INSA de Lyon - Département Informatique / Universität Passau PhDTrack

Semantic extraction from natural language using deep learning and Linked Open Data

GARCHERY Mathieu
mathieu.garchery@insa-lyon.fr

Tutor
LAPORTE Léa
lea.laporte@insa-lyon.fr

January, 2016

Abstract

Deep learning techniques such as neural networks have lately proven to be very efficient in several research fields, including natural language processing. In this paper, we focus on semantic relations (similarities and analogies) extraction from text in natural language. With this in mind, we compare different language models, ranging from classic probabilistic models to recent neural network language models, and other deep learning techniques for natural language processing. We focus on distributed word representations based on context-predictive approaches (also called word embeddings). Then, we take an inventory of existing software for word embeddings and present Linked Open Data resources that can be used for semantic extraction. Finally, we present SemanticExtractor, a program based on word2vec for recognizing entities from natural text, extracting analogies and characterizing their relations using DBpedia.

Contents

1 Introduction

Emergence of deep learning Machine learning, a branch of artificial intelligence, aims to build systems capable of learning and predicting patterns from data. Recently, deep learning (a subdivision of machine learning) has become more and more popular, because of its ability to automatically determine which properties of input data are relevant to the given classification task. Thus, less domain-specific knowledge and time-consuming engineering are needed. Furthermore, deep learning approaches can also be seen as using computer-understandable - instead of human-readable - representations. One other advantage of deep learning methods is their scalability : unlike most algorithms, huge datasets can often be processed smoothly and make the models even stronger and more accurate.

Natural language processing (NLP) For these reasons, deep learning methods are particularly well suited for natural language processing (NLP), whose objective is to make natural human (ambiguous) language understandable to computers. However, this is still a distant goal, as there is no consensus on the best representations to use to achieve such human-computer interactions[Collobert et al., 2011]. In the meantime, the very large domain of NLP has been divided into many subtasks to tackle this complex problem from different perspectives.

Semantic extraction from natural text In this paper, we will mainly focus on the NLP subtask of semantic relations extraction (also referred to as semantic extraction hereinafter), which consists in identifying entities or concepts and their semantic relations in natural text corpora. We are interested in two types of semantic relations : similarities (concepts with close signification, e.g. "car" and "vehicle") and analogies (entity pairs corresponding to the same conceptual relation, e.g. "Paris is to France what Berlin is to Germany"). In the case of analogies, the nature of the conceptual relation in question can also be characterized (e.g. "capital city" in the previous example). Although semantic extraction will be our main topic, other related or underlying NLP tasks such as named entity recognition (NER) and semantic role labeling (SRL) will also be mentioned. Moreover, we will also investigate how knowledge databases and other resources from the Linked Open Data Cloud (LODC) can be used for semantic extraction. As a practical application to this state-of-the-art survey, we propose and evaluate SemanticExtractor, our tool for analogies discovery and characterization in natural text.

Organization of the paper This paper is organized as follows. In section 2, we present prior NLP models (language models, co-occurrence count-based models and semantic role labeling systems), whose existence and functioning are necessary to understand later deep learning models. In section 3, we explain why deep learning is adapted to NLP, before presenting neural network language models (NNLM) and word embeddings for semantic extraction. In section 4, we take a survey of existing word embeddings implementations. In section 5, we describe several Linked Open Data resources useful for semantic extraction. In section 6, we present SemanticExtractor, our tool to extract and characterize analogies from natural text, and evaluate it. Finally, in section 7 we draw a conclusion of our state-of-the-art survey and practical application results on semantic extraction.

2 Models for semantic extraction from natural language

2.1 Probabilistic language models

Although it is not their main purpose, language models are important in the domain of semantic extraction from natural language. Recently, probabilistic language models have been seriously challenged by neural network language models (NNLMs) and other deep learning approaches (see section 3). However, many concepts used in NNLMs come from or where influenced by probabilistic language models. Here we take a review of these models, their importance in language processing and their link to semantic extraction from text. According to our objective, we focus on language models for NLP only (language models for information retrieval are out-of-scope).

2.1.1 Language modeling : definition and applications

Definition A language model is a statistical model that assigns a probability to a word sequence. It is generally built on a specific natural language corpus, and its distribution corresponds to the conditional probability of the corresponding word sequences to be generated by the corpus. Implicitly, such a probability should be representative of the syntactic, grammatical and semantic correctness of the input word sequence. Thus, language models can be used to check whether a given word sequence is likely to appear in a corpus (and therefore to be correct and relevant), but also to generate new, probable word sequences [Collins, 2013, Hiemstra, 2009]

Applications Language models have been studied for decades and are used in many applications such as speech and handwriting recognition, part-of-speech tagging, syntactic and semantic analysis, information retrieval. In this paper, we focus on the comparison of different language models and their application to semantic extraction from text.

2.1.2 From basic probabilistic models to n-grams

A simple probabilistic model The simplest way to assign a probability to a word sequence in a corpus is to estimate the appearance frequency with its empirical average number (i.e. number of appearances divided by total number of sequences). First, we need to compute the number of occurrences of each separate word of the sequence and divide it by the total word count. This gives us a probability for each unique word. Assuming that words are independent from each other, the probability of the whole sequence is then given by the product of the single word probabilities.
Although such a model can be easily computed, it has several drawbacks. First of all, using the product of probabilities implies that the sequence will be rated with a zero probability if any of its single words does not appear in the corpus. There are very specific types of applications (for instance web search) where this might not be a problem, because query terms are (almost always) contained in the corpus [Hiemstra, 2009], but generally it is not the case. This issue will be discussed later in next section. Secondly, for simplification purpose we assumed that words are independent from each other in a sequence, which is both false and counter-intuitive. In reality, we cannot arbitrarily swap words in natural language without altering the sentence. Furthermore, we should rely on the context (i.e. previous words) to predict the most probable next word [Jurafsky and Martin, 2014].

n-grams The most common way to take into account the previous context words is to use so-called "n-grams" [Jurafsky and Martin, 2014]. Instead of counting all single word occurrences, n-grams language models rely on the occurrences of n-sized word tuples. For example, a bigram model ($n = 2$) is built on the occurrences count of all word sequences of size 2 (or less) appearing in the corpus : $c(w_1, w_2)$.
The probability of a word w_n knowing the previous word w_{n-1} is then given by :

$$p(w_n|w_n - 1) = \frac{c(w_{n-1}, w_n)}{c(w_{n-1})} \tag{1}$$

This formula can be generalized to other values of n :

$$p(w_n|w_{n-1},\dots,w_1) = \frac{c(w1,\dots,w_n)}{c(w_1,\dots,w_{n-1})} \tag{2}$$

As shown in the previous formula, n-gram models are based on the assumption that the likelihood of the word to predict depends on the $n-1$ previous words (called the Markov assumption). Thus, the following question appears: what is the optimal value for n ?
Like often in parameter optimization, this appears to a dilemma. A low value of n restricts the model on very local context dependencies, while a greater value of n might give more precise results, but also a much more complex model (the number of unique n-grams grows exponentially with n). When using n-grams for text generation, the greater n, the more coherent are the generated sequences in accordance with the corpus. But given that the number of possible n-grams increases exponentially, the relative proportion of actually seen n-grams decreases accordingly. With $n = 5$, the model becomes too specific: the generated sequences are not only similar to the corpus, they are mostly actual sequences from the corpus [McKeown, 2009]. Using bigrams or trigrams seems to be a good trade-off [Hiemstra, 2009, Jurafsky and Martin, 2014, Collins, 2013].

2.1.3 Improving n-grams

However, even bigrams or trigrams do not give very good results in their simple form explained above. As mentioned before, the problem of zero probability for unseen n-grams remains. A word sequence will be assigned a probability of zero by the model if it is not present in the corpus. This is a major problem, because due to the high number of word combinations, only a few n-grams are actually observed: the n-gram probability matrix is extremely sparse [Jurafsky and Martin, 2014]. Even with unigrams, there are huge probability differences between common and unusual word, because the word occurrence distribution follows Zipf's law[1] on most languages [McKeown, 2009].

Smoothing and discounting We can overcome this difficulty with probability smoothing. The basic idea of this technique is to change the probability distribution slightly, so as to flatten the probability distribution and remove

[1]Zipf's law is an empirical law stating that the appearance frequency of a word in natural language is inversely proportional to its frequency rank : $f_n = k/n^c$, where c is constant.

zero values for better generalization. The simplest way to do that is to consider that all n-grams are observed at least once in the corpus (i.e. to add 1 to the occurrence count for all n-grams), and to compute the word sequence probabilities like before. Of course, when computing the likelihood of a sequence, the occurrences counts must all be replaced with the new values, so as to guarantee that the result can still be interpreted as a probability. This method is called add-one smoothing, or Laplace smoothing (see equation (3)). Since it significantly improves language models, more complex smoothing functions have been studied, such as add-k (add k counts instead of 1, see equation(4)), linear interpolation (weighted average of $n, n-1, \ldots, 1$-grams) or Dirichlet smoothing [Hiemstra, 2009, Collins, 2013].

$$p(w_n|w_n-1) = \frac{1 + c(w_{n-1}, w_n)}{1 + c(w_{n-1})} \tag{3}$$

$$p(w_n|w_n-1) = \frac{k + c(w_{n-1}, w_n)}{k + c(w_{n-1})} \tag{4}$$

All in all, probability smoothing improves n-gram models, but also makes them more complex, because most smoothing functions have parameters whose optimal values must be determined [Hiemstra, 2009, Jurafsky and Martin, 2014].
A complementary approach to probabilty smoothing known as discounting (i.e. lowering actual high occurrences counts) consists in transfering a small fraction of probability mass from the most common n-grams to the non-observed ones [Jurafsky and Martin, 2014].

Backoff Smoothing techniques consist in replacing occurrences for unseen word sequences with non-zero values, because zero values are problematic for model generalization. As mentioned before, longer n-grams produce a more detailed model, but less general model. Backoff methods try to solve that problem with a different approach than smoothing : instead of replacing zeros with arbitrary values, the main idea of backoff is to choose the more detailed model (i.e. the model with greater n) that still is general enough for the word sequence (i.e. in which the probability of the sequence is not zero). For example, when trying to compute the probability of word sequence (w_1, w_2, w_3, w_4), the model initially uses 4-grams. If the result probability is 0 (if (w_1, w_2, w_3) is never observed in the corpus), the model will backoff to 3-grams, which might give a non-zero probability if (w_2, w_3) was actually observed. If not, the model can backoff again, and so on until a non-zero result is found.
The following example show the functioning of backoff n-grams. Suppose

you want to compute the probability of the sequence "where is this going" to appear in a corpus. At some point, you must compute the conditional probability :

$$p(\text{"}going\text{"}|\text{"}where\ is\ this\text{"}) = \frac{c(\text{"}where\ is\ this\ going\text{"})}{c(\text{"}where\ is\ this\text{"})} \tag{5}$$

which might be zero if "where is this going" never appears in the corpus. In this case, it would be necessary to backoff to 3-grams :

$$p(\text{"}going\text{"}|\text{"}where\,is\,this\text{"}) \approx \alpha * p(\text{"}going\text{"}|\text{"}is\,this\text{"}) = \alpha * \frac{c(\text{"}is\ this\ going\text{"})}{c(\text{"}is\ this\text{"})} \tag{6}$$

If $c(\text{"}is\ this\ going\text{"})$ still is zero, it is possible to backoff to 2-grams recursively, and so on.
When backing-off to $(n-1)$-gram model, the probability is multiplied by a coefficient (α in (6)) to ensure that the values sum to 1 (not done, for simplification of the example). This model, known as Katz backoff in reference to its author, adapts the context history length to the given word sequence and is commonly used in natural language processing [Katz, 1987, Seymore and Rosenfeld, 1996, Goodman, 2001].

Skipping As explained previously, backoff models try to overcome the issue of unseen n-grams by reducing the history length. Skipping methods rely on a different idea : even if the exact word context has never been observed, similar contexts might exist in the corpus. Common skipping models consider two context as similar if they contain the same words except one. This can be generalized to allow to skip more context words. Such skipping techniques are efficient at handling word sequences with proper nouns, or unseen words with an observed synonym - i.e. a semantically close word - in the corpus.
Here is an example from [Goodman, 2001] : even if the sequence "Show John a good time" was never observed in the corpus, it is possible that the - semantically close - sequence "Show Stan a good time" was. Using backoff models would reduce the history length and switch to a less detailed model, although a very similar sequence does exist in the corpus. Skipping models can prevent this shortcoming by simply ignoring the proper noun in the sequence, and therefore use the most precise n-gram model.

2.1.4 Using probabilistic language models for semantic extraction

As explained before, semantic extraction is not the main purpose of probabilistic language models. These models try to compute if word sequences are likely to appear in natural language. Nevertheless, natural language follows - more or less precise - syntactic and semantic rules, which thus have an influence on the generated models.

From class-based n-grams to semantic relations Class-based n-grams models - also known as clustering n-grams models - rely on a classification approach to take advantage of syntactic and semantic regularities in natural language. Their main principle is to replace words by their class (also named cluster or category) [Brown et al., 1992, Niesler and Woodland, 1996]. A word class is a set of similar words, which can be synonyms, semantically or syntactically close words, or more precise or more general associated terms. Given this objective, class-based n-gram models have classify corpus words into classes. Different methods can be used for that:
- In [Brown et al., 1992], the authors initialize the model with one class per word, and then merge successively classes until they obtain a fixed number of clusters. At each iteration, the two classes with the least mutual information are merged. This process determines a specific order for merging the classes, which can be represented by a dendogram.
- Alternatively, in [Niesler and Woodland, 1996], the authors build a language model tree. In this tree, each node corresponds to a word class, each depth level to a n-gram size, and possible paths in the tree represent observed n-grams from the corpus. The tree is built while discovering the different n-grams in the corpus, and a new class is created only if it matches determined quality criterion (in order to keep the model as compact as possible and group words in classes).
There are many ways to build class-based n-gram models, but we will not focus on that matter here. Such models are not conceived for semantic extraction, they are based on semantic (and syntactic) rules to improve n-grams performance.
Still, words classes can be interpreted as semantically close words groups, as we can see in these cluster examples from [Brown et al., 1992]:
- Friday Monday Thursday Wednesday Tuesday Saturday Sunday weekends Sundays Saturdays
- water gas coal liquid acid sand carbon steam shale iron
- man woman boy girl lawyer doctor guy farmer teacher citizen
Classes contain semantically and syntactically close words : the first cluster corresponds to "days of the week", the second to "raw material and energy sources", and the third one to "persons and jobs".
The methods used to classify words into classes in n-gram models can obviously be used for semantic extraction. After obtaining the preceding cluster

examples, the authors successfully use their results to collect sticky word pairs (words that appear often together) and build semantic clusters.

Drawbacks of probabilistic language models for semantic relations discovery As explained previously, semantic relations can actually be extracted using probabilistic language models, in particular class-based n-grams. However, such models are not optimal for semantic extraction, because they were not designed with this objective in mind.
In practice, computation complexity is not really a problem with n-grams, since it has been shown that high n-values give too specific models that fail to generalize. But the main problem is that some words (or words sequences) will inevitably be absent from the model vocabulary. Since classic n-grams cannot be used on unseen words (or sequences), we face a major problem. Solutions to this problem exist (like backoff) and have been successfully applied, but make the models more complicated. All in all, probabilistic language models appear to be a cornerstone of semantic extraction, but also not the best suited tool for this task.

2.2 Statistical count-based models for semantic similarity

Probabilistic language models aim to compute the likelihood of appearance of word sequences, thus allowing to check correctness of existing sentences and to generate new ones. As explained before, those models can also be used to build semantic clusters, although this is not their core feature.
Unlike probabilistic language models, which are typically used for speech recognition, lexical co-occurrence models were predominantly studied for information retrieval. Lexical co-occurrence models are statistical models, based on the co-occurrence count of words. Such models intend to represent words in a high-dimensional vector space, where distances can be semantically interpreted. It is important to notice that co-occurrence models are mainly mathematical - not language processing - models.
In this section, we will focus on three co-occurrence based models : HAL (Hyperspace Analogue to Language) [Lund and Burgess, 1996], LSA (Latent Semantic Analysis) [Deerwester et al., 1990] and COALS (Correlated Occurrence Analogue to Lexical Semantic) [Rohde et al., 2006]. First, we will examine the HAL and LSA models, given their great influence. Then, we will compare HAL and LSA to a third model named COALS (Correlated Occurrence Analogue to Lexical Semantic), inspired from the two previous ones. Lastly, we will discuss strengths and weaknesses of statistical co-occurrence

models in terms of semantic extraction.

2.2.1 Hyperspace Analogue to Language model (HAL)

Unsupervised semantic approach The HAL model [Lund and Burgess, 1996] is based on human semantic memory, and its main idea is to build a semantic hyperspace. In this representation, each vector represents a word, and the coordinates are correlated to the meaning of the word. We will later discuss how to interpret this mathematical representation. To build the semantic hyperspace, word co-occurrences are used : the more often two words appear together, the more semantically close they are.
HAL positions words in a semantic hyperspace. The main innovation in HAL is to build the semantic hyperspace automatically, in an unsupervised way. In previous methods, each axis is chosen manually to corresponds to a given property. Such a procedure is very restrictive and difficult to scale because each word needs n human judgements (where n is the dimension of this space) to be positioned manually along all axes in the hyperspace. Thus, the choice of axes is complex, but it is also difficult to choose the number of axes to use. Low-dimensional spaces require less effort to be built, but their ”degree of freedom” - and therefore their expressivity - is also low. On the contrary, high-dimensional spaces are much more reprensentative and expressive, but they are difficult to build and to interpret for humans. The authors of HAL propose to overcome this dilemma by building the semantic hyperspace automatically from co-occurrence counts. This leads to a rapidly buildable high-dimensional model.

Procedure for HAL models The detailed procedure used to compute the semantic hyperspace in HAL is the following. First of all, all words co-occurrence wihtin the corpus are counted. HAL uses word windows of fixed size for this step, so the co-occurrence score of two words is incremented each time these words appear in the same window. The co-occurrence score is not the exact co-occurrence count, but takes into account the relative distance of the two words in the window, i.e. the co-occurrence score is higher if two words are separated by only one word than if they were separated by two words. The default window size is 20 words (10 words before, 10 words after). A co-occurrence matrix is then built by sliding the window over the whole corpus. It is also interesting to note that HAL distinguishes between the left side (before) and the right side of the window, and consequently the co-occurrence matrix does not have to be symmetrical. The resulting matrix contains a vector for each unique word in the corpus, obtained by concatenating the ”left” and ”right” co-occurrence vectors of this particular word

with all others.
Classical distance metrics can then be applied to the co-occurrence vectors to qualify the similarity between words. Obviously, it is also possible to explore the nearest neighbors of a given word in this semantic hyperspace. In next section, we discuss results obtained by the authors of the HAL model.

Results and discussion As mentioned in [Lund and Burgess, 1996], HAL gives encouraging results in the domain of semantic relationships extraction. The similarity measures of the model seems to correlate well with human judgement, and can be used to cluster words into semantic categories.
Although it can discover semantic relationships, the major weakness of HAL is that it cannot characterize these relationships. More specifically, syntactic and semantic relations are not distinguishable. Furthermore, associative relations (i.e. coffee-cup) should be differentiated from pure semantic relations (i.e. table-bed), which is not possible here.
As a conclusion, the HAL model has been very influential through its new, unsupervised and therefore scalable approach to build semantic vector spaces, yet it still suffers from lack of precision for semantic extraction.

2.2.2 Latent Semantic Analysis (LSA)

Like HAL, LSA is based on word co-occurrences, but it is used for information retrieval, in particular matching documents with queries. The LSA method has been proposed in [Deerwester et al., 1990] to fill the semantic gap between documents (composed of words) and the conceptual topics they represent, by taking advantage of what its authors call the ”underlying latent semantic structure”.

LSA main concept : filling the semantic gap between query and document LSA's primary goal is to improve information retrieval in documents, where simple term matching is not sufficient. This insufficiency comes from two properties of natural language:
- synonymy : two words can refer to the same concept or entity
- polysemy : most words can have more than one meaning depending on their context
When matching documents to a query, synonymy influences recall[2] (exact term matching retrieves the documents with the exact terms, not the ones

[2]proportion of positive that are actually detected, corresponds to the statistical type-II error (false negatives)

using the terms' synonyms) while polysemy influences precision[3] (documents containing query terms used in a whole different sense can be retrieved). To put it more clearly, synonymy can yield false negatives (relevant documents are not retrieved), whereas polysemy can yield false positives (irrelevant documents are retrieved).
The LSA method described here consists in building a co-occurrence based semantic hyperspace - like in HAL - but this time both words and documents will be represented in this space. Document querying can then be done by retrieving close elements in this hyperspace.

Building the LSA model The LSA model is based on the word co-occurences within the documents. No word window is used here, contrary to the HAL method. The first step of LSA is to compute the terms / documents occurrences matrix. Then, SVD (Singular Value Decomposition, an equivalent of diagonalization for rectangular matrices) is applied, resulting in three matrices D (documents), S (singular vectors) and T (terms). This conversion allows a lossy compression of the co-occurrence matrix by leaving singular vectors out, but this is not a problem, since the original matrix is considered unreliable (because of synonymy and polysemy). Quite the opposite, it makes it possible to reduce the dimensionality of the model.
Once the semantic hyperspace is built, documents can be retrieved using queries. Given a query - which is nothing else than a set of terms - it is easy to find its neighbor documents according to a distance metrics. In practice, cosine similarity is used in LSA.

Results and discussion In [Deerwester et al., 1990], the authors affirm that LSA results are promising, giving a slightly better precision-recall tradeoff than previous models. LSA successfully uses indirect associations for information retrieval, making it more robust than simple term matching.
LSA handles the problem of synonymy well thanks to its similarity model, but the problem of polysemy remains at least partially, as one word is represented by only one vector. A subcategorization would be necessary to allow disambiguation. Lastly, LSA uses segmented text as input data, which is convenient for document retrieval, but not in the case of general semantic extraction from natural language.

[3]proportion of detections that are actually true positives, corresponds to the statistical type-I error (false positives)

2.2.3 Correlated Occurrence Analogue to Lexical Semantic model (COALS)

We present here the COALS model, an improved co-occurrence model based on HAL and influenced by LSA.

From HAL to COALS Like HAL, COALS is a co-occurence based model for human similarity judgement prediction, and uses an unsegmented text corpus to build a semantic vector model in an unsupervised way.
The particularity of COALS is to use SVD (like LSA) and other mathematical improvements to enhance the similarity computation. As mentioned in Rohde et al. [2006], the main weakness of HAL is that high frequency columns (i.e. most common words) have too great influence in the similarity measures. COALS handles this issue by normalizing the word vectors : the conditional occurrence of two words (Does w_1 appear more or less with w_2 than in general?) must be favored over the raw co-occurrence (How many times do w_1 and w_2 appear together?). This is why COALS uses a correlation measure for similarity. Additionnally, the most common words are ignored because they are not representative enough, and COALS can also output binary word vectors, which are better suited for neural networks than float vectors.

COALS : using correlation and SVD for semantic similarity The COALS semantic model is computed as following. First of all, the co-occurences are counted within 4-word windows, taking into account the relative distance between two words like in HAL. Then, the most common columns are discarded and the co-occurence counts are converted to correlations. Pearson's correlation is used for this step. The correlation of w_1 with w_2 is positive if w_1 tends to appear more often when w_2 is present, negative if less often. This correlation is finally used as a semantic similarity measure. Most word pairs are slightly negatively correlated (two randow words are likely to have no direct semantic link). Therefore, positive are more interesting because they are based on actual observations (negative correlations are based on non-observations and unreliable). In COALS, positive correlation values are low and replaced by their square root, negative correlations are replaced by 0 to reduce noise. This second strategy also has the advantage to make the correlation matrix sparse, which is a useful property for SVD (SVD can be used as an option if dimension reduction is needed).

Results and discussion According to [Rohde et al., 2006], COALS seems to perform better at semantic extraction than LSA, HAL and other co-occurrence based models. The use of undifferentiated text and unsupervised approach make COALS a scalable model. Nevertheless, like in other co-occurrence models, word-sense disambiguation still has to be improved.

2.2.4 Strengths and weaknesses of statistical co-occurrence based models for semantic extraction

We have described and analyzed three co-occurrence based models and their functioning. It appears that the greatest strength of co-occurrence models is to be unsupervised. Therefore, no human annotation is needed and these models can be applied on large and representative datasets. These models all use semantic hyperspaces to calculate word similarities. Once this space has been computed, similaritiy scores are obtained very easily. In some models, methods to introduce new words into the vocabulary without having to compute the hyperspace again have been proposed. All these advantages explain the popularity of statistical co-occurrence models for semantic extraction from natural language.
However, co-occurrence models also have their weaknesses. The main critic against co-occurrence models is that they are relative models : a word vector cannot be interpreted in an absolute way, but only in comparison to another word vector. Thus semantic relationships can be found, but it is impossible to determine the intrisic meaning of a word. Another problem of co-occurrence models is their inability to distinguish syntactic from semantic and semantic-associative relationships, due to their purely statistical nature. Moreover, polysemy remains a problem as co-occurrence map one word to one vector only, even if several meanings exist. For all these reasons, it appears that statistical co-occurrence models are useful and simple, but unsifficient to perform complete and precise semantic extraction.

2.3 Semantic role labeling (SRL)

As explained previously, statistical co-occurrence models can be useful for semantic extraction, but they suffer from several drawbacks (only relative semantic distance, no distinction between syntactic and semantic structure, no polysemy consideration). This is because co-occurrence models are plain mathematical models.
On the contrary, semantic role labeling (often abbreviated as SRL) is a subtask of natural language processing based on linguistic considerations, therefore allowing to extract syntactic and semantic information from text without the disadvantages cited above.
First, we quickly present the general principle of SRL and its goals. Then,

we show how SRL can be used for semantic extraction and take a survey of existing implementations. Finally, we analyze the pros and cons of SRL for semantic extraction.

2.3.1 An overview of semantic role labeling

Definition, general principle and goals of SRL Semantic role labeling, also called "shallow semantic parsing", is a subtask of NLP which consists in identifying the arguments of a predicate (i.e. the verb in a sentence) and their types (called roles). Roles are for example agent (does the action), patient (subjected to the action), temporal, location, ... A simple and popular definition of SRL is to find "who did what to whom (and perhaps when and where)" [Jurafsky and Martin, 2015].
As syntactic analysis is not sufficient for complete and accurate semantic understanding (mostly because of the polysemy problem), SRL aims to find a high-level representation for semantic information. Although it is mostly based on syntactic tree constructions, SRL assigns roles to words groups in their relation to the corresponding verb. By doing so, semantic role labeling can deal with the polysemy problem and can be syntax-independent. For example, the same sentence in active and passive voice should be represented in the same way with SRL, because its meaning stays the same even though the syntax changes.
For example, the sentences "John scored a goal." and "A goal was scored by John." are lexically and syntactically different, but semantically identical, so they should have the same SRL representation which should contain following elements : predicate = "score", agent = "John", patient = "a goal".

SRL for semantic extraction After understanding the principle of semantic role labeling, it becomes quite clear why this subtask of NLP is essential and how it can be applied to semantic extraction from text : SRL characterizes precisely the action of a verb and its arguments.
This semantic information is very different from the statistical distances that can be obtained with n-grams or co-occurrence models, and it is also more precise because mainly based on linguistics. This is clearly an advantage over mathematical language models for some tasks (such as disambiguation).
However, the main problem of this linguistic systems is that they need reference datasets, which must be crafted by humans. This is why research in the field of SRL made significant progress only after such datasets were made available. In the next section, we quickly describe the functioning of SRL systems, reference datasets (FrameNet [Baker et al., 1998] and PropBank [Palmer et al., 2005]) and implementation examples [Gildea and Jurafsky,

2002, Pradhan et al., 2004].

2.3.2 Implementing SRL systems

Basic tasks for SRL implementation The general steps to build an SRL system are the following : build a set of roles (which will be assigned to the action arguments), identify predicates and their arguments in natural text, and finally classify the arguments in regard to the semantic roles.
First, the set of roles is constructed manually and arbitrarily (each SRL system can have its specific role definitions). Then the predicates and arguments are identified by automatically parsing the text, most of the time by building syntactic trees. Finally, predicate arguments are classified into role categories, using so-called "features". Features are properties of the given word sequence to classify, for example the phrase type, the headword of the sequence, the path in the parse tree, the relative position to the verb. The choice of features is primordial for classification performance and has been widely studied [Gildea and Jurafsky, 2002, Xue and Palmer, 2004]. The features set is traditionnally chosen in the conception phase of the SRL system, but it has also been proposed to learn which features are the most relevant [Carroll et al., 2005].
Once the features are chosen, a supervised classification algorithm is applied to annotated datasets to evaluate the system's performance. The most common datasets used for SRL are FrameNet and PropBank, which are described next.

FrameNet FrameNet [Baker et al., 1998] is a dataset for NLP containing semantic annotations in a formal, computer-readable form.
Three elements constitute the FrameNet database:
- a lexicon, with classical dictionary entries and usage examples
- a frame database : "frames" (i.e. text extracts) with conceptual structure. For example, the frame "transportation" consists of the elements MOVER, MEANS and PATH, arranged in the conceptual structure "MOVER moves along PATH by MEANS".
- annotated sentences based on the two previous resources, which can be used as training data for SRL systems.

In FrameNet, frame elements correspond to role names defined locally to conceptual structures. Frames can then be combined to express more complex semantic structures. To continue the previous example, the frame "driving" inherits "transportation" with the elements DRIVER(=MOVER), VEHICLE(=MEANS), RIDER(=MOVER) and CARGO(=MOVER).
In this way, FrameNet represents a ground truth for SRL constituents iden-

tification and classification. At its publication, the FrameNet database contained 10,000 sentences and 20,000 frame elements (250,000 sentences were expected at the end of the research experiment), making it a very useful resource for SRL.

PropBank Like FrameNet, PropBank provides a dataset for SRL, based on the Penn TreeBank [Palmer et al., 2005].
The main difference with FrameNet is that the roles are generic : the predicate arguments are labeled arg0, arg1, and so on. The meaning of each argument is then defined relatively to the verb. PropBank characterizes each sense of each verb with a set of argument labels and a definition of the predicate. With this mechanism, PropBank can perform verb disambiguation based on the number of arguments and their roles. As an example, the predicate "aim(arg0 = aimer, arg1 = plan)" corresponds to "intend", whereas "aim(arg0 = aimer, arg1 = weapon, arg2 = target)" corresponds the sense "point (a weapon) at".
Although the role labels are generic, arg0 mostly represents the agent and arg1 the patient. Universal roles (i.e. common to all verbs) are also defined to model temporal, location, purpose, ...
With all its characteristics, PropBank is well suited for SRL classification training.

Examples of SRL implementation We have explained the general procedure of SRL and described two popular training datasets used in this community. In this section, we take a very brief survey of existing SRL implementations.
Most systems use pruning techniques to filter constituents that are not predicate arguments acccording to syntactic rules. The remaining constituents are then classified into role categories with statistical (probabilistic) algorithms on their properties ("features"), like in [Gildea and Jurafsky, 2002] and [Xue and Palmer, 2004]. As mentioned before, the choice of features has great influence on the system performance and has therefore been studied with attention.
But other improvement leads have been explored too, like in ASSERT (Automatic Statistical SEmantic Role Tagger) [Pradhan et al., 2004], where the authors use a support vector machine (SVM) as classifier, obtaining significantly better results. The ASSERT example is interesting not only for its promising results, but also because it showed that SVMs (and more complex classifiers inspired by SVMs, like neural networks) were very well adapted to NLP tasks.

2.3.3 Limits and perspectives of SRL

Now that we have exposed the functioning of SRL and explained how it can be used for semantic extraction, we will focus on limits and improvement possibilities of such approaches.

Consituent independence hypothesis The consituent independence hypothesis is one important weakness of SRL systems [Jurafsky and Martin, 2015].
In such systems, constituents are word sequences corresponding to a semantic role (which can be none). In the FrameNet dataset, two constituents cannot be overlapping and in PropBank, overlapping constituents are detected but ignored for simplification [Jurafsky and Martin, 2015]. As almost all SRL implementations use FrameNet and PropBank as training data, semantic constituents necessarily have to be independent from each other, and this does not always correspond to the reality. The ASSERT system [Pradhan et al., 2004] tries to overcome this problem and takes into account the most probable constituent when there are two overlapping, but it still limits the number of recognized elements to 1.
Moreover, PropBank admits only one constituent per type in a sentence, which can be seen as an additional limitation. This can lead to mislabeling if two constituent of same type are detected in the same sentence. A solution to that problem is to apply a first local classification (assigning the best role to each constituent) before applying a second global classification for best consensus [Jurafsky and Martin, 2015].

Improving SRL with selectional restrictions We have discussed the issue of constituent (in)dependence in SRL, because it is of great importance in the case of semantic extraction. In this section, we describe a technique called ”selectional restriction”, used to improve SRL performance.
In SRL, the most important word in a sentence is the verb (predicate), because all semantic arguments are defined relatively to it. However, the same verb can have different syntactic constructions. In PropBank, the number of arguments is used to disambiguate verbs, but this technique cannot be applied for all sentences. For example, ”He is going to eat an apple.” and ”He is going to eat alone” have the same number of predicate arguments and yet the verb is transitive in one case, intransitive in the other.
Selectional restriction consists in defining semantic categories. In the previous example, ”an apple” would be assigned to the ”edible” category, and the SRL system would only label ”edible” entities as ”object”-role of the transitive ”eat” predicate [Jurafsky and Martin, 2015]. Selectional restrictions can

be implemented using WordNet (see 5.2) and permit disambiguation through word categorization (this main idea can be compared to the principle of class-based n-grams, where a word is replaced by its class).

3 Neural network language models and deep learning techniques for NLP

In previous section 2, we have described classical models for semantic extraction from text, which have been known for several years and widely studied. Keeping semantic extraction in mind, in this section we will focus on more recent NLP models, based on deep learning (neural network language models in particular). Although neural networks have existed for a while, their large usage in the NLP field is quite recent.
First, we present the advantages of deep learning approaches in the case of NLP, in comparison to feature engineering (manually extracting properties for classification, see SRL in 2.3). Then, we quickly present the functioning of neural network language models (NNLM) for classical NLP tasks, and the data structures used by these models, including distributed word representations. Lastly, we will aim our attention at distributed word representations (also called word embeddings) and their usage for semantic relations extraction.

3.1 Motivations : machine learning instead of feature engineering

Deep learning is a subset of machine learning methods, that create high-level abstractions from low-level input data, using multiple non-linear transformations, for example with artificial neural networks. This capacity to build high-level representations automatically (i.e. without human intervention) is seen as major progress in artificial intelligence, and makes deep learning methods particularly well suited for applications where "human-like judgement" is required, such as natural language processing, speech and image recognition, ...
In this paper, we focus on a particular deep learning usage : neural networks for NLP tasks and especially semantic extraction.

Learning "hidden" computer-readable representations As mentioned previously, deep learning methods are representative of a "large scale machine

learning" approach, that is to be opposed to "large scale engineering" [Collobert and Weston, 2009].
In large scale engineering methods, features (properties of input data) are meticulously studied and chosen for their supposed relevance according to the task. This implies to have good knowledge of the task and its larger domain. For example, in SRL (see 2.3), the choice of the best features to use has been largely discussed [Gildea and Jurafsky, 2002]. One could qualify such large scale engineering methods as "using fast and simple algorithms on complex features", or as in [Collobert et al., 2011], "applying linear statistical models to ad-hoc features".
Large scale machine learning approaches claim to be the exact opposite : (very) simple features are fed to complex algorithms [Collobert and Weston, 2009]. Because the input features are simple, very few preprocessing and prealable knowledge are needed. Task-specific engineering can therefore be avoided and systems are conceived almost "from scratch", as in [Collobert et al., 2011]. Complex features are still used, by only implicitly and internally to the system (as non-linear combinations of the input features) and this allows to discover "hidden structure" in processed data. Because of this particular architecture, neural networks and other deep learning systems are efficient for artificial intelligence tasks (including NLP), but their intermediary results are difficult to interpret for humans.

Scalability Inability to scale to large datasets often is a problem with complex models, but it is not the case with deep learning approaches. On the contrary, usage of very large datasets is rather beneficial for neural network systems because the more examples they learn, the more precise they are. As learning is fully automatic and consists in simple algorithms, it can be applied to large data.
For instance, in [Collobert and Weston, 2007], the authors construct a semantic extraction system that gives results comparable to state-of-the-art 250 times faster (0.02s instead of 5s). Moreover, multitasking approaches are also possible to improve performance [Collobert and Weston, 2008]. This ability to process enormous amounts of input data represents a consequent advantage in NLP, where models need large datasets to generalize well.

Supervised learning, unsupervised and semi-supervised learning Like other machine learning systems, neural networks can be trained in different ways [Chapelle et al., 2006].
Supervised learning consists in finding a function that maps input to output, which is generally a manually labeled dataset. In pratice, supervised learning

is difficult to use in NLP because of the high labeling cost. However, it can be applied to large datasets when the expected output is available without annotation, as in context-predictive word embeddings (see 3.3.2).
Unsupervised learning corresponds to finding structures in unlabeled datasets, that is to estimate the function or distribution most likely to have generated the observations (maximum likelihood estimation). In this case, the input entries are often grouped by common class or category, so as to estimate a function per class.
Semi-supervised learning tries to combine both previous methods, using labeled and unlabeled data. Thus, only a fraction of the input can be labeled. Semi-supervised learning is therefore very well adapted to NLP tasks, for which the labeling cost is high but large unlabeled datasets are available and necessary to obtain a good model.

3.2 Neural network language models

We have exposed the motivations for using deep learning systems, and particularly neural networks, for NLP tasks. In this section, we focus on neural network language models (NNLM). At first, we present classic NLP tasks that can be performed with neural networks. Secondly, we explain the principle of feature learning applied to these tasks, and finally we take a closer look at data structures used in NNLMs : words vectors.

3.2.1 Classic NLP tasks

Divide and conquer approach Neural network language models and other deep learning models for NLP aim to build computer-readable representations of natural language. Complete natural language representation and understanding is still very far from current NLP state-of-the-art, because it implies various complex tasks. Moreover, there is no consensus about the form (the structure) such a representation should have [Collobert et al., 2011]. Until progress is made in this direction, most NLP systems use a "divide and conquer" approach by separating the problem into many subtasks [Collobert and Weston, 2008, 2009], each one being performed independently with a specific purpose.

NLP subtasks Classic NLP subtasks for syntactic and semantic extraction are listed below. Part-of-speech tagging (POS) consists in assigning syntactic tags to each word (i.e. noun, verb, adjective, singular or plural, ...). Chunking aims to delimit coherent syntactic elements which can be words or word sequences and giving them roles (i.e. noun, verb phrase, ...).

Named entity recognition (NER) is the task of recognizing proper names in text and to categorize them (i.e. persons, places, companies, ...). Another important subtask of NLP is semantic role labeling (SRL), described previously in 2.3 [Collobert and Weston, 2009]. These tasks are mostly basic, but their combination can lead to more powerful semantic extraction from text, as explained in next section.

3.2.2 Feature learning in NNLM

NLP systems use so-called "features" (properties of the words and sentences to analyze) as input for syntactic and semantic extraction tasks mentioned above : POS, NER, chunking, SRL, ... Implicitly, these features are used because they are likely to be relevant for these tasks. To give a simple example, the position of the word in the surrounding sentence may be a relevant feature for part-of-speech tagging. In engineering-based NLP systems, features are chosen according to domain knowledge, whereas machine learning approaches (such as neural network language models) are trained to learn relevant features automatically [Collobert and Weston, 2009].

Principle of neural networks To learn which features are relevant, neural network language models are fed with simple features as input. Neural networks consist of several computation layers, each layer taking as input the output of the underlying layer. In this way, neural nets can compute complex (non-linear) combinations of the inputs efficiently. The output is computed as a weighted combination of complex features given by the previous layers, and compared to an expected value (this corresponds to the case of supervised training). The error between real and expected output is then computed and the weights of the net are adapted accordingly (this procedure is called backpropagation). This operation, referred to as "training", is repeated until the output error becomes inferior to a given threshold.

Cascade and multitask feature learning In the case of NLP, the global "natural language representation and understanding" problem has been separated in many subtasks for simplification, and these tasks are often performed independently. However, that doesn't mean they are completely independent : results of some subtasks are actually very useful to other higher-level tasks (i.e. part-of-speech tagging, which corresponds to single-word syntactic analysis, can be useful for chunking, which can be seen as word sequence syntactic analysis).
In the case of NNLM, this is why features and neural nets outputs are interdependent. Some implementations, like in [Collobert and Weston, 2007], use

cascade feature learning : outputs for low-level tasks (such as POS, chunking) are used as input features for higher-level tasks (like NER, SRL). The main problem with this approach is error propagation : incorrect output from low-level system can be taken as input by higher-level systems. To reduce this cascade error propagation, joint training (where all systems are trained simultaneously) can be implemented, as in [Collobert and Weston, 2008]. In this case, all systems are trained at the same time, so as to reduce errors globally.

3.2.3 Structure for NNLM : from words to neural network vectors

Now that we have explained the principle of functioning of neural networks, described the features they need as input and which NLP tasks they are capable to perform, we focus here on pratical details of NNLM implementation. In particular, we describe how to transform raw text into NN-understandable input data : word vectors.

Word feature vectors Starting from raw natural text, the first step is preprocessing, to remove special characters (like symbols, numbers). The nature of the characters to remove or transform depends on the final goal of the system. The second step, the most important, is to transform the words of the corpus into feature vectors. In [Collobert et al., 2011], all unique words are first identified by an indice. A ”lookup table” is then built to associate a feature vector to each indice. A feature vector is simply a vector of real values, each one corresponding to one of the chosen features set. For example, if the first feature is the capitalization (1 for capitalized, 0 for uncapitalized) and the second the length of each word, then the word ”city” would be represented by the vector (0,4). This is of course a very simple model to show how to build a word vector. In practice, NNLM vectors can contain hundreds of features. The important idea here is that neural networks need a mathematical, vectorial representation of words, so as to automatically learn which features are relevant for the current task.

From word vectors to word embeddings A new category of word representations has emerged with deep learning techniques for NLP : distributed word representations, also known as word embeddings. These models are doubly linked to NNLMs, because they are computed using neural networks, and represent words as vectors, so that they can also be used as input for other NLP with deep learning systems. In next section, we describe word embeddings, focusing on their promising possibilities for semantic extraction.

3.3 Predictive models and distributed word representations

As already mentioned, automatic semantic understanding of natural language is a complex task, that has been divided into many subtasks. The emergence of deep learning techniques such as neural networks and their application to NLP have allowed significant progress in this domain recently. More specifically, a new family of distributed word representations - called predictive models - outperforms previous models for semantic extraction. In this section, we first present predictive models and compare them to classic count models (see 2.2), before focusing on different types of distributed word representations, and finally showing their utility in terms of semantic extraction.

3.3.1 Distributional semantic models : predictive models against count models

Distributional semantic models According to [Baroni et al., 2014], distributional semantic models (DSM) are based on the main idea that similar words appear within similar contexts, and aim to represent words as semantic vectors. Intuitively, one could say that two words are close if it is possible to replace one by the other (synonyms). Distributional semantic models can be count-based (like HAL, COALS and LSA, see 2.2) or predictive models.

Count-based models Count-based models rely on co-occurrence counts to characterize word contexts. Unfortunately, raw co-occurrence counts are not performant to find similarities between words, and this why complex transformations have to be applied [Lund and Burgess, 1996, Rohde et al., 2006, Deerwester et al., 1990].

Predictive models Unlike count-based models, predictive models for distributed word representations don't use co-occurrence counts directly, but try to predict the context given a word, or vice versa. This is done by representing each word with a vector and "the weights in a word vector are set to maximize the probability of the contexts in which the word is observed in the corpus" [Baroni et al., 2014]. We will not explain the mathematical functioning of predictive models in details here, but computing such a model corresponds to an optimization that can be performed with neural networks [Rong, 2014]. Predictive models have become very popular in the last months because they seem to largely outperform count-based models [Baroni et al., 2014].

3.3.2 Distributed word representations (word embeddings)

Distributed representations use word context prediction and rapidly became popular on account of their good performance for semantic extraction [Mikolov et al., 2013a, Rong, 2014, Baroni et al., 2014]. Here, we present three different methods to compute word embeddings : continuous bag of words (CBOW), skip-grams and ParagraphVector.

Continuous bag of words and skip-gram CBOW and skip-grams are two very close approaches for distributed word representations. The skip-gram model is inspired from n-grams (see 2.1.2) and consists in predicting the context (as output) given a word (as input). Inversely, the CBOW (continuous bag of words, inspired from bag of words models for information retrieval) model takes the context as input to predict the target word as output. CBOW and skip-grams are very similar models, and one of the few differences between them is that skip-grams seem to perform better on small datasets, whereas CBOW is best suited for large datasets [Mikolov et al., 2013a,b, Rong, 2014].

ParagraphVector : extending vector representations to larger pieces of text The ParagraphVector model is an extension of CBOW and skip-gram to larger contexts : sentences, paragraphs, documents (instead of words) are represented by vectors. Because the ordering of words within a paragraph (or sentence, or document) and its semantic meaning are preserved, it improves results for semantic analysis, as explained in [Le and Mikolov, 2014].

GlobalVectors (GloVe) Unlike CBOW, skip-gram and ParagraphVector, GloVe (for GlobalVectors) is not a context-prediction model, but rather a count-based system (see 2.2). However, we describe it here because it aims to outperform CBOW and skip-gram models (see 3.3.2) for analogies extraction (see 3.3.3).
As explained in [Pennington et al., 2014], GloVe uses ratios of co-occurrence counts to filter out non-discriminative (unrelated) words in analogies, because such values can be only be interpreted relatively. For example, by computing the co-occurrence probability ratio $P(word_C|word_A)/P(word_C|word_B)$ for different words $wordC$ from the corpus, values much greater than 1 indicate a relatedness to $wordA$, values much less than 1 a relatedness to $wordB$, and values around 1 that $wordC$ is neither related to $wordA$ nor to $wordB$.
Based on this observation, GloVe learns word vectors so that the dot product of two words equals the logarithm of the words' probability of co-occurrence.

3.3.3 Using distributed word representations for semantic extraction

Distributed word representations consist in converting words into vectors assuming that similar words appear in similar contexts (and vice versa). With neural networks, these models can be trained on large datasets, making them very performant. In particular, word embeddings can be used in several ways to extract semantic relations between words.

Automatic translation As they are based on word context prediction, one very interesting property of word embeddings is that the vector space they generate is robust from one language to another. That is, words corresponding to the same concept in two different languages (i.e. mutual translations) will be assigned similar vectors if the same context predictive models is applied, because they represent the same reality, as explained in [Mikolov et al., 2013a]. In the cited paper, the authors show that the words representing the five first numbers in English (”one”, ”two”, ”three”, ”four”, ”five”) and Spanish (”uno”, ”dos”, ”tres”, ”cuatro”, ”cinco”) have similar positions in their respective vector spaces. This demonstrates that distributed word representations could be used to mathematically compute language translations with simple vector operations.

Associations and analogies computation It has been shown that distributed word representations can be used to translate words from one language to another. Here we explain how word embeddings can also help to find word associations and analogies.
Word embedding models build a vectorial word space, in which words are positioned closely to each other if they are similar (i.e. if they tend to appear in the same contexts in the corpus). Indeed, in [Mikolov et al., 2013b], it is clear that neighbors of a given word in the generated vector space are often its synonyms or semantically associated terms (for example the nearest neighbors of ”Italy” among countries are ”France”, ”Greece” and ”Spain”). It also appears that analogies can be retrieved from this vector representation with additions and substractions. For instance, ”Paris:France::Berlin:Germany” (Paris is to France what Berlin is to Germany) corresponds to vec(”France”)-vec(”Paris”) = vec(”Germany”)-vec(”Berlin”), so the answer to the question ”Paris:France::Berlin::?” could be computed by vec(”France”)-vec(”Paris”) +vec(”Berlin”). In this example, the vector vec(”France”)-vec(”Paris”) = vec(”Germany”)-vec(”Berlin”) implicitly represents the semantic relation ”is the capital of”.

Further open questions Distributed word representations are powerful models for semantic extraction, as shown in the previous examples. However, like other models issued from neural networks, they remain difficult to interpret. Word embeddings seem to encapsulate semantic knowledge, but only implicitly : dimensions of the vector space and word vectors cannot be interpreted directly, but only understood in a relative way.

4 Existing software for distributed word representations

We have previously described the functioning of predictive models for distributed word representations, and explained how these models outperform count-based models. In this section, we take a survey of existing implementations of predictive word embeddings.

4.1 Word2vec

Word2vec is the implementation of the predictive model described in [Mikolov et al., 2013a] (see 3.3.2). This NLP tool rapidly became popular and has been developed in many programming languages. We describe here the original C version, the Java version of the deeplearning4j package and the Python version of the gensim package.

4.1.1 Word2vec toolkit

The original Word2Vec implementation[4] was written in C by Mikolov and his team at Google.
It supports both continuous bag of words (CBOW) and skip-gram models. As training algorithms, hierarchical softmax[5] and negative sampling[6] can be used.

This word2vec implementation must be provided with the following main parameters : sub-sampling rate of frequent words, vector dimensionality and context window size.

[4] https://code.google.com/p/word2vec/

[5] Hierarchical softmax computes softmax efficiently using a tree representation, where the probability of a word is given by the path to its representing node [Rong, 2014].

[6] Negative sampling deals with the problem of having too many words for hierarchical softmax. In this case, only the ground truth is kept as positive sample, and negative samples are selected using an arbitrary distribution function[Rong, 2014].

The website provides well-documented use case examples (computing the nearest neighbors of a word, finding analogies, classify words into clusters), training data and pre-trained models.

4.1.2 Word2vec in Java (deeplearning4j package)

Another popular word2vec implementation in Java can be found in the deeplearning4j package[7].
This version of word2vec only includes the skip-gram model (better for large datasets) and uses negative sampling as training algorithm. The following parameters must be tuned : minimum word occurrence (words occurring less than this value are discarded), context window size and vector dimensionality. AdaGrad (adaptative gradient, which corresponds to automatic learning rate adaptation) can also be used for the training. It is also possible to import word models from this implementation (deeplearning4j), from the word2vec C version and from GloVe models (see 3.3.2).
Lastly, many examples, use cases and interpretations are provided on the website.

4.1.3 Word2vec in Python (gensim package)

A Python implementation of word2vec is also available as part of the gensim package [8].
This implementation supports both skip-gram and CBOW models (see 3.3.2), and hierarchical softmax and negative sampling for training. It must be given following parameters : vector dimensionality, context window size and minimum word occurrence count. It is possible to import models from the word2vec C version.
Few usage examples are given on the website, but available functions are very well documented.

4.2 GlobalVectors (GloVe)

Several implementations of the GloVe model (see 3.3.2) are described here : the original implementation from Stanford NLP, and other Java and Python adaptations.

[7]`http://deeplearning4j.org/word2vec.html`
[8]`https://radimrehurek.com/gensim/models/word2vec.html`

4.2.1 Stanford NLP implementation

The original GloVe implementation by the Stanford NLP Group is written in C and available to download on the project's website[9]. Pre-trained word vectors, as well as a preprocessing script are provided. Several use case examples (semantic nearest neighbors and analogies) are explained to illustrate the functioning of the system.

4.2.2 GloVe in Java (deeplearning4j package)

A Java implementation of GloVe is also available as part of the deeplearning4j package[10]. Unfortunately, this GloVe version seems to be still in development and no documentation is available as of now.

4.2.3 GloVe in Python

There are at least two versions of GloVe for Python. The first one was implemented by a student of the Stanford NLP group. The website of the project[11] doesn't provide much documentation or examples, but the functioning of the model is explained. The second version is a toy implementation available on GitHub[12], for which only few examples of semantic nearest neighbors are provided.

4.3 Doc2vec

In this section, two existing implementations of ParagraphVectors (also called doc2vec, see 3.3.2) are presented : the first one is a Java version, the second one a Python version.

4.3.1 Doc2vec in Java (deeplearning4j package)

The doc2vec model is implemented in Java as part of the deeplearning4j package. On the project's official page[13], few documentation is available, although a working code example is provided.

[9] http://nlp.stanford.edu/projects/glove/

[10] https://github.com/deeplearning4j/deeplearning4j/tree/master/deeplearning4j-scaleout/deeplearning4j-nlp/src/main/java/org/deeplearning4j/models/glove

[11] http://www.foldl.me/2014/glove-python/

[12] https://github.com/maciejkula/glove-python

[13] http://deeplearning4j.org/doc2vec.html

4.3.2 Doc2vec in Python (gensim package)

The Python version of doc2vec from the gensim package[14] is much more documented than the Java version. Some code examples and more importantly and a complete documentation of available functions are accessible.

5 Linked Open Data Cloud (LODC) for semantic relations discovery

We have described models for semantic extraction from text (see 2 and 3) and their existing implementations (see 4). In this section, we take a survey of Linked Open Data (LOD) systems, and show how they can be used for semantic extraction from natural text.
The Linked Open Data project is part of the Semantic Web, and consists in creating large knowledge datasets containing facts. These datasets are freely accessible with standard protocols and linked all together. Two important standards for Linked Open Data are RDF (Resource Description Framework) and SPARQL. RDF defines the format in which LOD (facts) are stored : (subject, predicate, object), called "RDF triple", while SPARQL is a querying protocol and language for RDF databases (i.e. equivalent of SQL for the Semantic Web). In LOD, resources (subjects, predicates and objects) are uniquely identified by URIs (Uniform Resource Identifier), which guarantee permanent access.
We here present several LOD datasets : DBpedia, WordNet, FreeBase and YAGO, and show their utility for semantic extraction.

5.1 DBpedia : from Wikipedia to the LODC

5.1.1 DBpedia, a structured Wikipedia

DBpedia[15] is one of the most important knowledge bases of the Semantic Web. Its goal is to extract and store facts contained in Wikipedia in a structured way, and make it freely available on the Web. Therefore, DBpedia is a general-purpose knowledge base and follows the evolution of Wikipedia. Additionnally, DBpedia data can be linked to other semantic datasets.
The English version of DBpedia contains 4.58 millions concepts, from which 4.22 millions are classified into the DBpedia ontology (see 5.1.3). These concepts are for example persons, places, organizations, ... DBpedia is available

[14]https://radimrehurek.com/gensim/models/doc2vec.html
[15]http://wiki.dbpedia.org/about

in 125 languages for a total of 38.3 millions concepts.
Connected to other linked datasets by RDF triples, DBpedia claims to be "one of the central interlinking-hubs of the emerging Web of Data".

5.1.2 Technical architecture

Technically, DBpedia is a large RDF dataset stored on a Virtuoso server (a universal server for semantic web[16]) that offers HTML, RDF and SPARQL endpoints. However, a migration from Virtuoso to Pubby (a "Linked Data Frontend for SPARQL Endpoints"[17]) is already scheduled.

5.1.3 DBpedia ontology

The DBpedia ontology (that defines the relations among the semantic dataset) is shallow and cross-domain[18]. It was created manually from Wikipedia infoboxes and holds 685 classes and 2795 properties. These classes are populated with instances (735,000 places, 1,450,000 persons, 241,000 organizations, etc.).

5.2 WordNet : lexical linked dataset

5.2.1 WordNet, a lexical database

Unlike DBpedia, WordNet[19] is not a semantic but a lexical database. In WordNet, words are grouped in synsets (groups of synonyms) linked by lexcial and conceptual relations. WordNet can be described as a thesaurus in which word senses are differentiated (semantical disambiguation) and semantic relations labeled.

5.2.2 Structure and relations of WordNet

The main relation in WordNet is synonymy (117,000 synsets exist and are linked to other with conceptual relations). Other relations are hypernomy (more general), hyponomy (more specific) and meronymy (part-whole relation). Additionally, nouns are divided in common nouns and instances (named entities), and verbs are organized in a hierarchical structure depending on the more or less specific action they describe. Adjectives are mostly

[16]http://semanticweb.org/wiki/Virtuoso.html
[17]http://wifo5-03.informatik.uni-mannheim.de/pubby/
[18]http://wiki.dbpedia.org/services-resources/ontology
[19]Princeton University "About WordNet." WordNet. Princeton University. 2010. https://wordnet.princeton.edu/wordnet/

organized with antonymy relations. Although it has not been created as part of the Semantic Web, WordNet is now integrated in the Linked Open Data Cloud.

5.3 FreeBase and Google Knowledge Graph

FreeBase[20] is a knowledge base organized as a graph, like DBpedia. Nodes of this graph represent objects, and edges links. FreeBase contains 39 millions topics of various multiple types. The particularity of FreeBase was its collaborative functioning : users could add new entities and relations. Now, FreeBase is read-only until it will be replaced by Google's Knowledge Graph API[21].

5.4 YAGO : "high-quality" knowledge base

YAGO[22] is a knowledge base built on Wikipedia, WordNet and GeoNames[23] (a geographical database), containing 10 millions entities and 120 millions relations.
YAGO defines itself as a "high-quality" database, because its accuracy was manually evaluated to 95%. YAGO combines WordNet taxonomy and Wikipedia categories to improve the precision of its relations, which also have spatial and temporal dimensions. YAGO data is extracted from Wikipedia in ten different languages.
As part of the Linked Open Data Cloud, YAGO has links to DBpedia and FreeBase, and provides a SPARQL endpoint for querying its contents.

5.5 Using knowledge databases for semantic extraction

This quick survey of LODC systems has shown their functionalities. Such databases contain huge amounts of semantic relations between entities (DBpedia, FreeBase, YAGO), or lexical information (WordNet), that can be very useful for semantic extraction applications. DBpedia for example provides datasets for NLP[24] that were used for the development of DBpedia Spotlight[25], a Named Entity Recognition (NER) tool to retrieve DBpedia entities

[20]https://www.freebase.com/
[21]https://developers.google.com/knowledge-graph/
[22]http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago//
[23]http://www.geonames.org/
[24]http://dbpedia.org/services-resources/datasets/nlp
[25]https://www.w3.org/2001/sw/wiki/DBpedia_Spotlight

mentions from natural text. NLP datasets from DBpedia include lexicalization data (DBpedia resource / surface form mappings) and topic signatures (doc2vec applied on Wikipedia paragraphs).
Although semantic relations from the Linked Open Data Cloud are possibly not reliable enough to be considered as ground truth for NLP applications, they can be compared with each other. This is the goal of the SemanticExtractor project described in next section.

6 Application : SemanticExtractor

6.1 Introducing SemanticExtractor, a tool to characterize word2vec analogies

As explained previously (see 3.3.3), word2vec is an efficient tool to compute word similarities and analogies from natural text corpora. However, word2vec is a pure mathematical model, and the vectors it produces can only be interpreted in a relative, limited way.
For example, if word2vec finds the analogy A:B::C:D (A is to B what C is to D), we understand that A and B on one side, and C and D on the other side are linked by the same relation. This is the very definition of the analogy. But what is the nature of this semantic relation? Moreover, we intuitively expect A and C (just like B and D) to be entities of same type : how can we model this in word2vec? And last but not least, does the analogy even make sense?
To answer these questions, we present SemanticExtractor, our tool to find semantic analogies with word2vec, and characterize them using Linked Open Data resources from DBpedia. SemanticExtractor generates annotated analogies between DBpedia entities or words.
First, we present SemanticExtractor's principle of functioning, then we propose a manual evaluation method for generated analogies, which we use to determine the optimal parameters of our system, and finally we discuss several improvements that could be implemented to obtain better results.

6.2 Principle of functioning

In this section, we explain the functioning of SemanticExtractor in three steps. First, we present the corpus we have chosen and how it was preprocessed, then we explain how the corpus is annotated and the word2vec model built, and in the end we detail how analogies are extracted.

6.2.1 Corpus choice and cleaning

Wikipedia dump as training corpus For SemanticExtractor, we used the corpus "enwik8"[26], a dump of the first 10^8 bytes of English Wikipedia. We chose this corpus because it is general, and its size appeared to be a good compromise between training time (long if the corpus is too large) and model precision (poor if the corpus is too small) in word2vec. Additionally, using a Wikipedia dump as training corpus was suggested on word2vec's original implementation page.

Formatting the training corpus The enwik8 corpus is a raw HTML dump from Wikipedia, and must be pre-processed to be usable with word2vec. A Perl script to clean this corpus is available on Matt Mahoney's website[27]. When applied to the dump, this script keeps only the words separated by spaces. All other characters are deleted. As word2vec expects sentences separated by linebreaks as input, the script had to be adapted to obtain such a format.

6.2.2 Annotation and extraction of word embeddings

Once we have a clean corpus, we can build a word embedding model on it. First, we annotate the corpus to identify DBpedia resources, then we apply word2vec on it.

Annotation with DBpedia Spotlight Before applying word2vec on the corpus, we annotate it with DBpedia Spotlight. Spotlight is a named-entity recognition (NER) tool to identify DBpedia entities in natural text. We use it to retrieve entities mentions and replace them with their DBpedia URI in the corpus.
Replacing entities mentions by their URI has two purposes. First, entities have to be identified by their URI in order to use them in Sparql queries on DBpedia (for example to retrieve their relations, see 6.2.3). Secondly, some entities correspond to word sequences (e.g. "Barack Obama" corresponds to `http://dbpedia.org/resource/Barack_Obama`) and their surface form (the recognized text, i.e. "Barack Obama") has to be replaced by a single word (i.e. "http://dbpedia.org/resource/Barack_Obama") to be interpreted as a single element by word2vec, which considers all words independently.
For the annotation task, we used the DBpedia Spotlight service at `http://spotlight.sztaki.hu:2222/` because it was much faster than the official

[26]`http://mattmahoney.net/dc/enwik8.zip`
[27]`http://mattmahoney.net/dc/textdata.html`

mirror. We used a confidence value of 0.3 with a support of 10, as it seemed to be a good compromise between recall and precision for NER on our corpus. Unfortunately, we couldn't try many different parameter values for confidence and support, because it took about 20 to 25 hours to annotate the entire corpus.

Word2vec model computation We then computed a vector representation for the words in the annotated corpus using word2vec. We kept the same parameter values as in the code example on the deeplearning4j website[28], except for the context window size and the size of the vectors, for which we compared different values (see 6.3.2).

Models persistence We saved our annotated corpus and our word2vec models to files, in order to be able to load them quickly later, instead of computing them again. We also computed an index of all entity types in the corpus, so as to efficiently find all entities of a given type (see 6.2.3).

6.2.3 Analogies discovery

In this section, we describe the procedure used by SemanticExtractor to obtain labeled analogies from our DBpedia-annotated word2vec model. This procedure is also represented as diagram in Figure 1.

Base entity and its relations As SemanticExtractor runs on large corpora, it is not possible to find all possible analogies in reasonable time. To tackle this problem, the user has to choose a base entity as starting point of the analogies. In other words, the user chooses manually an entity A from the corpus, and SemanticExtractor will generate all analogies A:B::C:D (A is to B what C is to D) contained in the word2vec model (i.e. analogies for which B, C and D are actually in the model).
To do so, we query DBpedia using Sparql, in order to retrieve all entities related to the base entity A (and the corresponding relations). This set of entities (and relations) is then filtered to only keep the entities appearing in the word2vec model of the corpus (other entities are useless as they have no vector representation). In this way, we obtain a set of entities related to A by known relations (these entities correspond to B in A:B::C:D) and contained in the word2vec model.

[28] `deeplearning4j.org/word2vec.html`

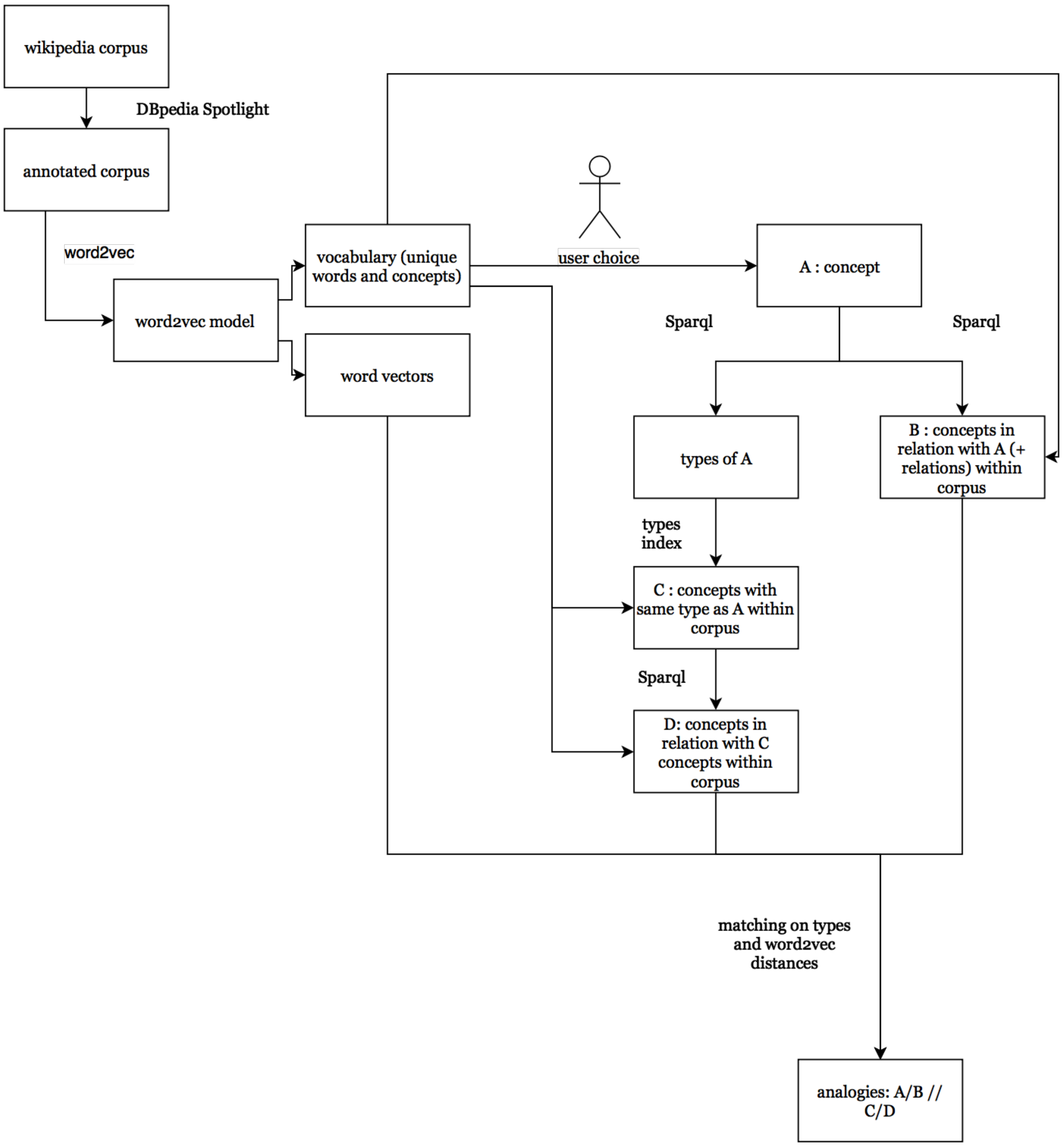
wikipedia corpus
DBpedia Spotlight
annotated corpus
word2vec
word2vec model
vocabulary (unique words and concepts)
word vectors
user choice
A : concept
Sparql
Sparql
types of A
B : concepts in relation with A (+ relations) within corpus
types index
C : concepts with same type as A within corpus
Sparql
D: concepts in relation with C concepts within corpus
matching on types and word2vec distances
analogies: A/B // C/D

Figure 1: Global functioning of SemanticExtractor

Base entity similarities We then search for entities corresponding to C in A:B::C:D. The number of words in the corpus is too large to consider all possibilities, so we focus on entities similar to the base entity A. Intuitively, this seems to be an acceptable hypothesis because if the analogy makes sense, A and C have to be of same type (or at least of compatible types). To find entities similar to A, we use its DBpedia types. A DBpedia entity can have several types, so we consider that the entities most similar to A are the ones who share the most common types with A.
For all DBpedia types of base entity A, we retrieve all entities of this given type (using a type index previously built to improve performance). By doing so, we get a list of entities sharing a type with A. This list can contain duplicated elements, corresponding to entities which share more than one type with A. Is it then easy to count all duplicates in the list to find the number of types shared with A. Lastly, we sort these entities by descending order of shared types with A. The first entities are those which share the most types with, therefore the most similar to A. Finally, we select a fixed number of these similar entities to obtain C in A:B::C:D (the optimal number of entities to select is evaluated in 6.3.3).

Analogies generation We have explained how to get B and C entities for analogy A:B::C:D, starting from given entity A. All A, B and C entities have a corresponding vector in the word2vec model of the corpus. We now want to find D entities for which A:B::C:D makes sense (i.e. for which AB and CD vectors are similar).
First, we consider all possible D entities, by applying all different AB vectors to all C entities using simple vector operations (additions and substractions) in word2vec. We obtain a set of D entities. We then filter this set to only keep entities for which the cosine similarity of AB and CD is superior to a fixed threshold, so as to guarantee reliable analogies. The optimal value of this threshold is discussed in 6.3.3.
Finally, we have obtained A:B::C:D analogies, where AB and CD relations should be equal and are characterized, because they correspond to a known DBpedia relation.

6.3 Evaluation of our results

In this section, we evaluate SemanticExtractor, to determine its optimal parameter values and estimate its performance for analogy computation. First, we explain our manual evaluation method, and then we determine optimal parameter values for word2vec (context window size and vector length) and analogy extraction (number of similar entities and minimum cosine similarity).

6.3.1 Evaluation method

Manual evaluation We chose to evaluate SemanticExtractor manually because it was (almost) impossible to do it automatically. SemanticExtractor generates new analogies, which could be theoretically be checked with DBpedia. But this supposes that the generated analogies must be exactly the same relations as in DBpedia, and this is not always the case. We prefer to evaluate the relations manually to quantify "how correct" they are, instead of considering that they can only be right or wrong. Moreover, some relations could actually be perfectly true and missing in DBpedia, which is why a manual evaluation is more reliable, although far less scalable.
We define a scale from 0 (worst) to 3 (best) for this manual evaluation. 0 will be assigned to analogies that don't correspond to a valid semantic relation at all. 1 will be assigned if the analogy is false, but still reflects some semantic meaning. 2 will be assigned to analogies that make sense in a certain way, but could be more precise. 3 corresponds to perfectly valid and correct analogies.

Base entities for evaluation To perform the evaluation, we need base entities, as SemanticExtractor cannot explore all entities. We chose to use following entities of various types : persons ("Barack Obama"), places ("Switzerland"), organizations ("IBM") and others ("Nobel Prize", "Rhododendron"). Clearly, this choice is arbitrary, and therefore disputable.
For all models, we computed all analogies of these five base entities and evaluated them manually according to the scale defined in previous section. We then computed a score as the average score for all analogies.

Default parameter values Unless mentioned explicitly, we used following parameter values:
- context window size = 10
- vectors dimensionality = 100
- minimum cosine similarity = 0.6
- number of similar entities (to base entity) = 15.
In the evaluation process, we analyzed the effect of changing each one of these parameters to understand their influence.

6.3.2 Word2vec parameters

In this section, we study the influence of two word2vec parameters on SemanticExtractor : the context window size, and the dimensionality of the word vectors.

Context window size results for three context sizes (5, 10 and 15 words) are given below. The average score is lower for a context size of 10 than for 5 or 15. The number of retrieved analogies decreases when the context window size increases, so the best choice appears to be a context size of 5, because it gives more analogies than 15, and they are more accurate than with 10. Also, the scores become more regularly distributed as the context window size increases (the standard deviation decreases).

Context window size	5	10	15
Analogies count	65	48	30
Average score (/3)	0.69	0.56	0.70
Std deviation	1.12	1.03	0.92

Vector dimensionality Evaluation results for three vector sizes (50, 100 and 150 dimensions) are given above. Increasing the vector dimensionality leads to less analogies retrieved. This could be because the vector space becomes more sparse as the number of dimensions raises. However, analogies are more accurante in high dimension (scores are much higher in the 150-model). A balance has to be found between the number of analogies discovered on one side, and their accuracy on the other side.

Vectors dimensionality	50	100	150
Analogies count	127	48	11
Average score (/3)	0.77	0.56	1.36
Std deviation	1.12	1.03	1.57

6.3.3 Analogy parameters

We also study the influence of two parameters specific to SemanticExtractor : the minimum cosine similarity (which can be seen as a minimum quality factor for the semantic relations) and the number of entities similar to the base entity to explore (C in A:B::C:D).

Minimum cosine similarity We computed the score of the same model with a minimum cosine similarity threshold of 0.5, 0.6 and 0.7. The higher the threshold, the less the count of retrieved analogies. This was expected because a higher similarity threshold selects less, but more specific relations.

But surprisingly, increasing the minimum threshold did not give higher, scores as expected. This seems to be because most analogies have a similarity score under 0.7.

Min. cosine similarity	0.5	0.6	0.7
Analogies count	121	48	11
Average score (/3)	0.60	0.56	0.45
Std deviation	1.03	1.03	0.52

Number of similar entities The number of entities similar to base entity to consider mainly has an effect on the number of retrieved analogies, as expected. Indeed, more similar entities explored means more possible relation combinations, therefore more analogies. The average score seems stable for 10, 15 and 20 similar entities, so it seems to be possible to consider 20 base-entity-similar entities without losing in precision (similar entities are ranked in descending similarity order). Higher values would need to be tested in order to determine the optimal count (to retrieve many accurate analogies).

Number of similar entities	10	15	20
Analogies count	33	48	59
Average score (/3)	0.50	0.56	0.64
Std deviation	0.92	1.03	1.06

General tendencies All in all, average scores are relatively low (the highest obtained score is 1.36/3) and the standard deviation is high. This indicates a few high-quality analogies (scores of 2 or 3), and many low-quality analogies. Indeed, SemanticExtractor often generates many A:B::C:D analogies with same entities for A, B and D, and among these analogies, only few have a C entity that makes sense. In other words, when applying the AB relation to all C entities, SemanticExtractor often retrieves the same D entity. It is difficult to say where this problem comes from, but this is why all average scores all relatively low.

6.4 Possible improvements

We have discussed results obtained with SemanticExtractor, which are globally not very accurate. In this section, we propose different possible improvements of our system.

Corpus choice To confirm our results, it would be necessary to use other corpora. If possible, it would be interesting to use a larger corpus (not feasible here because the annotation takes too long). As SemanticExtractor seems to give good results for domain-specific analogies (for example, we obtained many correct botanic analogies using the base entity "Rhododendron"), using a domain-specific corpus also could significantly improve our results.

Automatic validation One current great weakness of SemanticExtractor is that the evaluation has to be done manually. A possible improvement would be to use DBpedia to check if the analogies are valid. However we must be careful, as relations can be missing in DBpedia, and this works only for annotated entities (the analogy can be valid even if its entities are simple words). In other words, using DBpedia could confirm that a relation is valid, but not that it is incorrect.
Another problem of the manual evaluation process is the arbitrary choice of base entities (see 6.3.1). This difficulty could be overcome by selecting base entities automatically from the corpus using a representative sampling (based on the types of each entity).

Parameter tuning Lastly, we have shown that parameter tuning can have significant effects on our results (see 6.3.2 and 6.3.3). Due to a lack of time and material resources, we could not perform an extensive study of all parameters (DBpedia Spotlight parameters, word2vec parameters, SemanticExtractor parameters). We think that SemanticExtractor's overall performance could be improved by smartly tuning these parameters. Moreover, parameter tuning could be easier to perform once an automatic evaluation process is implemented.

6.5 Conclusion

We have proposed SemanticExtractor, a tool to extract analogy relations from natural text using DBpedia and word2vec. SemanticExtractor's overall results are mitigated, but we have good hope that the system could be significantly improved with different complementary approaches we have described, but not put into practice due to lack of time.

7 Conclusion and perspectives

We have presented the context of semantic relations extraction from natural text and several models for this NLP task, ranging from probabilistic language models to recent word embeddings, a vectorial representation of words. Although word embeddings (like word2vec) are powerful for semantic extraction, they are unable to characterize the nature of semantic relations precisely. With our SemanticExtractor tool, we have shown that Linked Open Data Cloud resources (such as DBpedia) could be used to label semantic relations obtained with word embeddings. Even if our system's overall results are currently limited, we do believe that they could be improved with further development, which we hope will occur, especially as no equivalent to SemanticExtractor has been brought to our attention so far.

References

Collin F Baker, Charles J Fillmore, and John B Lowe. The berkeley framenet project. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 86–90. Association for Computational Linguistics, 1998. URL http://www.aclweb.org/anthology/P/P98/P98-1013.pdf.

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 238–247, 2014. URL http://anthology.aclweb.org/P/P14/P14-1023.pdf.

Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992. URL http://www.aclweb.org/old_anthology/J/J92/J92-4003.pdf.

Lucien Carroll, Rebecca Colavin, Hannah Rohde, and Lara Taylor. Semantic role labeling. Northwester University, 2005. URL http://faculty.wcas.northwestern.edu/~hro501/presentations/CarrollColavinRohdeTaylor.sdsu.2005.pdf.

Olivier Chapelle, Bernhard Schölkopf, Alexander Zien, et al. Semi-supervised learning. 2006. URL http://www.acad.bg/ebook/ml/MITPress-%20SemiSupervised%20Learning.pdf.

Michael Collins. Language modeling - course notes for coms w4705. Columbia University, 2013. URL http://www.cs.columbia.edu/~mcollins/lm-spring2013.pdf.

R. Collobert and J. Weston. Deep learning in natural language processing. Tutorial at NIPS, 2009. URL http://ronan.collobert.com/pub/matos/2009_tutorial_nips.pdf.

Ronan Collobert and Jason Weston. Fast semantic extraction using a novel neural network architecture. In *Annual meeting-association for computational linguistics*, volume 45, page 560, 2007. URL http://ronan.collobert.org/pub/matos/2007_senna_acl.pdf.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning.

In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008. URL http://www.thespermwhale.com/jaseweston/papers/unified_nlp.pdf.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011. URL http://arxiv.org/pdf/1103.0398.pdf.

Scott C. Deerwester, Susan T Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *JAsIs*, 41(6):391–407, 1990. URL http://www.cob.unt.edu/itds/faculty/evangelopoulos/dsci5910/LSA_Deerwester1990.pdf.

Daniel Gildea and Daniel Jurafsky. Automatic labeling of semantic roles. *Computational linguistics*, 28(3):245–288, 2002. URL http://www.mitpressjournals.org/doi/pdfplus/10.1162/089120102760275983.

Joshua T Goodman. A bit of progress in language modeling. *Computer Speech & Language*, 15(4):403–434, 2001. URL http://arxiv.org/pdf/cs/0108005.pdf.

Djoerd Hiemstra. Language Models. In LING LIU and M.TAMER ÖZSU, editors, *Encyclopedia of Database Systems*, pages 1591–1594. Springer US, January 2009. ISBN 978-0-387-35544-3. URL http://dx.doi.org/10.1007/978-0-387-39940-9_923.

Dan Jurafsky and James H Martin. *Speech and language processing*. Pearson, 2014. URL https://lagunita.stanford.edu/c4x/Engineering/CS-224N/asset/slp4.pdf.

Dan Jurafsky and James H Martin. *Speech and language processing*. Pearson, 2015. URL https://web.stanford.edu/~jurafsky/slp3/22.pdf.

Slava M Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 35(3):400–401, 1987. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.129.7219&rep=rep1&type=pdf.

Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053*, 2014. URL http://arxiv.org/pdf/1405.4053.pdf.

Kevin Lund and Curt Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*, 28(2):203–208, 1996. URL https://tottdp.googlecode.com/files/lb96brmic.pdf.

Kathy McKeown. Introduction to natural language processing. Natural Language Processing Lecture - Columbia University, 2009. URL http://www.cs.columbia.edu/~kathy/NLP/ClassSlides/Class3-ngrams09/ngrams.pdf.

Tomas Mikolov, Quoc V Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, 2013a. URL http://arxiv.org/pdf/1309.4168.pdf.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013b. URL http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

Thomas R Niesler and PC Woodland. A variable-length category-based n-gram language model. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 1, pages 164–167. IEEE, 1996. URL ftp://svr-ftp.eng.cam.ac.uk/pub/reports/auto-pdf/niesler_icassp96.pdf.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Computational linguistics*, 31(1): 71–106, 2005. URL http://www.mitpressjournals.org/doi/pdfplus/10.1162/0891201053630264.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

Sameer S Pradhan, Wayne Ward, Kadri Hacioglu, James H Martin, and Daniel Jurafsky. Shallow semantic parsing using support vector machines. In *HLT-NAACL*, pages 233–240, 2004. URL https://www.researchgate.net/profile/Kadri_Hacioglu/

publication/220817505_Shallow_Semantic_Parsing_using_Support_Vector_Machines/links/0f317538c9b258eb45000000.pdf.

Douglas LT Rohde, Laura M Gonnerman, and David C Plaut. An improved model of semantic similarity based on lexical co-occurrence. *Communications of the ACM*, 8:627–633, 2006. URL https://www.cnbc.cmu.edu/~plaut/papers/pdf/RohdeGonnermanPlautSUB-CogSci.COALS.pdf.

Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014. URL http://arxiv.org/pdf/1411.2738.pdf.

Kristie Seymore and Ronald Rosenfeld. Scalable backoff language models. In *Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on*, volume 1, pages 232–235. IEEE, 1996. URL http://www.cs.cmu.edu/~roni/papers/scalable-TR-96-139.pdf.

Nianwen Xue and Martha Palmer. Calibrating features for semantic role labeling. In *EMNLP*, pages 88–94, 2004. URL http://verbs.colorado.edu/~xuen/publications/emnlp04.pdf.