Technische Universitat Dresden

International MSc Program in
Computational Logic (MCL)

Institute for Artificial Intelligence

Extracting Vector Representation Of DBPedia Properties from Word2Vec

Student:
Happy Rani Das

Supervisor:
Prof. Dr. Sebastian
Rudolph

October 29, 2017

Abstract

Vector representation of words has been learned by word2vec and useful in many natural language processing and information retrieval. Though, surprising fact is that word2vec have not been applied for Dbpedia data. In this paper, I am focusing on how word2vec can be applied for Dbpedia properties in order to get missing information. The main goals of Extracting Vector Representation of Dbpedia Properties 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 output. It first constructs a vocabulary from the training text data and then learns vector representation of words. Word2Vec tool provides an efficient implementation of the continuous bag-of-words (predicts a missing word given a window of context words or word sequence) and skip-gram (using a word to predict a target context) architectures for computing vector representations of words. These representations can be subsequently used in many natural language processing, machine learning applications and for further research [1].

The word2vec model and its applications have recently attracted a great deal of attention from the machine learning community. The dense vector representations of words learned by word2vec have remarkably been shown to carry semantic meanings and are useful in a wide range of use cases ranging from natural language processing to network flow data analysis. For instance, words that we know to be synonyms tend to have similar vectors in terms of cosine similarity and antonyms tend to have dissimilar vectors. Even more surprisingly, word vectors tend to obey the laws of analogy. For example, consider the analogy "Woman is to queen as man is to king". It turns out that

$$v_{queen} - v_{woman} + v_{man} = v_{king}$$

where $v_{queen}; v_{woman}; v_{man} and v_{king}$ are the word vectors for queen, woman, man, and king respectively. [2].

1.2 Project Description

This project is focused on the Extracting Vector Representation of DBPedia Properties from word2vec. Word2Vec has been applied in several areas with the purpose of detect similarity and to find out nearest word. However, Word2Vec has not been applied for DBPedia properties. 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 (from "DB" 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 Extracting Vector Representation of DBPedia Properties from word2vec is developed in java which takes DBPedia properties as input and returns a nearest or similar missing information as output.

2 Related Work

3 System Architecture

Figure 1 shows the flowchart diagram of our project.

3.1 Properties Extraction

This is most important and tricky part in this project. Properties is a connections between objects in DBPedia. In order to get properties first we need to find out how to get objects or resources from DBPedia. To get required objects, the system first takes text entered by the user and returns a set of

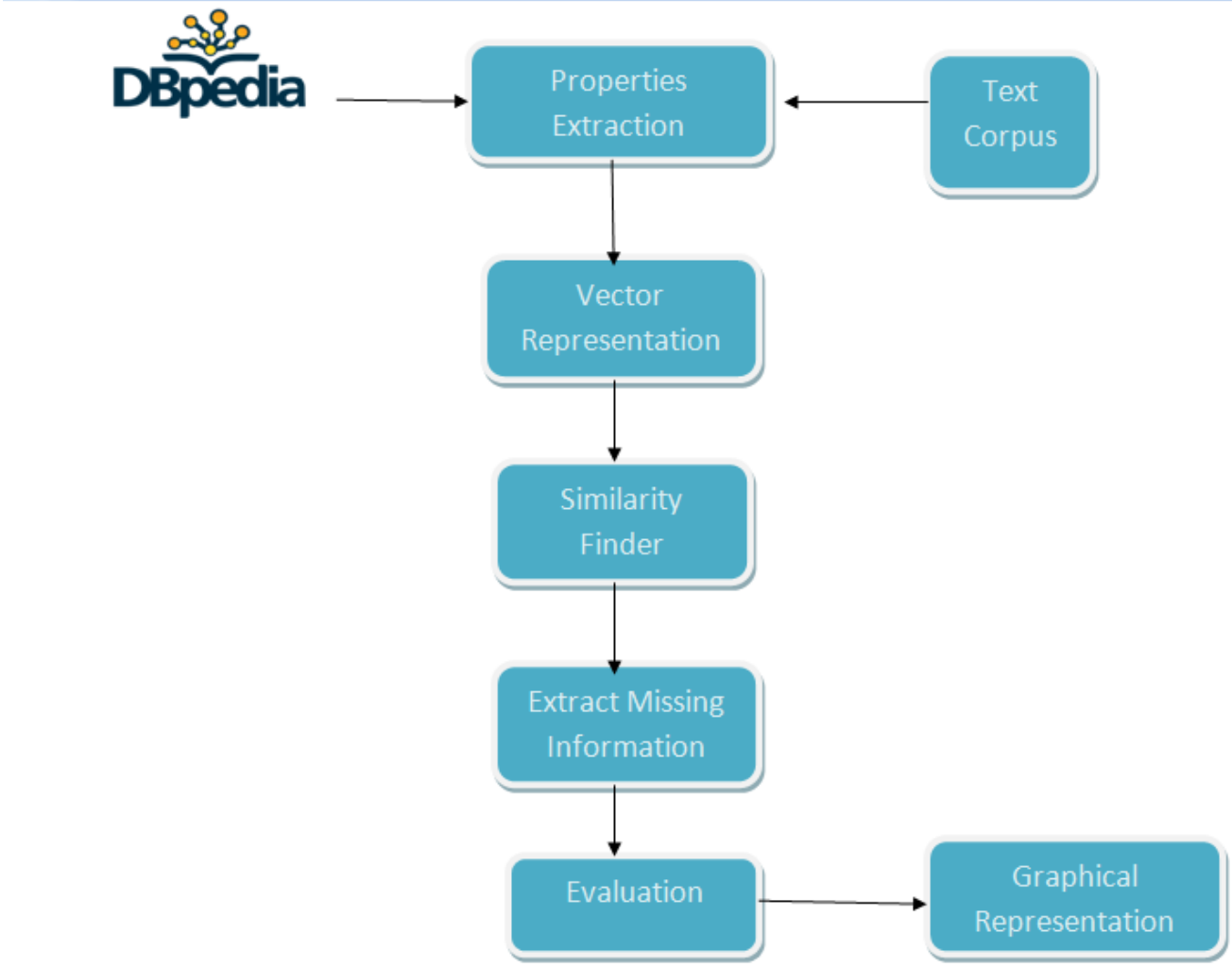

Figure 1: Figure 1: The General Scheme of Extracting Vector Representation of DBpedia Properties from Word2Vec

corresponding objects or resources as output. The objects entered by the user retrieves the total number of incoming and outgoing edges and similar URIs which contains at least one of the input words. The following code shows how to execute SPARQL query against DBpedia for the user input 'Barack_Obama':

1. SELECT DISTINCT ?resource ?l count(?resource) as ?Totaledge WHERE

2.{

3. ?obj ?p ?resource .

4. ?resource <http://www.w3.org/2000/01/rdf-schema#label >?l .

5. ?l bif:contains '"Obama"' .

6. FILTER (!regex(str(?resource), '^ http://dbpedia.org/resource/Category:')).

7. FILTER (!regex(str(?resource), '^ http://dbpedia.org/resource/List')).

8. FILTER (!regex(str(?resource), '^ http://sw.opencyc.org/')).

9. FILTER (lang(?l) = 'en').

10.FILTER (!isLiteral(?obj)).

11.} ORDER BY DESC(?Totaledge) LIMIT 15

Properties Extraction process consists of different types of SPARQL queries searches for re- lationships between the given objects entered by user and returns all the relationship based on the user input. Properties Extraction procedure tries to find out all of the relationships between two objects in a DBPedia dataset by first generating a list of SPARQL queries.There are two types of SPARQL queries-

1. One - way and 2. Multi - way

One - way query type can be divided into two types-

1. One - way Forward: x → .... → y

One - way forward tries to find out those relationship which starts from the node x and ends at node y and it consider other nodes/(subjects or objects) and edges/predicates where x should be the leftmost node and y should be the right most node and the edge should be come out from the left node x [3].

The following SPARQL query searches for relationships of the objects Barack_Obama and Michelle_Obama

1.SELECT * WHERE

2. {

3. dbr : Barack_Obama ?rel db r: Michelle_Obama

4. FILTER ((?rel!=rdf:type) &&

5. (?rel!=skos:subject) && newline

6. (?rel!=dbp:wikiPageUsesTemplate) &&

7. (?rel!=dbp:wordnet_type) &&

8. (?rel!=dbr:Barack_Obama) &&

9. (?rel!=dbr:Michelle_Obama)

10.).

11. } LIMIT 10

2. One - way backward: x ← . . . . ← y

One - way backward is the opposite of one - way forward, which tries to find out those relationship which starts from the node y and ends at node x and it consider other nodes/(subjects or objects) and edges/predicates where

x should be the leftmost node and y should be the right most node and the edge should be come out from the right most node y.

The following SPARQL query applies one - way backward procedures in order to get the relationships .
1.SELECT * WHERE

2. {

3. dbr : Michelle_Obama ?rel db r: Barack_Obama

4. FILTER ((?rel!=rdf:type) &&

5. (?rel!=skos:subject) && newline

6. (?rel!=dbp:wikiPageUsesTemplate) &&

7. (?rel!=dbp:wordnet_type) &&

8. (?rel!=dbr:Barack_Obama) &&

9. (?rel!=dbr:Michelle_Obama)

10.).

11. } LIMIT 10

2. Multi - way

$$x \rightarrow .... \rightarrow z \leftarrow . \ . \ . \ . \leftarrow y$$

$$x \leftarrow . \ . \ . \ . \leftarrow z \rightarrow .... \rightarrow y$$

in multi - way relationship there is an object z in between such that there is a one-way relationship each from x and from y to z or from z to x and y and z is unknown at first but found within the searching process [3].

The following SPARQL query applies multi- way procedures in order to get the relationships-

```
1.SELECT * WHERE

2. {

3. dbr:Barack_Obama ?sub ?prop .

4. dbr:Michelle_Obama ?obj ?prop.

5. FILTER ((?obj !=rdf:type) &&

6. (?obj !=skos:subject) &&

7.(?obj !=dbp:wikiPageUsesTemplate) &&

8. (?obj !=dbp:wordnet_type ) &&

9. (!isLiteral(?prop)) &&

10. (?obj !=dbr:Barack_Obama ) &&

11. (?obj !=dbr:Michelle_Obama ) &&

12. (?prop !=dbr:Barack_Obama ) &&

13. (?prop !=dbr:Michelle_Obama )

14. ) .

15. } LIMIT 20
```

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