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 17, 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, we are 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 aggregate DBpedia properties contained in Word2Vec model, find corresponding resources and to apply Word2Vec technique to predict missing information.

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 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.

Words that have equivalent meaning 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 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. [2].

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 a enormous open linked data and data source, extract structured information 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 target information. 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, a Super_vector is made by aggregating 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 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 intension 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 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 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

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 clustering, information retrieval, text classification and so on [7].

2.2 Related Work

Vector representation of word has gained enormous attention in the field of machine learning. Several techniques are introduced to get word vector. But word2vec has gained tremendous popularity. Aneesh Joshi proposed a method ”Learn Word2Vec by implementing it in tensorflow” [8]. In his method, he explained how word2vec works.

According to him[8], The idea behind word2vec is that:

1) Take a 3 layer neural network. (1 input layer + 1 hidden layer 1 output layer)

2) Feed it a word and train it to predict its neighbouring word.

3) Remove the last (output layer) and keep the input and hidden layer.

4) Now, input a word from within the vocabulary. The output given at the hidden layer is the ‘word embedding’ of the input word.

Here is an example proposed by Chris McCormick [9]. He trained the neural network by feeding it input output pair, such that for input word it predicts nearest words. He consider the window size of 2 for his example. It means 2 word before and after of that word.

Chris McCormick [9] proposed some of the training samples taken from the sentence “The quick brown fox jumps over the lazy dog.” and it is captured in figure: 1

Here is an amazing post by Andy Thomas [10]. In his Word2Vec word embedding tutorial in Python and TensorFlow post he describes ”Why do we need Word2Vec?” He explained it in the following way:-

If we want to feed words into machine learning models, we need to con-

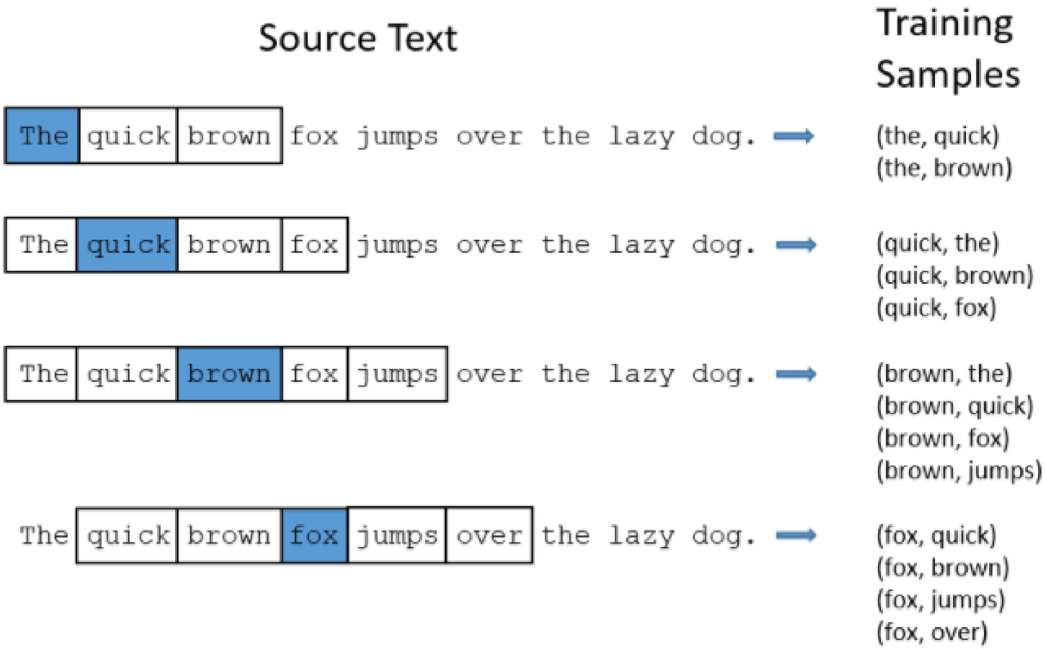
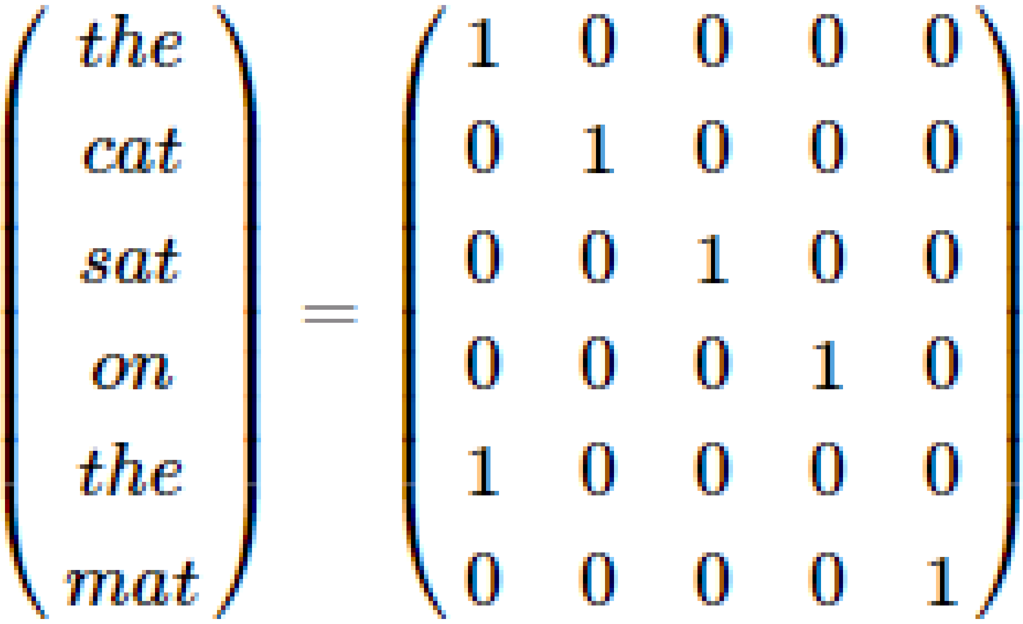


Figure 1: A training sample generation



$$\begin{pmatrix} the \\ cat \\ sat \\ on \\ the \\ mat \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 2: Vector Representation of “the cat sat on the mat”

vert the words into some set of numeric vectors. A simple way of doing this would be to use a “one-hot” method to convert the word into a sparse representation with only one element of the vector set to 1 to uniquely identify the word, the rest being zero.

If the sentence is “the cat sat on the mat” we would get the vector representation captured in figure:2:-

Andy Thomas [10] has transformed a six word sentence into a 6×5 matrix, with the 5 being the size of the vocabulary (“the” is repeated).

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.

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
```

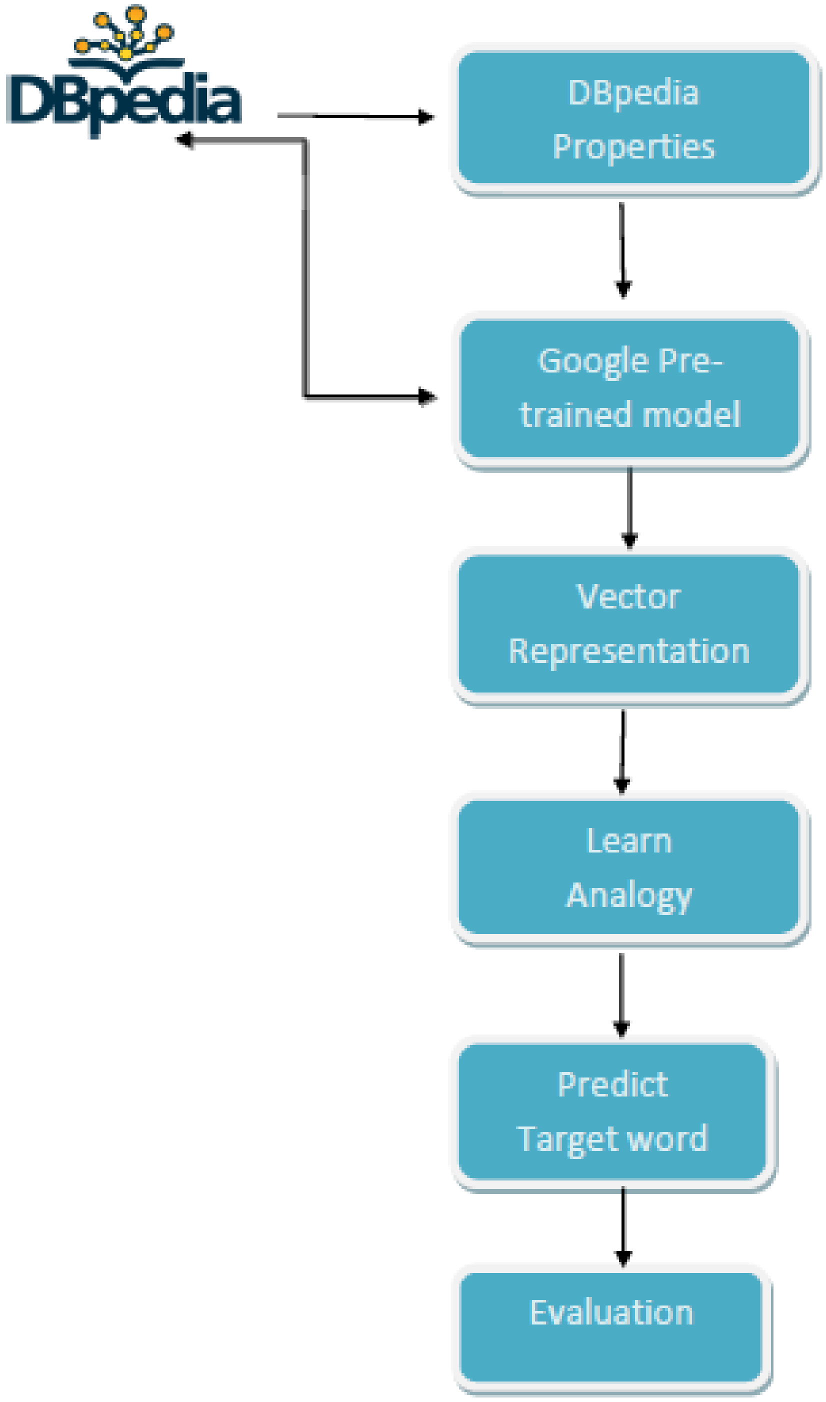
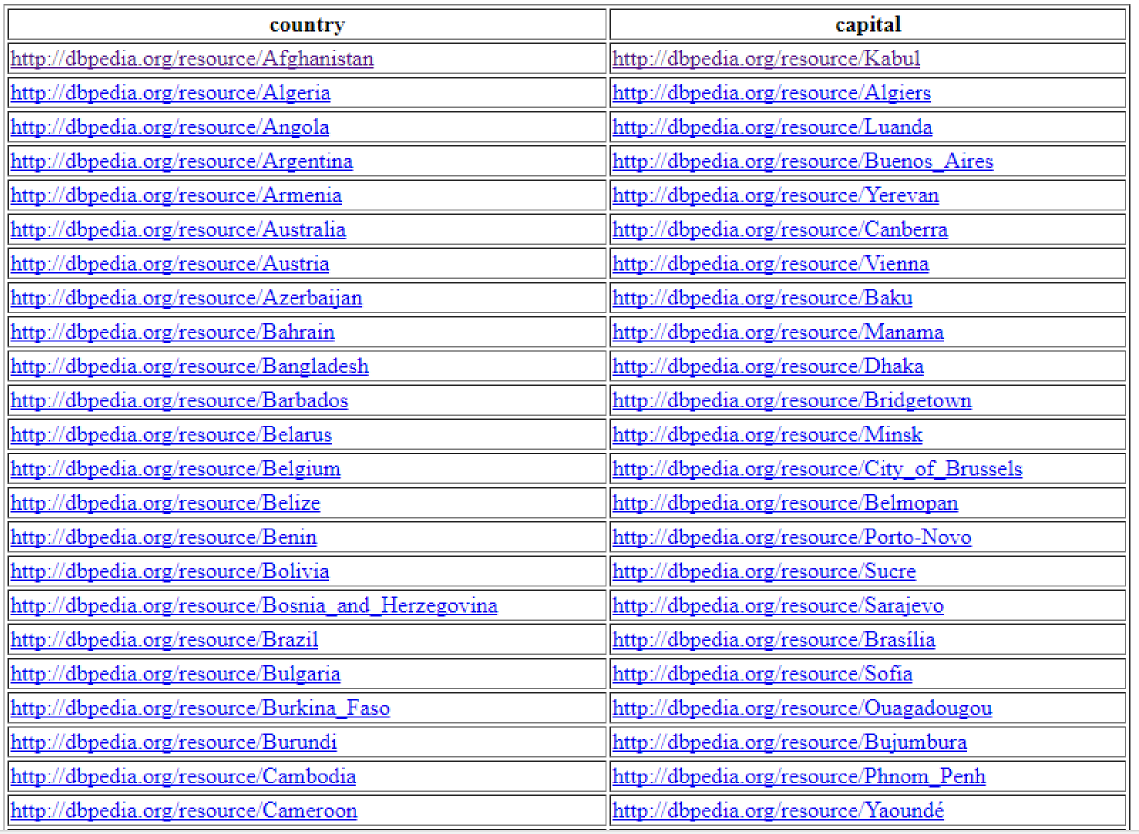


Figure 3: The General Scheme of Extracting Vector Representation of DBpedia Properties from Word2Vec



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.

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 .

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:

```
SELECT DISTINCT ?person ?party WHERE
{
?person rdf:type dbo:Person .
?person dbo:party ?party.
} order by ?person
offset 43880 .
```

Query: 4

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

```
SELECT DISTINCT ?x ?y WHERE
{
?x dbo:spouse ?y.
?y dbo:spouse ?x.
} 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

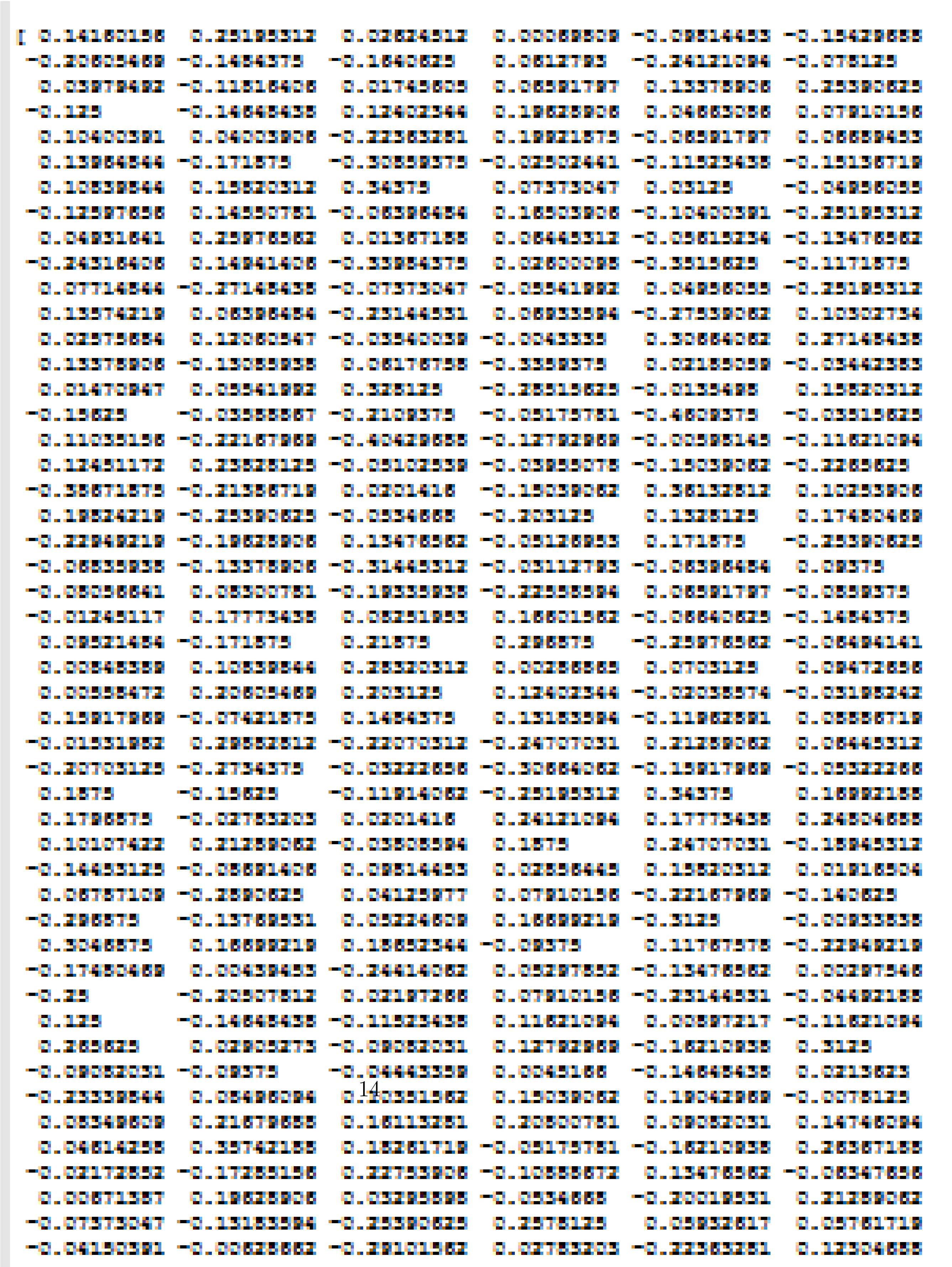
Representation of vector for word rely on by many technique such as GloVe, Word2Vec and so on. We consider Word2Vec in order to get vector representation of word. The concept behind word to vector has many advantages. For example, it is possible to do matrix addition and subtractions and semantic likeliness (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

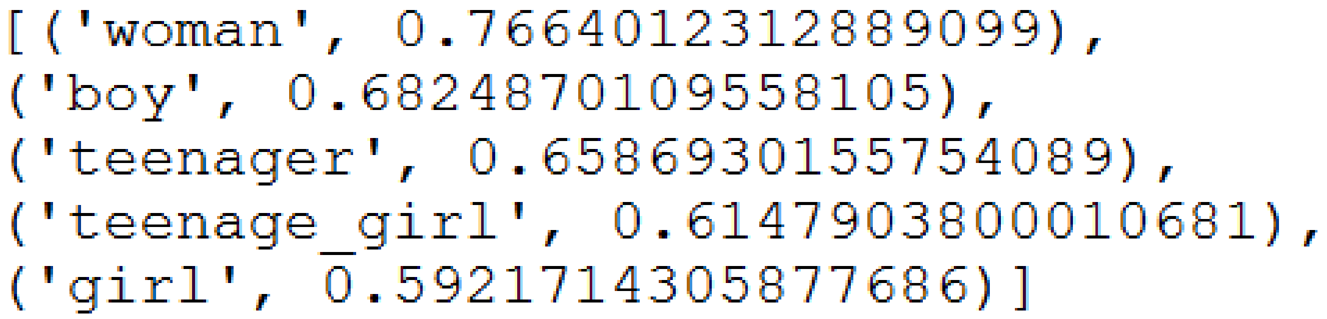
One of the advantages of vector representation of word is similarity prediction.It's possible to find out the most similar word and their distance by using word vector.Suppose, if we enter the man as an input it return the following words and corresponding distance from Google's pre-trained model as 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:



```
[ 0.14160156  0.25195312  0.02624512  0.00069809 -0.09814453 -0.15429688
 -0.20805469 -0.1484375  -0.1640625   0.0612793  -0.24121094 -0.078125
  0.03979492 -0.11816406  0.01745605  0.06591797  0.13378906  0.25390625
 -0.125      -0.14648438  0.12402344  0.19628906  0.04663086  0.07910156
  0.10400391  0.04003906 -0.22363281  0.19921875 -0.06591797  0.06689453
  0.13964844 -0.171875   -0.30859375 -0.02502441 -0.11523438 -0.15136719
  0.10839844  0.15820312  0.34375     0.07373047  0.03125    -0.04956055
 -0.12597656  0.14550781 -0.06396484  0.16503906 -0.10400391 -0.25195312
  0.04931641  0.25976562  0.01367188  0.06445312 -0.05615234 -0.13476562
 -0.24316406  0.14941406 -0.33984375  0.02600098 -0.3515625  -0.1171875
  0.07714844 -0.27148438 -0.07373047 -0.05541992  0.04956055 -0.25195312
  0.13574219  0.06396484 -0.23144531  0.06933594 -0.27539062  0.10302734
  0.02575684  0.12060547 -0.03540039 -0.0043335   0.30664062  0.27148438
  0.13378906 -0.13085938  0.06176758 -0.3359375   0.02185059 -0.03442383
  0.01470947  0.05541992  0.328125   -0.28515625 -0.0135498   0.15820312
 -0.15625    -0.03588867 -0.2109375  -0.05175781 -0.4609375  -0.03515625
  0.11035156 -0.22167969 -0.40429688 -0.12792969 -0.00598145 -0.11621094
  0.12451172  0.23828125 -0.05102539 -0.03955078 -0.15039062 -0.2265625
 -0.35671875 -0.21386719  0.0201416  -0.15039062  0.36132812  0.10253906
  0.19824219 -0.25390625 -0.0534668  -0.203125    0.1328125   0.17480469
 -0.22949219 -0.19628906  0.13476562 -0.05126953  0.171875   -0.25390625
 -0.06835938 -0.13378906 -0.31445312 -0.03112793 -0.06396484  0.09375
 -0.08056641  0.08300781 -0.19335938 -0.22558594  0.06591797 -0.0859375
 -0.01245117  0.17773438  0.08251953  0.16601562 -0.06640625 -0.1484375
  0.09521484 -0.171875    0.21875     0.296875   -0.25976562 -0.06494141
  0.00848389  0.10839844  0.28320312  0.00286865  0.0703125   0.09472656
  0.00558472  0.20605469  0.203125    0.12402344 -0.02038574 -0.03198242
  0.15917969 -0.07421875  0.1484375   0.13183594 -0.11962891  0.08886719
 -0.01531982  0.29882812 -0.22070312 -0.24707031  0.21289062  0.06445312
 -0.20703125 -0.2734375  -0.03222656 -0.30664062 -0.15917969 -0.05322266
  0.1875     -0.15625    -0.11914062 -0.25195312  0.34375     0.16992188
  0.1796875  -0.02783203  0.0201416   0.24121094  0.17773438  0.24804688
  0.10107422  0.21289062 -0.03808594  0.1875      0.24707031 -0.18945312
 -0.14453125 -0.08691406  0.09814453  0.02856445  0.15820312  0.01916504
  0.06787109 -0.2890625   0.04125977  0.07910156 -0.22167969 -0.140625
 -0.296875   -0.13769531  0.05224609  0.16699219 -0.3125     -0.00933838
  0.3046875   0.16699219  0.15652344 -0.09375     0.11767578 -0.22949219
 -0.17480469  0.00439453 -0.24414062  0.05297852 -0.13476562  0.00297546
 -0.25       -0.20507812  0.02197266  0.07910156 -0.23144531 -0.04492188
  0.125      -0.14648438 -0.11523438  0.11621094  0.00897217 -0.11621094
  0.265625    0.02905273 -0.09082031  0.12792969 -0.16210938  0.3125
 -0.09082031 -0.09375    -0.04443359  0.0045166  -0.14648438  0.0213623
 -0.23339844  0.08496094  0.10351562  0.15039062  0.19042969 -0.0078125
  0.08349609  0.21679688  0.16113281  0.20800781  0.09082031  0.14746094
  0.04614258  0.35742188  0.18261719 -0.05175781 -0.16210938  0.26367188
 -0.02172852 -0.17285156  0.22753906 -0.10888672  0.13476562 -0.06347656
  0.00671387  0.19628906  0.03295898 -0.0534668  -0.20019531  0.21289062
 -0.07373047 -0.13183594 -0.25390625  0.2578125   0.05932617  0.05761719
 -0.04150391 -0.00628662 -0.28101562  0.02783203 -0.22363281  0.12304688
```



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

Figure 5: Most similar word of man

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

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 randomly half of the filtered data for training and another half for testing. We have tasted all of the testing data against training data in Word2Vec model. Running

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

Continuing

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).

[7] Tommaso Teofili: par2hier: towards vector representations for hierarchical content. ICCS 2017: 2343-2347

[8] https://towardsdatascience.com/learn-word2vec-by-implementing-it-in-tensorflow-45641adaf2ac

[9] http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

[10] http://adventuresinmachinelearning.com/word2vec-tutorial-tensorflow/