Ontology Matching with word2vec

Harmen Prins

February 23, 2016

Summary

Contents

1 Introduction

2 Problem

2.1 Short Context

On the Semantic Web, ontologies are used to define and share knowledge.

2.2 Problem

When knowledge bases are combined, their ontologies need to be merged.

2.3 Research questions

2.4 Hypotheses

2.5 Justification

Reliable, error proof ontology matching will be key for the distributed Semantic Web.

3 Context

In this section I will discuss all concepts that are required to understand the study.

3.1 The Semantic Web

The Semantic Web, or Web 3.0, is a concept of a Web that enhances the User Experience in ways that the current Web can not provide. Web 2.0, the current Web, models the internet as *hubs* (or sites) with a specific purpose and barely any interconnectivity. For example, Facebook is a hub for sharing content with people you consider friends, but if you want to share that content with more people you will have to go to another hub, like imgur or Pinterest. The Semantic Web, on the other hand, sees web pages as sources of *information* that can be combined whenever the user needs it. This allows not only humans to browse the internet, but also Artificial Intelligences, which will use this information to aid their users.

For example, if a user wants to see a movie this weekend, he orders his AI, or *softbot*, to find him a suitable time and movie to see. The softbot then accesses the Internet Movie Database to see which movies currently play and which of those are similar to what the user liked in the past. He will then access the pages of the local movie theaters to see when those movies play. Lastly, he will contact the softbots of the friends of the user to ask if they will also come to see that movie. After a short exchange the optimal time, place and participant

set are decided, and the softbot tells the user what the results are. When the user accepts, the appointment is automatically added to his agenda.

All of this is possible, if the softbot can access all of the information reliably. In the current Web, this is not possible, as all information is written in an ambiguous, unstructured format called human language. To allow softbots to access the same information that we can, the Semantic Web proposes to add a layer to the internet where all information is saved in a unified format that is unambiguous, robot-interpretable and can be used to store any type of knowledge.

3.1.1 Goal

The goal of the Semantic Web is very pragmatic: help people in everyday activities by leveraging all information available on the internet. The fact that it is pragmatic goes a long way of making it a reality: if even one application comes into existence that uses online information in a structured way and thus makes everyday tasks a little easier, the goal is accomplished. Of course, then the goal resets and we should make more applications leveraging more information for more tasks.

3.1.2 Current state

3.1.3 Technologies

3.2 Ontologies

3.2.1 OWL

3.2.2 Role

3.3 Current alignment strategies

Many different alignment strategies have already been developed. All strategies follow the same two-step approach. The two steps are independent and as such different methods can be used interchangeably. The first step is to generate an initial correspondence set, where correspondences between nodes are found based on just their labels and meta-data. The second step is to use this initial set and the structure of the ontology to find more correspondences. These steps are called the terminological and the structural steps. Some strategies also use extra steps like the extensional and semantic steps, which use vector space models and inference, respectively. https://hal.inria.fr/hal-00917910/document

Popular terminological strategies are WordNet comparison and edit distance. The former uses the popular WordNet hierarchy as a distance measure between to concepts, e.g. the number of edges between the concepts and their least general common superconcept. Edit distance is a purely string-based similarity measure. The similarity is a (weighted) count of edits required to transform one word into another, where common edits are insertion, deletion and replace-

ment. Similar methods include substring matching and n-gram matching, which compare parts of the strings to find similarities.

3.4 Problems with ontology alignment

3.5 Other related research

3.5.1 Word sense representation

Skip-gram models have been widely used to represent words as vectors. This is useful as it is easy to calculate the distance between two vectors, which is a useful property for ontology matching. This vector distance has been shown to be related to the semantic similarity between words. This semantic similarity is very useful for ontology alignment, as parts of one ontology have to be aligned with parts of another ontology depending on their semantic relation.

Recently, an effective and efficient skip-gram model has been developed. It is called word2vec, and uses a number of extensions over previous methods that enable it to efficiently learn an effective representation. Word2vec learns this representation by trying to predict a context matrix. First a one-hot encoding is made for the vocabulary. Then the model learns to predict a context matrix of C one-hot vectors based on the given one-hot input vector. The model is a neural network with one hidden layer and multiple output layers. Since the words are one-hot encoded the input to the hidden layer is equal to the row of the weight matrix that corresponds to the word. This row is the vector representation of the word.

Word2vec has been expanded in a number of different ways. For example, word order can be preserved, leading to a similarity measure that is closer to syntax, as syntax defines which words go where in a sentence. This research shows that the input does not have to be a bag-of-words representation, which will be useful later on. Another interesting development is the multi-sense word2vec, which has multiple vector representations per word, depending on the number of different definitions a word has. For example, the word bank represents both the monetary institute and the riverside, which would both have a different representation in this extension. It can differentiate between two different meanings based on the context. This will help with resolving ambiguity, a very important factor in ontology alignment.

3.6 Used technology

3.6.1 Ontology constructor

3.6.2 AgreementMaker

AgreementMaker is an ontology matching system which obtained the highest F-measure in 6 of the 7 ontology matching tracks of OAEI 2015

The system uses an extensible framework which allows its users to add new modules to the system. This is very useful for research as it allows the researcher

to compare different modules by swapping in just those modules in the framework. The module-swapping technique will also be used in this work, but more of that will be discussed in *Evaluationmethods*.

4 Method

In this section, I will describe the method that I developed. Firstly, I will describe the general idea. In the second section, I will give a theoretical justification for why this method should work and how it works conceptually. After that I will describe the different ways that the input graphs can be used to train the word2vec model and create the alignment.

4.1 Idea

As explained earlier, word2vec is a model that learns two things: implicitly it learns the semantic vector representation of words, and explicitly it learns to predict words from their context or vice versa. I will use both of these pieces of information in our alignment algorithm. The semantic representation will be used for finding nodes in the two ontology graphs that are similar, i.e. close in the vector space. This step is relatively straightforward if proper vector representations are found. However, this may be hard due to a number of problems which I have listed below.

- The problem I am trying to solve is the problem of differently labelled nodes referring to the same concept. For example *writer* and *author* will not be matched by a string matcher, but should be matched. The two concepts may also have different labels in their context even though their contexts refer to the same concepts.
- The size of the training corpus should be large enough for proper representation to be learned for every concept. It should also be relevant to the labels that are in the ontologies. This is a problem since the ontologies may be relatively small, so they need to be extended while keeping relevance.
- Some labels may be ambiguous, for example homonyms, which are concepts that have the same label but represent a different context.

To solve these problems I will adapt word2vec to graphs and extend it with multi-sense embeddings. Also hot-starting is considered as an improvement.

4.2 Theory

4.3 Possibilities

4.3.1 Training corpus

Word embedding models need to be trained on a large corpus. These corpora need to cover the concepts that are contained in the ontology, but also need to be large enough to build good embeddings. Since the ontologies themselves do not necessarily contain enough examples to embed the concepts properly, I need to use ontologies that are likely to contain the concepts I want to align. As a benchmark, I will use the WordNet and NELL ontologies. To check if adding the ontologies that are to be aligned helps, I will also train a model on an ontology consisting of WordNet, NELL and the two alignment candidates.

4.3.2 Node vectorisation

The most basic node2vec model converts a node to a vector purely based on its own label. This method has one advantage over simple string matching: if no node is found in the other ontology that matches exactly, we can still find a node that is similar since it is close in the vector space. Therefore, this method should already be an improvement over the most basic string matching algorithms. It may even be competitive with more advanced string matchers (that look at substrings of labels). For example, *ear lobe* and *ear* will be matched by a substring matcher, but are also semantically similar since they will often be mentioned in the same context.

4.3.3 Combine with neighbours

To improve the model, we can add context information from the neighbours of a node. The most basic context adaptation would be to create a vector as the average of the context vectors. The resulting vector should be combined with the node vector of the node that is being investigated. This can be done by averaging or weighted averaging where the context is weighed more if there are more neighbours, though not necessarily linearly. These possibilities were also investigated.

4.3.4 Select from senses

Another way to take into account the influence of neighbouring nodes is to train a multi-sense node embedding model as described in appropriate section, and selecting from the different node senses based on which is closest to the context average.

4.3.5 Adding edge labels

To add more information to the model, edge labels can also be added to the context of a node. This method effectively doubles the training data and context

size when creating the context average. This should make the model more robust, although edge labels may be duplicate (one has many unique family members) and less informative in general, so a lower weight may be appropriate. Another problem with this method is that the relation between a neighbour node and its corresponding edge is lost, since they are just treated as independent contexts.

4.3.6 Select from senses

This method is exactly the same as the selecting from node senses with just the neighbour nodes except that you now also take into account the edge labels.

4.3.7 Combining edges with nodes

As can be read in appropriate section,