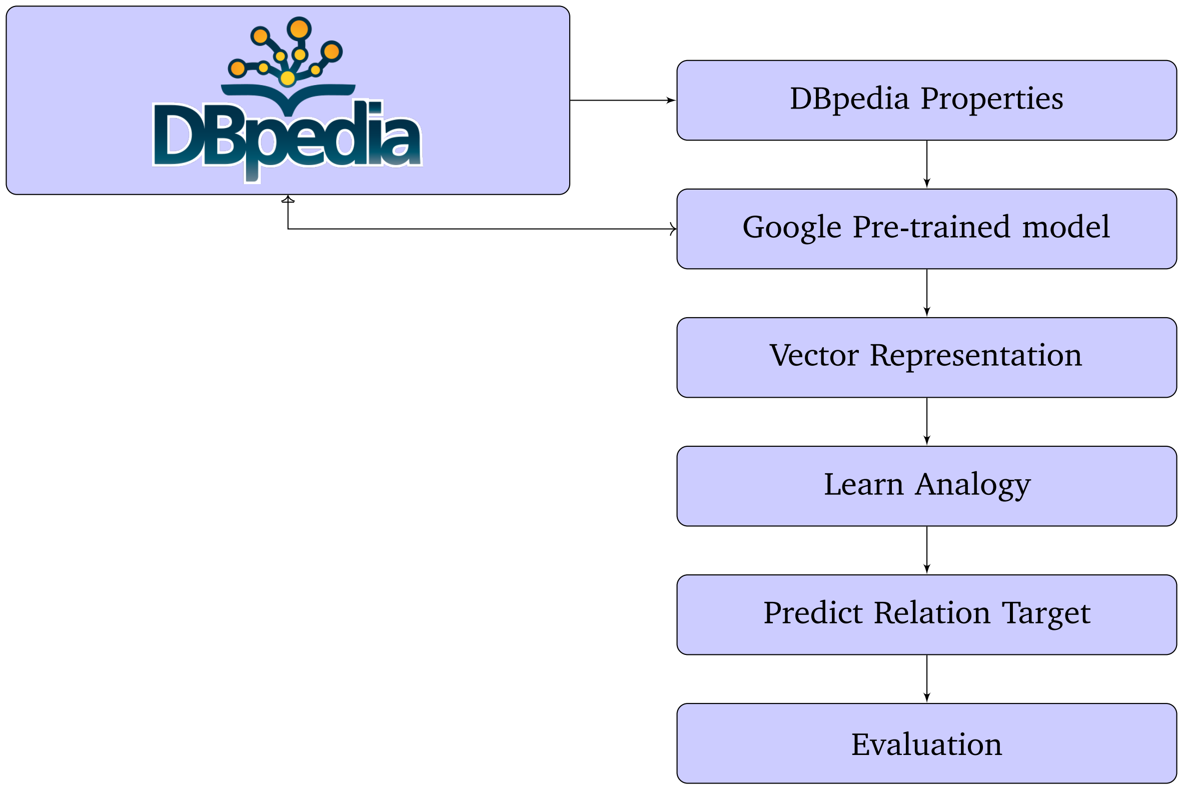


Figure 3.1: The General Scheme of Extracting Vector Representation of DBpedia Properties from Word2Vec

3.1 Query DBPedia Properties for relations

DBpedia has huge amounts of data. It is the 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 our project, we think about four SPARQL query against DBPedia.

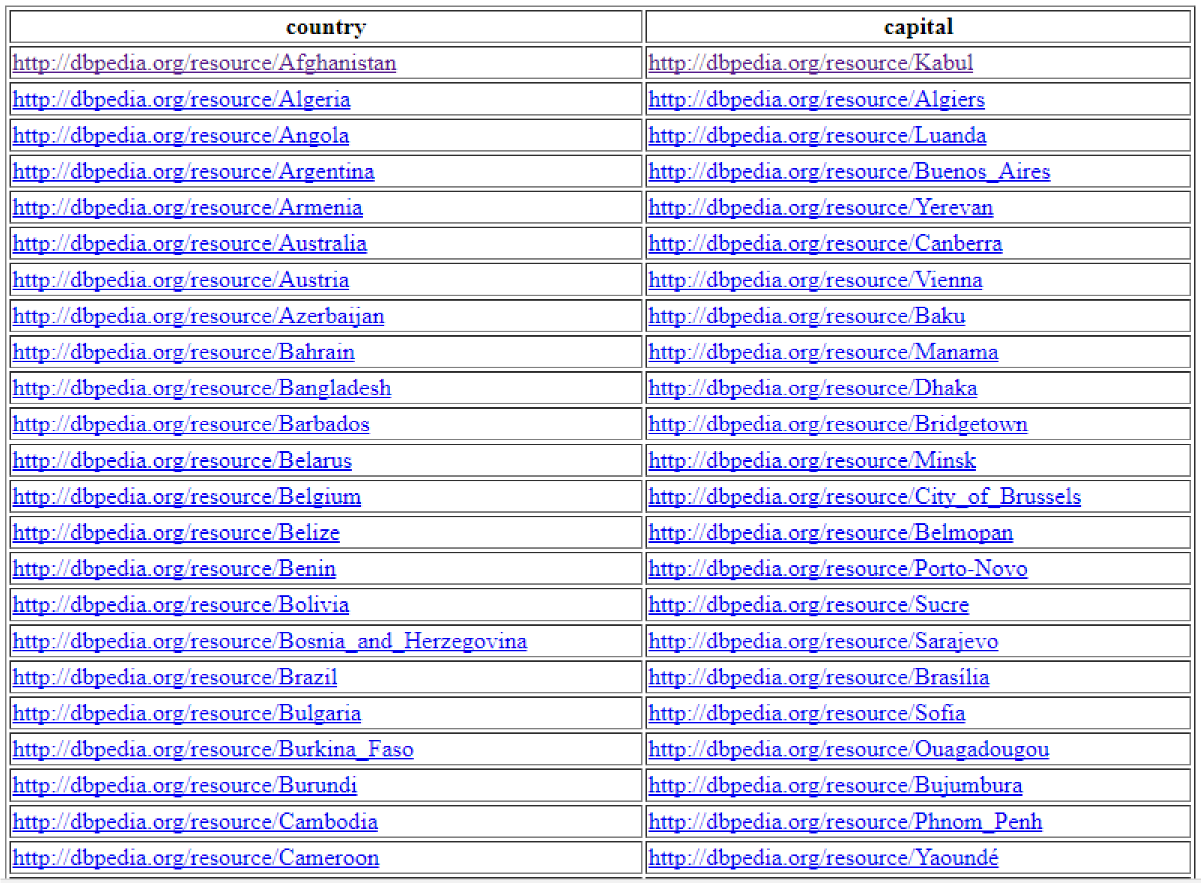
Query 1:

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:

```
SELECT DISTINCT ?country ?capital
WHERE
{
?city rdf:type dbo:City ;
rdfs:label ?label ;
dbo:country ?country .
?country dbo:capital ?capital .
} order by ?country
```

Table: 1 displayed the result of query: 1.

Query:2



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.

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 ;

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

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 result:

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.} order by ?x

3.2 Cleaning

After retrieving all data from DBpedia we try to clean them . Such as we remove "http://dbpedia.org/resource/" from all of the data, if the result contain "-" we replace it "_", in case 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.

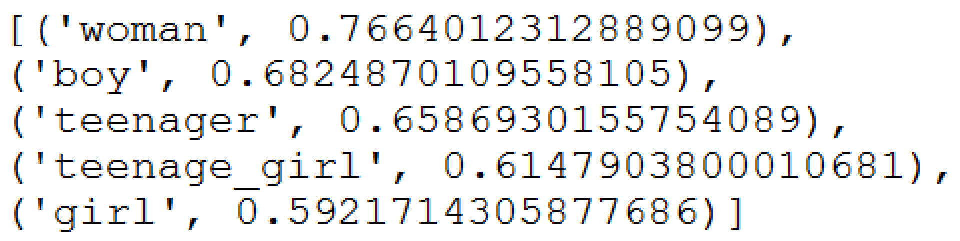
3.3 Google's pre-trained model

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 Tomas Mikolov et al. It contains 3 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

[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

behind word to vector has many advantages. For example, it is possible to do matrix addition and subtractions and semantic likeliness (similarity) of words is represented by vector too. When we send a word to Google's pre-trained model it returns vector for this word.The number of dimensions for a vector is fixed and it is usually 300.

3.5 Learn Analogy

One of the advantages of vector representation of word is similarity prediction.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 return the following words and corresponding distance from Google's pre-trained model as shown in figure 3:

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 implement 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 take the relation sources from the training set and subtract 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 take 20 vectors(source, target) pairs. Firstly, we take one vector pair(source, target), generate Super Vector, tested against all of the relation sources from testing set and get relation targets. For one vector pair, we take the data randomly 10 times from training set. In every time we counted how many correct results are there. Then we divided the correct results with the number of data of the testing dataset and get the accuracy. We repeated it 10 times randomly in the same way and calculated accuracy. Afterward, we divided the accuracy with the random choice number 10 and achieved the average accuracy.

Secondly, we choose two vector pairs(source, target), and apply the same procedure to get average accuracy. After that, we proceed it for third, fourth and so on till twenty respectively in order to acquire average accuracy.

4 System Implementation

Predicting relation targets of DBPedia properties using vector representations from word2vec is implemented by Python, SPARQL, Virtuoso and with the help of some other Python libraries such as Gensim, SPARQLWrapper etc.SPARQL query language is used to retrieve data from DBpedia. With the SPARQLWrapper it is possible to access the DBpedia dataset live through Virtuoso SPARQL Query Editor. By using Python DBpedia data is sent to word2Vec model to acquire vector representation.

5 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 subtract 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. In the beginning, we set correct result to zero. A correct result is the sum of all the results, which predict the correct answer for testing data. If the first predicted relation target is correct we increment the correct result zero to one. We repeated the process for all of the testing set and calculated correct result. 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 results to the total number of testing data whereas average accuracy is the ratio of the accuracy to the number of random choices of training data. The accuracy and average accuracy are calculated using the formulas in Equations 5.1 and 5.2 respectively.

$$Accuracy = Correct\_Result / Number\_of\_Testing\_data \tag{5.1}$$

$$Average\_Accuracy = Accuracy / Number\_of\_RandomChoise \tag{5.2}$$

For properties capital and country we get the following average accuracy against vector number, shown in Table 5.1:

The graphical visualization of relation(capital, country) shown in the figure 5.1 for average accuracy against the vector number-

For properties currency and country we get the following average accuracy against number of vectors, shown in Table 5.2:

Table 5.1: Average Accuracy of Country and Capital

Number of Vectors	Average Accuracy	Percentage
1	0.757534247	75.7534247
2	0.773972603	77.3972603
3	0.816438356	81.6438356
4	0.815068493	81.5068493
5	0.801369863	80.1369863
6	0.806849315	80.6849315
7	0.820547945	82.0547945
8	0.816438356	81.6438356
9	0.824657534	82.4657534
10	0.812328767	81.2328767
11	0.801369863	80.1369863
12	0.821917808	82.1917808
13	0.824657534	82.4657534
14	0.816438356	81.6438356
15	0.81369863	81.369863
16	0.81369863	81.369863
17	0.823287671	82.3287671
18	0.820547945	82.0547945
19	0.820547945	82.0547945
20	0.82739726	82.739726

The visualization of relation(currency, country) shown in the figure 5.2 for average accuracy against number of vectors.

For properties person and party we get the following average accuracy against number of vectors, shown in Table 5.3.
The visualization of relation(person, party) shown in the figure 5.3 for average accuracy against the number of vectors.

For properties spouse we get the following average accuracy against the number of vectors, shown in Table 5.4:
The visualization of relation(male_spouse, female_spouse) shown in the figure 5.4 for

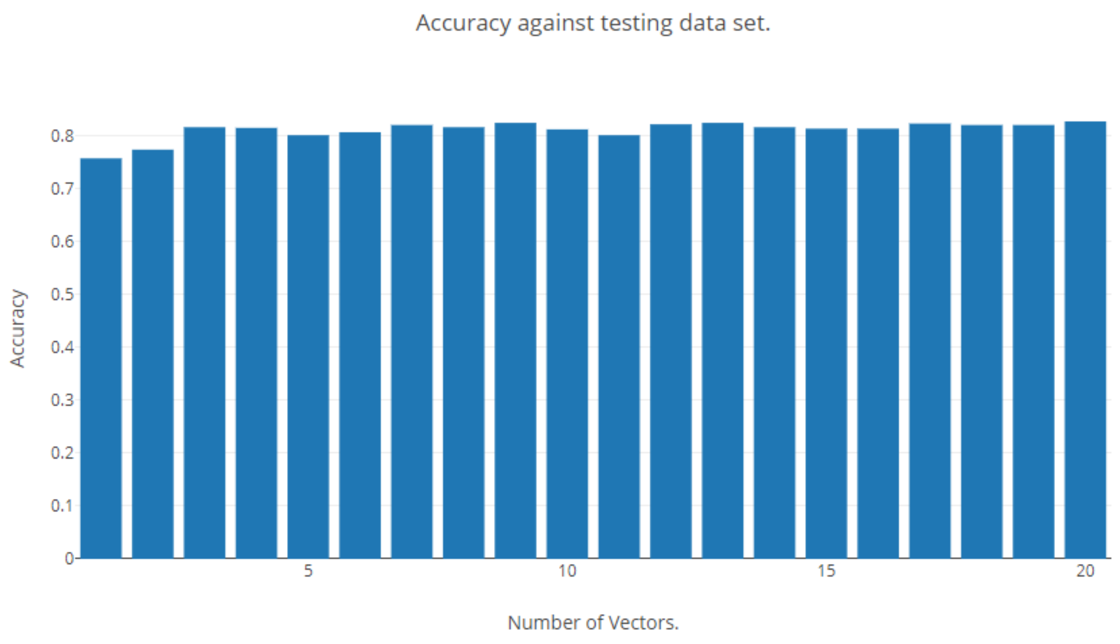


Figure 5.1: Average accuracy against vector number for capital and country

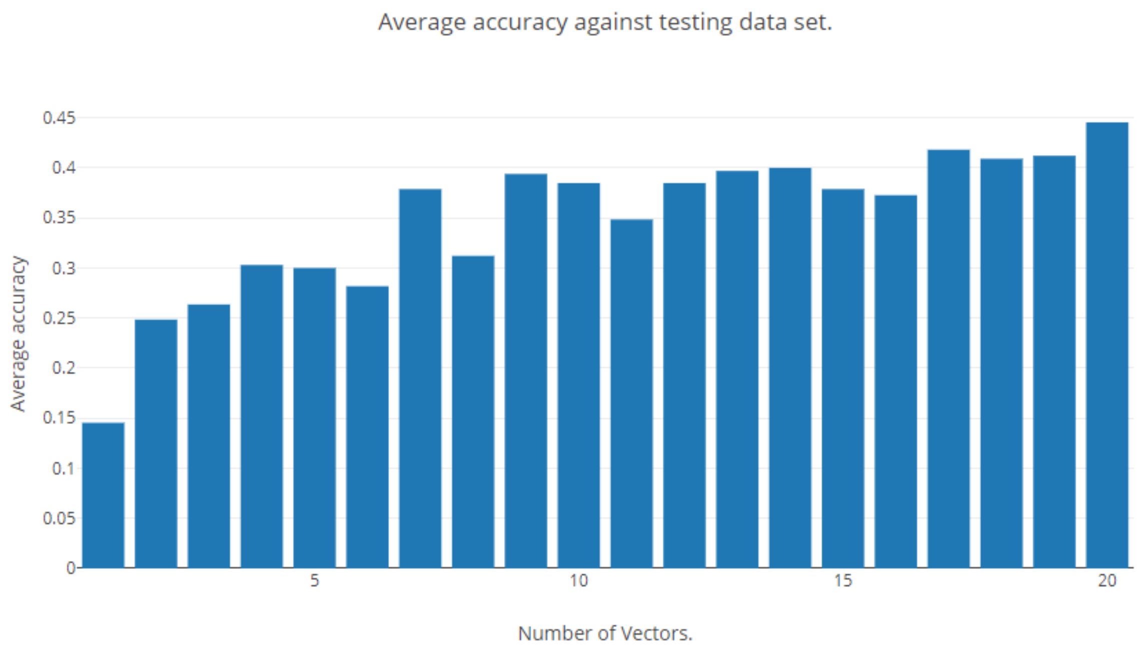


Figure 5.2: Average accuracy against vector number for currency and country

Table 5.2: Average accuracy for currency, country

Vector Number	Average Accuracy	Percentage
1	0.145454545	14.54545455
2	0.248484848	24.84848485
3	0.263636364	26.36363636
4	0.303030303	30.3030303
5	0.3	30
6	0.281818182	28.18181818
7	0.378787879	37.87878788
8	0.312121212	31.21212121
9	0.393939394	39.39393939
10	0.384848485	38.48484848
11	0.348484848	34.84848485
12	0.384848485	38.48484848
13	0.396969697	39.6969697
14	0.4	40
15	0.378787879	37.87878788
16	0.372727273	37.27272727
17	0.418181818	41.81818182
18	0.409090909	40.90909091
19	0.412121212	41.21212121
20	0.445454545	44.54545455

average accuracy against the number of vectors-

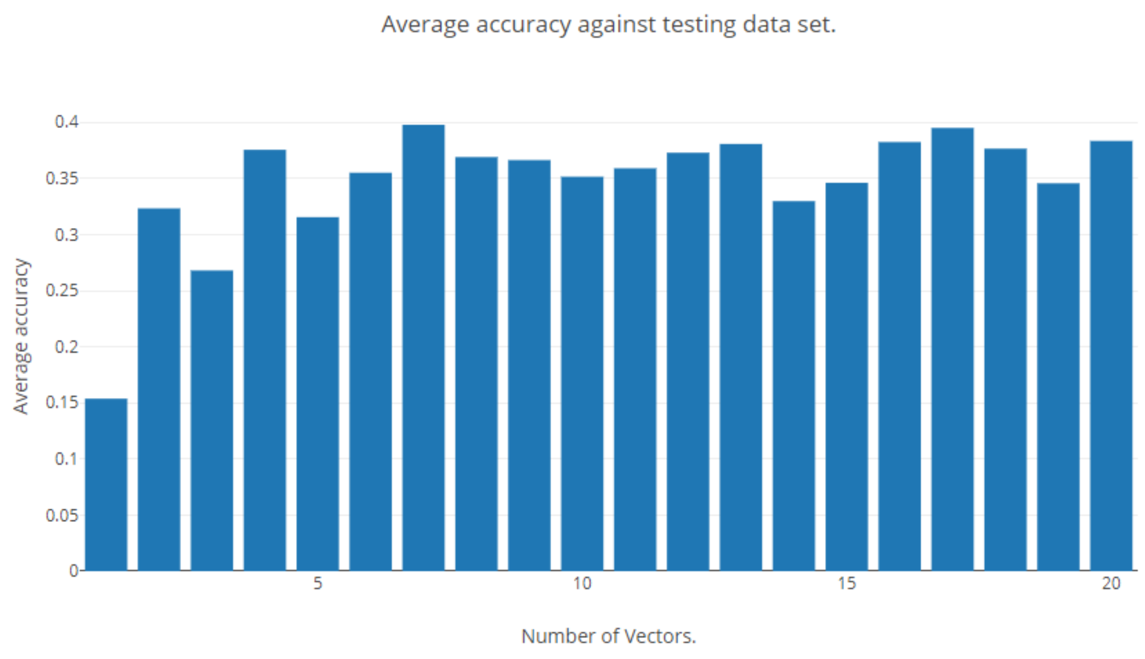


Figure 5.3: Average accuracy against vector number for person and party

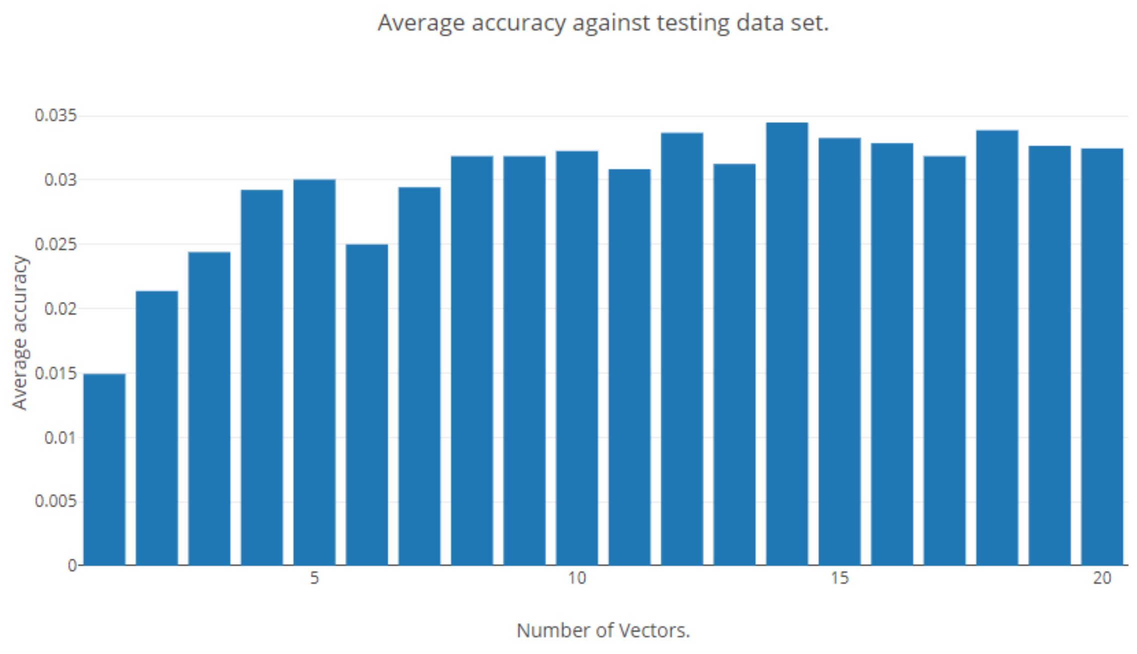


Figure 5.4: Average accuracy against vector number for spouse

Table 5.3: Average accuracy for properties person, party

Vector Number	Average Accuracy	Percentage
1	0.153735294	15.37352941
2	0.323264706	32.32647059
3	0.267970588	26.79705882
4	0.3755	37.55
5	0.315470588	31.54705882
6	0.355029412	35.50294118
7	0.397794118	39.77941176
8	0.369029412	36.90294118
9	0.366323529	36.63235294
10	0.351529412	35.15294118
11	0.359029412	35.90294118
12	0.372882353	37.28823529
13	0.380705882	38.07058824
14	0.329764706	32.97647059
15	0.346117647	34.61176471
16	0.382441176	38.24411765
17	0.394970588	39.49705882
18	0.376529412	37.65294118
19	0.345676471	34.56764706
20	0.383411765	38.34117647

Table 5.4: Average accuracy for properties spouse

Vector Number	**Average Accuracy**	**Percentage**
1	0.014919355	1.491935484
2	0.021370968	2.137096774
3	0.024395161	2.439516129
4	0.029233871	2.923387097
5	0.030040323	3.004032258
6	0.025	2.5
7	0.029435484	2.943548387
8	0.031854839	3.185483871
9	0.031854839	3.185483871
10	0.032258065	3.225806452
11	0.030846774	3.084677419
12	0.033669355	3.366935484
13	0.03125	3.125
14	0.034475806	3.447580645
15	0.033266129	3.326612903
16	0.032862903	3.286290323
17	0.031854839	3.185483871
18	0.033870968	3.387096774
19	0.03266129	3.266129032
20	0.032459677	3.245967742

6 Discussion

As we can see from the results for properties capital and country we obtained around 82% average accuracy. We can observe the

7 Conclusion and Outlook

continuing

Bibliography

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.

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

Mikolov, Tomas et al. (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 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, 30 April, 2018

Unterschrift