Technische Universitat Dresden

International MSc Program in
Computational Logic (MCL)

Institute for Artificial Intelligence

# "Predicting relation targets of DBPedia properties using vector representations from word2vec"

*Student:*
Happy Rani Das

*Supervisor:*
Prof. Dr. Sebastian
Rudolph

April 9, 2018

**Abstract**

Vector representation of words has been learned by many method, word2vec is one of them and useful in many natural language processing and information retrieval. Though, surprising fact is that Vector representation of words have not been applied for Dbpedia data to aggregate semantic information stored in word embeddings to predict dbpedia properties . In this paper, I am focusing on how Vector representation of words from word2vec can be applied for Dbpedia properties in order to get missing information. The main goals of Predicting relation targets of DBPedia properties using vector representations from word2vec are to find corresponding resources from the Dbpedia and to apply Word2Vec technique to find missing information.

# 1 Introduction

## 1.1 Background

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. It first constructs a vocabulary from the training text data and then learns vector representation of words. There are two ways to represent word2vec model architecture 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) These word2vec model architecture are used in machine learning areas, natural language processing and advance research areas. [1].

Continuous bag-of-words (CBOW) and continuous skip-gram model architecture are very popular nowadays in the machine learning areas and further research. Another depiction of words is dense vector came to know by word2vec. Dense vector have exceptionally been displayed to demonstrate same sense and it is beneficent in the immense range from data analytics to natural language processing.

As an example, words that have equivalent explanation will have analogous vectors because of cosine similarity and the words whose doesn't have equivalent explanation will have unalike vectors. It is quite surprising that, word vectors follow the likeness rule. For instance, presume the likeness

"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. [2].

## 1.2 Project Description

This project is focused on the Predicting relation targets of DBPedia properties using vector representations from word2vec. Word2Vec has been applied in several areas with the purpose of detect similarity and to find out nearest word. The main goal of this project is to introduce techniques that can be used for learning DBPedia data and to find out corresponding missing information 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 [defined by http://wiki.dbpedia.org/about]. If the user has two sentences like-

1. Berlin is the capital

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 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 Predicting relation targets of DBPedia properties using vector representations from word2vec is developed in python which takes DBPedia properties as input and returns nearest or similar missing information as output.

The rest of this report is methodized as follows. An overview of related work is discussed in Section 2 .Section 3 and 4 describe the methodology and architecture of the project. Section 5 discussed the results. We conclude the report with discussion and 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 look for the pattern matching stored in the RDF graph[5]. 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 [6]. 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

## 2.2 Related Work

# 3 Methodology

Figure 1 shows the flowchart diagram of our project. Dbpedia data is given as input which is then passed into pre-trained model GoogleNews-vectors-negative300.bin. then we 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 missing information 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.

## 3.1 Query DBPedia Properties for relations

This is most important and tricky part in this project. Properties is a connections between objects 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.

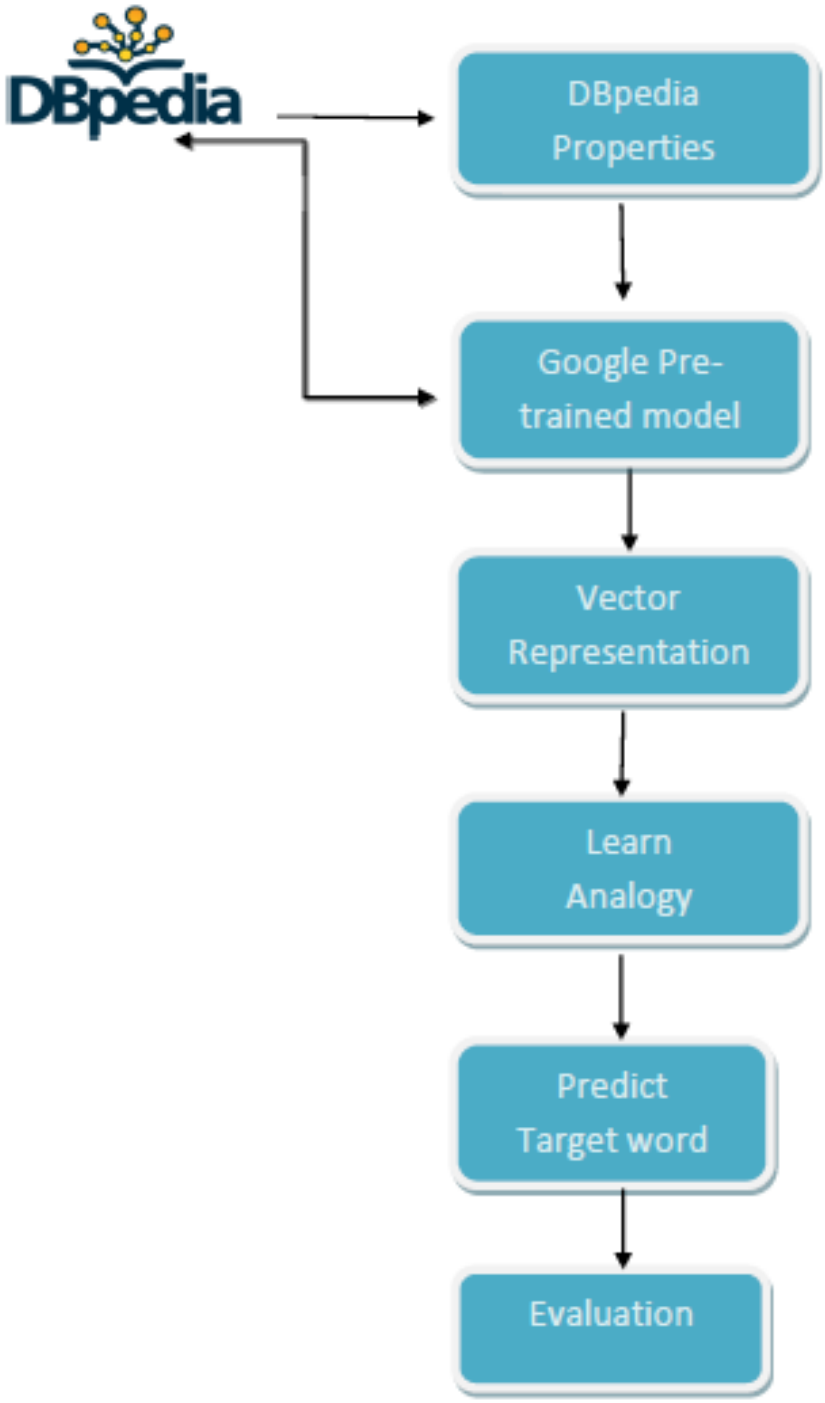




Figure 1: Figure 1: The General Scheme of Extracting Vector Representation of DBpedia Properties from Word2Vec

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
```

Table: 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 ;

5. dbo:country ?country .
```

| 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 1: Output for the input properties country and capital.

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:

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

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 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 Retrieved data clean

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 and his team. It contains 3 million words and phrases and from a Google News dataset they trained around 100 billion words [4]. We passed retrieved cleaned data into Google's pre-trained model. If the data is in the 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 by 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, 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. As an example, the following numpy vector is for the word Berlin:

## 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 Target word

It is also possible to predict target word 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 missing information $w_4$?. This is done by python. Presume, if Kigali is related to Rwanda, then Kabul is related to what? And the vector representation looks like:

SuperVector = vec["Kabul"] + vec["Rwanda"]
Target_Vector = vec.similar_by_vector((SuperVector - vec["Kigali"]), topn=3)

```
[ 0.141602156  0.251953312  0.026624512  0.000689509 -0.098144453 -0.156298555
 -0.206034688 -0.1464375  -0.1640625   0.0612793  -0.241211094 -0.0781125
  0.039794922 -0.115164406  0.017465605  0.065917969  0.133789006  0.253906625
 -0.1125      -0.146484436  0.124023344  0.186218906  0.048663086  0.079101562
  0.104003906  0.040039006 -0.223632812  0.199218875 -0.065917969  0.066589453
  0.139648444 -0.171875   -0.308593375 -0.025024441 -0.115234438 -0.151367188
  0.105339844  0.155820312  0.34375      0.073730047  0.03125     -0.049560055
 -0.123978556  0.146507781 -0.065986484  0.165039806 -0.104003906 -0.251953312
  0.049316641  0.259765562  0.013671855  0.064453512 -0.056152334 -0.134765562
 -0.243164406  0.149414406 -0.339843375  0.026000095 -0.3515625   -0.1171875
  0.077148544 -0.271484435 -0.073730047 -0.055419922  0.049560055 -0.251953312
  0.135742219  0.065986484 -0.231445531  0.069335594 -0.275390062  0.103027734
  0.025756854  0.122060547 -0.035400039 -0.0043335    0.320664062  0.271484435
  0.133789006 -0.130859938  0.061767756 -0.3359375    0.021850059 -0.034423363
  0.014709947  0.055419922  0.328125   -0.255156625 -0.0133495    0.155820312
 -0.15625     -0.035858867 -0.2109375  -0.051757781 -0.4609375  -0.035156625
  0.110351156 -0.221679969 -0.404296855 -0.127929969 -0.005859145 -0.116211094
  0.124511172  0.238281125 -0.051025539 -0.038550078 -0.150390062 -0.2265625
 -0.358715575 -0.213867119  0.0201416  -0.150390062  0.361328512  0.102539006
  0.195242219 -0.253906625 -0.0534668  -0.203125     0.1328125    0.174804669
 -0.229492219 -0.196289006  0.134765562 -0.051269853  0.171875   -0.253906625
 -0.066535935 -0.133789006 -0.314453512 -0.031127793 -0.063964854  0.09375
 -0.058056641  0.083007781 -0.189338935 -0.225585594  0.065917969 -0.0859375
 -0.012451117  0.177734435  0.052251953  0.166015562 -0.066406625 -0.1464375
  0.095214454 -0.171875     0.21875      0.296875   -0.259765562 -0.066894141
  0.005463359  0.108539844  0.255320312  0.002586365  0.0703125   0.094726556
  0.005859472  0.206054669  0.203125     0.124023344 -0.020385574 -0.031982242
  0.159179668 -0.074218575  0.1464375    0.131835594 -0.119628891  0.055566719
 -0.015331982  0.298828812 -0.222070312 -0.247070031  0.212890062  0.066445312
 -0.207031125 -0.2734375  -0.033223656 -0.308664062 -0.159179668 -0.053222366
  0.1875      -0.15625    -0.119140062 -0.251953512  0.34375      0.169921855
  0.1796875  -0.027832203  0.0201416    0.241211094  0.177734435  0.248046855
  0.101074422  0.212890062 -0.038085594  0.1875       0.247070031 -0.159465312
 -0.144531125 -0.066894406  0.098144453  0.025556445  0.155320312  0.011962504
  0.067767109 -0.2890625    0.041259877  0.078101562 -0.221679969 -0.140625
 -0.298875   -0.137695531  0.052246609  0.166992219 -0.3125      -0.009333838
  0.3046875   0.166992219  0.156532344 -0.09375      0.117675775 -0.229492219
 -0.174804669  0.004394453 -0.244140062  0.052978552 -0.134765562  0.002975346
 -0.25        -0.205078512  0.021972266  0.078101562 -0.231445531 -0.044921855
  0.1125      -0.146484438 -0.115234438  0.116210094  0.009897217 -0.116210094
  0.2656625    0.029052273 -0.090820031  0.127929969 -0.162109838  0.3125
 -0.090820031 -0.09375    -0.044433359  0.0048166  -0.146484438  0.0213623
 -0.233398544  0.054986084  0.100351562  0.150390062  0.180429969 -0.0078125
  0.053369609  0.216796855  0.161132251  0.205000781  0.090820031  0.147460094
  0.048614258  0.357421855  0.152617219 -0.051757781 -0.162109838  0.263671855
 -0.021728552 -0.172851196  0.227539006 -0.108886672  0.134765562 -0.063476656
  0.008713357  0.196289006  0.032958985 -0.0534668   -0.200195331  0.212890062
 -0.073730047 -0.131835394 -0.253906625  0.25781125   0.059326617  0.057617719
 -0.041900391 -0.006835662 -0.291015562  0.027832203 -0.223632251  0.123046855
```

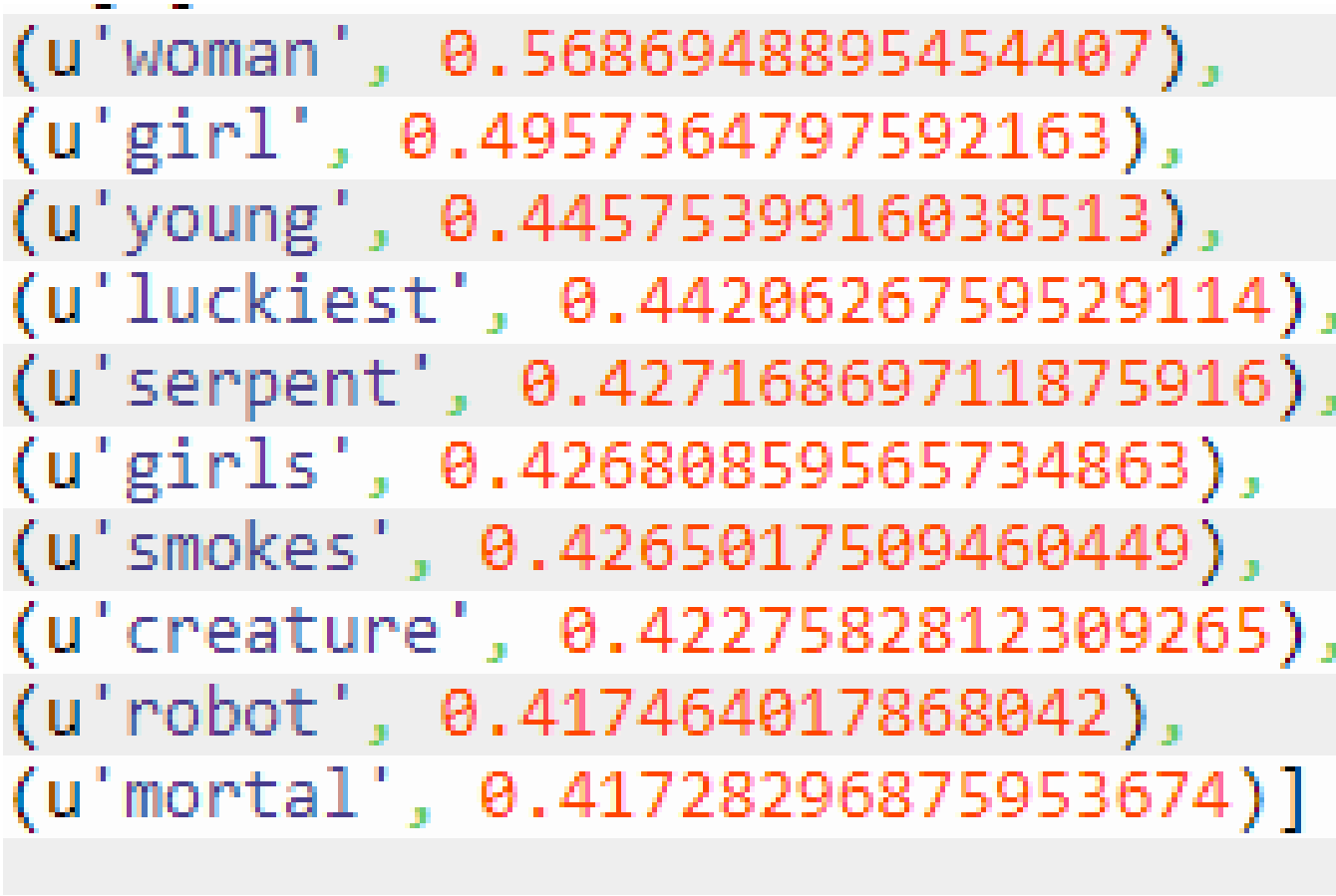


Figure 3: Figure 3: Most similar word of man

### 3.7 Evaluation

After retrieving cleaned data we sent them into Google's pre-trained model and found filtered data which is require for our next steps. We divided the filtered data into two parts:

1. For training

2.For testing.

Suppose for DBPedia properties country and capital we have taken half of the filtered data for training and another half for testing.

## 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

**References**

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

[2] http://www.1-4-5.net/ dmm/ml/how_does_word2vec_work.pdf

[3] viewexportask othersask others Philipp Heim, Sebastian Hellmann, Jens Lehmann, Steffen Lohmann, Timo Stegemann: RelFinder: Revealing Relationships in RDF Knowledge Bases. SAMT 2009: 182-187

[4] http://mccormickml.com/2016/04/12/googles-pretrained-word2vec-model-in-python/

[5] Mohamed Morsey, Jens Lehmann, Sören Auer, Claus Stadler, Sebastian Hellmann:DBpedia and the live extraction of structured data from Wikipedia. Program 46(2): 157-181 (2012)

[6] RDF Working Group. Resource Description Framework (RDF). https://www.w3.org/RDF/.2014-02-25 (accessed April 15, 2017).