Center for Big Data Statistics

Working paper

WordGraph2Vec

Using language constructs to create sentence embeddings

Working paper no.: 07-20

In collaboration with UWV

Paul Keuren
Dominik Blatt
Marc Ponsen
Bob van den Berg

Oct 2020

Contents

Abstract

It is estimated that the bulk of today's data is unstructured. This so-called Big Data contains huge amounts of information relevant to statistical institutes such as Statistics Netherlands (SN). Text documents posted on websites is one such category. Deriving information from texts is challenging. The data does not have a pre-defined data model and might be noisy. Traditional statistical techniques, used at statistical institutes, are therefore ill suited to analyse such data. In the recent years, great strides forward have been made with dealing with this type of data, most notably in the field of data mining, natural language processing (NLP) and text analytics.

In this paper, we propose a novel algorithm WordGraph2Vec (WG2Vec) to analyse unstructured text data. The proposed algorithm combines two aspects of NLP: so called **word graphs** and word embeddings. First, Word graphs are used to dissect and understand a text on a grammatical level, i.e., what is the role of words and how do they link to each other? As a next step the semantics for a word graph are obtained using wordembeddings models, such as Word2Vec. Words and phrases will be converted into a vector of numbers, where the semantic meaning is captured in the numerical vector (i.e., the phrases with vectors in close proximity to each other hold the same semantic meaning). These two steps are at the core of WG2Vec.

We present experimental results that show that WG2Vec outperforms a straightforward extension of Word2Vec in a labour market-use case. In this use-case, we match skills obtained from vacancy texts with validated skills in an expert system. We compare the matches from either algorithm with expert annotations made by the Dutch Employee Insurance Agency (UWV). This use-case is part of the CBS contribution to an Interreg project called 'Werkinzicht'. In this project, CBS works together with UWV and VDAB to generate cross-border insights about the Labour market.

The goal of this particular use-case is to analyse Big Data texts (such as online vacancies) to support the UWV in efficiently maintaining their expert system (a labour market ontology of jobs and connected skills). The maintenance of this expert system basically consists of two tasks, (1) finding synonyms for existing skills in online texts, and (2) finding new skills and jobs currently absent in the ontology. This up-to-date system creates an 'universal language of jobs and skills', which is relevant for both UWV and SN. For UWV, this allows them to efficiently match supply and demand on the granular level of skills. SN, on the other hand, can compute (timely) demand for specific skills and (potentially future) jobs. Combining this timely and detailed view of the labour market with the already existing register data at SN, can potentially provide complete new insights and statistics for a rapidly developing labour market. The recent COVID-19 crisis, and it's expected impact on the labour market, has made the need for new data sources to create rapidly available statistics to match offer and demand even more relevant.

1 Introduction

It is estimated that the bulk of today's data is unstructured. Crude estimates are that 80 to 90 percent of today's data is text-based. Some seminal examples are the texts on web pages (such as Wikipedia), tweets, google queries and people's social media content. This Big Data contains vast amounts of information (see Gentzkow, 2019) relevant to statistical institutes, such as Statistics Netherlands (SN). However, deriving information from texts is challenging. The data do not follow a pre-defined data model and thus contains information that can not be easily extracted. Traditional statistical techniques, typically used at statistical institutes, are therefore not particularly suited to analyse such data. Fortunately, developments have been made with dealing with this type of data, most notably in the field of data mining, natural language processing (NLP) and text analytics. Many different methods have been developed to find patterns in text, interpret the patterns and finally extract information from them. In this paper, two methods are of importance, namely *part-of-speech tagging in language models* and *wordembeddings*.

Part-of-speech tagging is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition and its relationship with adjacent and related words in a text (Cutting et al., 1992). Creating a part-of-speech tagger (POS) requires extensive domain expertise, as for example one must study the properties of each language so as to build linguistic resources, select appropriate features, configure parameters and set exceptions in the POS system. As with any system that relies on domain expertise, maintaining it can be time consuming and the quality may be negatively affected by biases of the human domain experts. Also, a grammatical understanding of text does not imply that we can derive the text's subject. In other words, the semantics are still unknown.

A fundamentally different approach, one that does capture semantic meanings of words or texts (and does not require any form of domain expertise in the process), are the wordembeddings algorithms. This purely data-driven approach use supervised machine learning to represent words as vectors. The semantic meaning of words is found by analyzing words surrounding the target word. Put differently, the vector of the words "man" and "woman" are closer to one another compared to "man" and "cat" (and this is derived from the context of these words). Wordembeddings require huge amounts of data, and may also contain biases, but these models have shown great impact on the natural language processing community.

Fully automated process of wordembeddings has two advantages: 1) Since language is continuously developing it is key to remain up to date, by creating an automated process remaining up to date requires little effort; 2) there is no expert bias. However, there is still great value in using the information extracted by language models. In this paper, we propose a new algorithm **WordGraph2Vec** (WG2Vec) that combines the best of both worlds. The grammatical understanding of texts given by POS systems is used to extract relevant parts in a text, that we call **WordGraphs** (essentially combination of connected words). As a next step the semantics are obtained by transforming WordGraphs to vectors using wordembeddings models.

In this paper we present results where the WG2Vec algorithm outperforms an extension of Word2Vec. The experiments involve a labour market use-case, where we match skills obtained from vacancy texts with validated skills in an expert system and compare the matches from either algorithm with expert annotations made by the Dutch Employee Insurance Agency (UWV). This use-case is part of the CBS contribution to an Interreg project called 'Werkinzicht'. In this project, CBS works together with UWV and VDAB to generate cross-border insights about the Labour market. We will detail the outline of the remainder of this paper in the next section.

1.1 Outline of paper

In section 2, we discuss related work on techniques of natural language processing (NLP) models that can deal with unstructured text. We will also focus on the family of machine learning (ML) algorithms that produce word or sentence embeddings. These algorithms convert text to numbers, thereby modelling the meaning of the words. Next, we will briefly discuss NLP that can be used for various applications, e.g., part-of-speech tagging and text tokenization. In addition, some advanced ML and/or NLP models are discussed. In section 3 all steps involved in our novel WordGraph2vec (WG2Vec) algorithm are explained in detail and we explain how it differs from the above algorithms. Next, in section 4 we introduce a labour market use-case and evaluate WG2Vec against a competing algorithm for this use-case. We will discuss our experimental methodology and the experimental results. Finally, in section 5 we conclude and propose directions for future work.

2 Related Work

In texts it is often of interest to figure out the semantic relation between the words, i.e., what is being done and on what subject. This semantic relation can be extracted through dependency parsing. Dependency parsing is the task of analyzing the grammatical structure of a sentence and establishing the relationships between word. This can be done by either using supervised (Kiperwasser and Goldberg, 2016) or unsupervised (Headden III et al., 2009) machine learning.

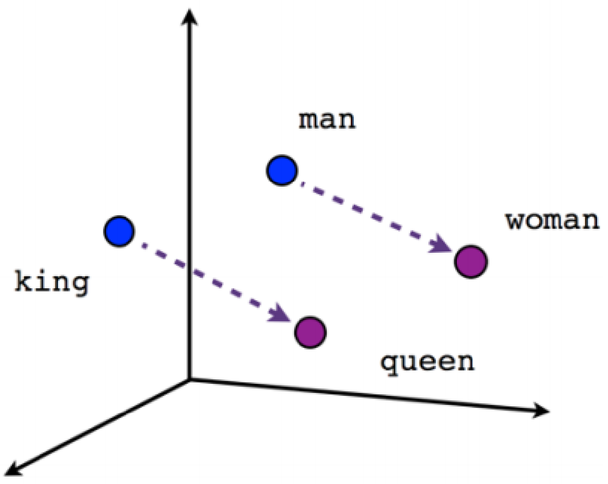
A different approach are the so called wordembedding models (Mikolov et al., 2013; Pennington et al., 2014). With wordembeddings, similar words are placed in close proximity in a vector space. These models are trained by using a bag-of-words approach, so their order and function in the sentence is not taken into account. We will describe these models in more detail below, explain their relevance to our specific use-case, and finally explain how our novel algorithm WordGraph2Vec combines the two models.

2.1 Word2vec

Word2vec is a one-hidden-layer neural network that processes text by "vectoring" words. Its input is a text corpus and its output is a set of numerical vectors, one for each word in the text corpus. For example, a trained Word2Vec model may represent the word *"man"* as follows:

$$\begin{bmatrix} 2.2915103 & 3.7818744 & -1.7644048 & \dots & -0.9715823 & -1.2458154 & -4.5027633 \end{bmatrix}$$

The vectors we use to represent words are called neural **word embeddings**. Why would we want to represent a word as a series of numbers? The purpose and usefulness of Word2vec is that it groups vectors of (semantically) similar words together in a vector space. In other words, it detects similarities between words mathematically, and does so fully automatically (without human intervention and without domain knowledge). Given enough data Word2vec

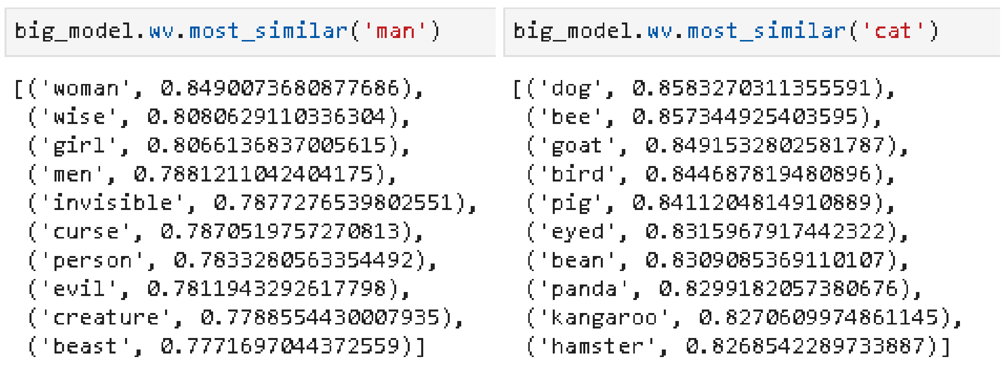


2.1 A two-dimensional plot of the word vectors of the words ‘man’, ‘woman’, ‘king’ and ‘queen’. The figure intuitively shows the association between these words and even allows for calculations (e.g., king - man + woman = queen.

can be used to establish a word’s association with other words (e.g. “man” is to “king” what “woman” is to “queen”, see Figure 2.1), or cluster sentences or documents and classify them by topic.

The output of the Word2vec neural net is a vocabulary in which each item has a vector attached to it. Similarity between vectors can be measured with the *cosine similarity*: vectors with a 90 degree angle have no similarity (i.e., a similarity value of 0), while a 0 degree angle corresponds to total similarity (i.e., a similarity value of 1). In Figure 2.2 we can find the most similar words according to the cosine similarity for the words *“man”* and *“cat”*. As expected, “woman” ranks highest compared to “man”, in other words, their vectors are most similar in terms of cosine similarity. This again shows that similarities were established based on the semantics and not based on a string metric, such as the Levenshtein distance. This is useful for matching semantically equivalent sentences, for example when comparing descriptions of *skills*, which is the focus in our labour market use-case.

Word2vec is similar to an auto-encoder, it encodes each word in a vector by training the word against the neighbouring words in the document corpus, i.e., its **context**. It does so in one of



```
big_model.wv.most_similar('man')
```

```
[('woman', 0.8490073680877686),
 ('wise', 0.8080629110336304),
 ('girl', 0.8066136837005615),
 ('men', 0.7881211042404175),
 ('invisible', 0.7877276539802551),
 ('curse', 0.7870519757270813),
 ('person', 0.7833280563354492),
 ('evil', 0.7811943292617798),
 ('creature', 0.7788554430007935),
 ('beast', 0.7771697044372559)]
```

```
big_model.wv.most_similar('cat')
```

```
[('dog', 0.8583270311355591),
 ('bee', 0.857344925403595),
 ('goat', 0.8491532802581787),
 ('bird', 0.844687819480896),
 ('pig', 0.8411204814910889),
 ('eyed', 0.8315967917442322),
 ('bean', 0.8309085369110107),
 ('panda', 0.8299182057380676),
 ('kangaroo', 0.8270609974861145),
 ('hamster', 0.8268542289733887)]
```

2.2 Similar words for two different words using a vanilla word2vec model trained on an English based text corpus.

two ways, either using *context* to predict a *target word* (a method known as continuous bag of words, or CBOW), or using a *word* to predict a *target context*, which is called skip-gram. The top of Figure 2.3 illustrates how a word2vec dataset might look like, for either training mode. The data consists of tuples of *word* and *it's context, i.e., words surrounding the word*.

In the remainder of this paper - and in our project - we focus on the skip-gram variant (see Figure 2.3). As mentioned before, the input of skip-gram is one word and the learning task is to best predict the context. The input word is represented as a vector using one-hot vector encoding. This vector contains exactly one element per distinct word in the text corpus. All but one element are labeled as '0'. Only the element that corresponds to our input word is labeled '1' (as for example, see the left-hand side of Figure 2.3). The input vector is given to a neural network with a single hidden layer. The output of the network is a single vector (of the same size of the input vector) containing, for every word in our vocabulary, the probability that this word was used nearby our input word. Our final word embedding is stored in the hidden layer. We can extract per word its n-dimensional vector (in our example, the network consisted of 300 hidden neurons) via matrix computations. A well trained set of word vectors will place (semantically) similar words close to each other in that space, as we already saw in Figure 2.1.

2.2 Paragraph2Vec

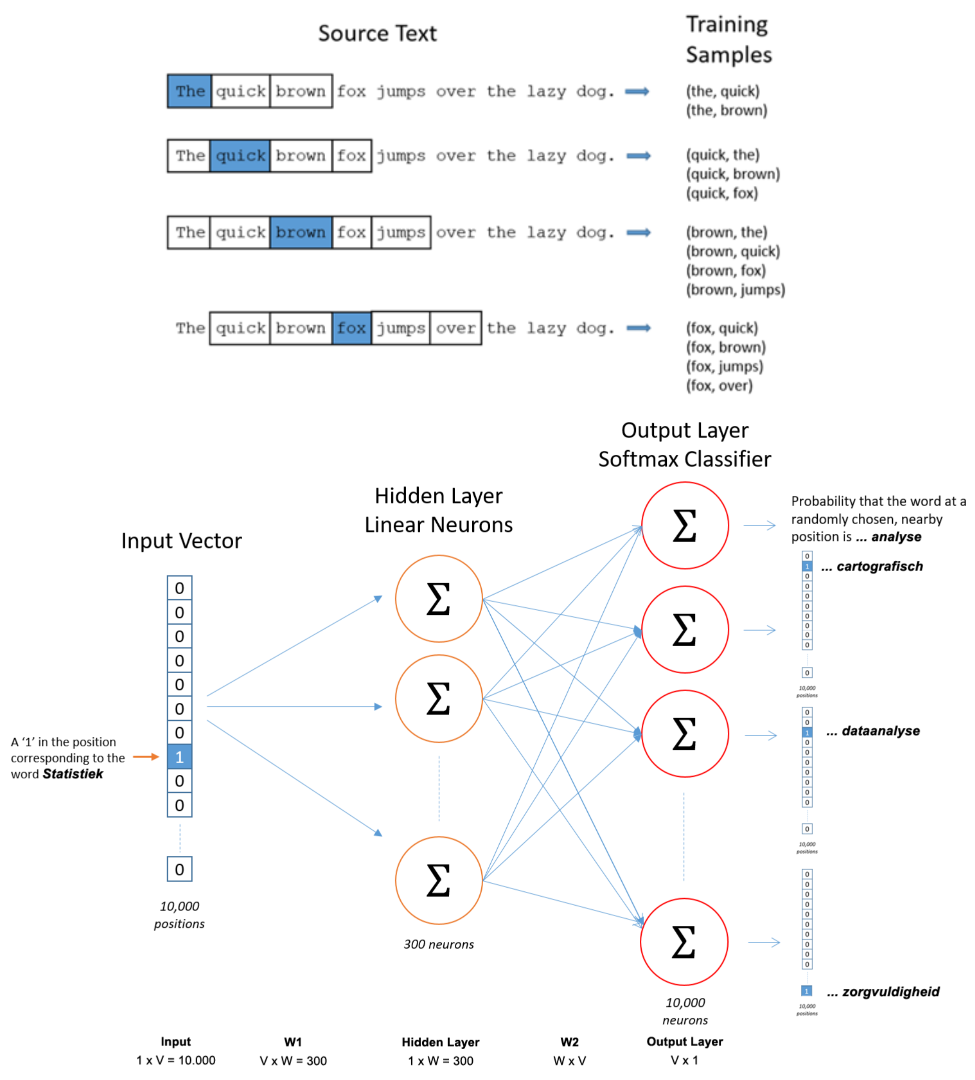
A drawback of Word2vec, like many machine learning approaches, is that it can not deal with input of varying length. They require the input to be represented as a fixed-length feature vector. In many use-cases, and most certainly in our labour market use-case, our input data are sentences, possibly even complete documents. When it comes to larger texts, one of the most common fixed-length approaches is bag-of-words (BOW). BOW has two major weaknesses: (1) it loses the ordering of the words and (2) ignores the semantics of words. Word2vec, on the other hand, captures semantic similarities but can only do so for words.

A simple approach to extend Word2vec to full texts is using a weighted average or concatenation of all the words in the document. In the remainder of this paper, we refer to this as **Avg-Word2vec**. However, this also loses the word order in the same way as the standard bag-of-words models do. Le and Mikolov, 2014 proposed the **Paragraph Vector** algorithm, an unsupervised algorithm that learns fixed-length feature representations from variable-length pieces of texts, such as sentences, paragraphs, and documents. This is achieved by extending Word2vec by training both word- as well as paragraph vectors. The paragraph vectors are unique among paragraphs, the word vectors are shared. Word vectors are trained as was described in Section 3. The paragraph vector are concatenated or averaged word vectors. At prediction time, the paragraph vectors are inferred by fixing the word vectors and training the new paragraph vector until convergence. More specifically, every paragraph is mapped to a unique vector, represented by a column in matrix D and every word is also mapped to a unique vector, represented by a column in matrix W (see Figure 2.4). The paragraph vector and word vectors are averaged or concatenated to predict the next word in a context. The paragraph vector is shared across all contexts generated from the same paragraph but not across paragraphs. The word vector matrix W, however, is shared across paragraphs. i.e., the word vectors are the same for all paragraphs.

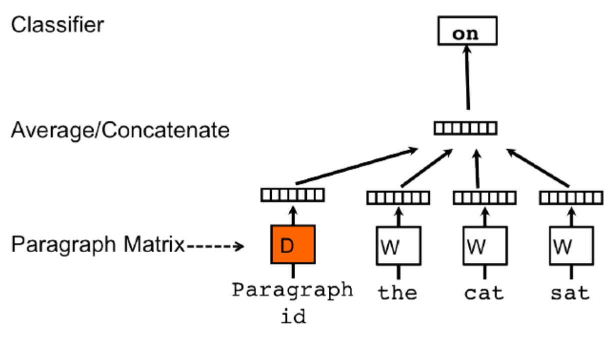
2.3 Language models

Another method for comparing texts is by using language models. A language model is a set of techniques that focus almost exclusively on one language. Using these models it is possible to derive the root of a verb (stemming/lemmatisation) and whether a word is a noun (part-of-speech tagging). Almost every language has its own grammar and spelling rules and a language model therefore needs to fit these specific rules if it is to search for certain constructs. A language model comprises multiple techniques for a singular language and can be used for a variety of goals, however for our research we are only interested in Part-Of-Speech tagging (POS-tagging) and dependency parsing. POS-tagging is a technique where each word is assigned a category. These categories consist of nouns, verbs, adjectives, conjunctions etc. This task of classifying words into categories is not trivial. Some of the categories are infinitely large and some have overlap in the words that can occur. An example of this is with nouns and verbs: both groups are infinitely large and can overlap. The word 'object' can be both a noun 'I hold the object in my hand.' and a verb 'I object to his statement.'. These problems make it difficult to correctly classify all words in a text. The goal behind dependency parsing is to link all items in a text. These links give additional information on what the sentence is describing. For example the sentence 'The yellow ball is rolling.'. In this sentence 'yellow' (an adjective) gives information about the 'ball' and the verb 'is' refers to the subject 'the yellow ball'. Many dependency parsers use POS-tagging (Buchholz and Marsi, 2006) and as such might struggle if tagging is done wrong. Just as misspelled words might get a wrong POS-tag, a sentence that does not follow grammar might be given wrong dependencies.

In the next section, we will describe our technique that uses both wordembeddings and information obtained from language models. The combination of both methods is novel. We will also explain where our algorithm differs from the above described approaches.



2.3 On the top we find an example dataset for the word2vec models, consisting of tuples {current word, context}, where one can choose if the current word is used as input or as target prediction (and vice versa for the context). Below that, we find the "skip-gram" model that is used in the current research.



2.4 A framework for learning paragraph vectors, taken from Le and Mikolov, 2014. A similar learning approach compared to Word2vec, but an additional paragraph vector is learned to deal with multiple words.

3 WordGraph2Vec

In this section, we will propose a novel algorithm, WordGraph2Vec (WG2Vec). It was developed to detect similarity in text information and combines elements from the techniques discussed in the previous section. The first component is wordembeddings, familiar to Word2Vec. It translates words into vectors, so that semantic similarities between words can be measured mathematically. The other component is the language model, which contains (domain) knowledge and is applied to extract sets of words from sentences with certain syntactical functions, like the verb(s) and object(s) of a sentence. Only words that are expected to carry relevant information are kept. These words and the relationship between them can be expressed in a **word graph**.

Our WordGraph2Vec algorithm combines the two components, words graphs and wordembedding. It detects similarity in text information by joining a syntactical understanding of sentences with a semantic representation of words.

We will discuss all steps involved in WG2Vec in the following sections.

3.1 Understanding a sentence

Sentences can be noisy, complex constructs. They may contain boilerplate text, which may not give very insightful information about the meaning. Furthermore, people might concatenate sentences, as for example:

– "You drive big trucks and cars."

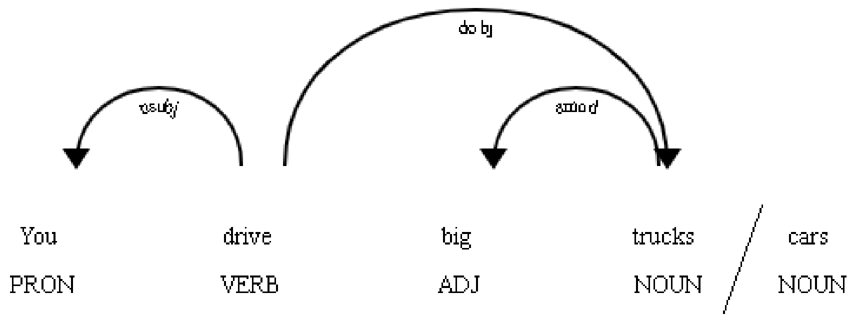
The problem with this sentence is in the number of combinations that can arise from these sentences. The example shown above has two different pieces of information. Each of these pieces requires a separate analysis to find the best match.

For this reason, WG2Vec first pre-processes sentences and splits them into elementary parts. This is done based on a rule-based logic that uses information provided by the language model. The sentence mentioned before could be split into multiple parts:

– "You drive big trucks."
– "You drive big cars."

These elementary pieces of information can be further analysed to better understand their meaning. Techniques such as Paragraph2vec will incorporate boilerplate text in their learned sentence embedding. We hypothesise that better results are obtained once we exclude them. However, pinpointing the relevant parts of a sentence is not an easy task. WG2Vec uses part-of-speech tagging with dependency parsing provided by language models to identify relevant parts in the pre-processed texts. We refer to these parts as *word graphs* in the remainder of this paper.

Let us give three examples of *word graphs* that were tailored to the labour market use-case. We are particularly interested in which skills people have. Therefore, we defined an **Activity** word graph. A set of two words is an **Activity** if and only if a VERB refers to a NOUN, return "VERB NOUN". The activities in our example sentence are (see Image 3.1):



3.1 A language model provides additional details on words, such as their word type (e.g., noun, adjective etc.) and their dependency to other words. WG2Vec uses this information to extract relevant parts from sentences.

- "drive (VERB) trucks (NOUN)"
- "drive (VERB) cars (NOUN)"

Within this activity, we may believe that the noun may hold more or less relevance compared to the verb. For that reason we explicitly also define a **Leading** word graph. So in our example sentence, both ”trucks” and ”cars” are Leading word graphs.

Finally, we might loose specific information when we only focus on VERB related information stated above. When the author of a text gave more information on a specific noun, he or she did so with a reason. To incorporate this, we introduced a **Specific** word graph.

A set of two words is a **Specific** if and only if an ADJECTIVE refers to a NOUN, return only the "ADJECTIVE NOUN". If a group of ADJECTIVES refer to the same NOUN, each ADJECTIVE gets a separate **Specific** word graph. In our example sentence, this yields into the following specifics:

- "big (ADJECTIVE) trucks (NOUN)"
- "big (ADJECTIVE) cars (NOUN)"

It is not always the case that these parts are found in sentences. As a fallback we also include take single nouns and verbs into account. All these parts are taken into account when representing the text in vector space, as will be described in the next Section.

3.2 Representing a sentence as a vector

In the next step, we convert the word graphs to a vector (i.e., sentence embedding). At the source of this is Word2vec, that produces the wordembeddings. Sentence embeddings in WG2Vec are constructed by appending the wordembeddings. Let us illustrate, we have a word graph of two words w_1 and w_2, with each two vectors of length 300.

$$w_1 = \{a_1, a_2, a_3, .., a_{298}, a_{299}, a_{300}\}$$

$$w_2 = \{b_1, b_2, b_3, .., b_{298}, b_{299}, b_{300}\}$$

WG2Vec then constructs a new sentence embedding by appending the two word embeddings, as opposed to the Paragraph2Vec algorithm which concatenates or averages them (see Section 2). More specifically, the sentence embedding for this word graph np_1 is now of length 600:

$$np_1 = \{a_1, a_2, a_3, ..., a_{298}, a_{299}, a_{300}, b_1, b_2, b_3, .., b_{298}, b_{299}, b_{300}\}$$

This is an example for a two-word word graph. In the case of a single word, obviously, the Word2vec embedding is used. Generally speaking, the length of the sentence embedding is a multiple of the number of hidden neurons chosen for Word2vec.

3.3 Defining sentence similarity using WordGraph2Vec

The previous section illustrates how texts are transformed into a numeric vector by WG2Vec. The varying length of the vectors require a different approach when comparing them to each other (to determine semantic similarity). Let us now describe a similarity function that can deal with the proposed (variable length) word graphs. One such similarity function for WG2Vec is denoted in Algorithm 1. This algorithm takes two (simplified) sentences as a input that will be transformed into word graphs, as was described in Section 3.1. In the pseudo-code this is done in lines 4 and 5. The *word-graph-type* gives the type of the word graph, such as *specific* or *leading*.

Similarity is then based on exact (textual) matches of parts, with the so-called *overlap score* (see lines 6 and 7), as well as semantic similarity on different parts with the *difference score* (see lines 8 to 17).

Different elements are vectorized as was described in 3.2. Cosine similarity gives the similarity (see line 15) for word graphs of the same type (and therefore of same length). Note that, we provided a kernel *K* per word graph, which allows us to weigh the different words. As for example, in the activity (i.e., a related verb and noun pair) the kernel may specify that the noun-part is more relevant by multiplying the corresponding numbers in the vector with a specified value. A higher value would make that part more relevant when comparing. Per word graph, the addition of overlap- and difference score, weighted by the so-called Vector Kernel V, leads to the final score. To be more precise, the word graph kernel weighs the words in the word graph, whereas the vector kernel weighs the overall score of the word graph. Once all (weighted) overlap- and difference scores are processed, the final score gives the similarity between the two sentences. Higher numbers represent better matches.

Algorithm 1 One suggested similarity score used by WordGraph2Vec.

Require: Word2vec W, Language model L, Word Graphs N, Word Graph Kernel K, Vector Kernel V

```
1:  WG2V_SIMILARITY(Sentence $S_1$, Sentence $S_2$):
2:  $final\_score = 0$
3:  for Each word_graph_type $n \in N$ do
4:      $A = S_1[word\_graph\_type]$
5:      $B = S_2[word\_graph\_type]$
6:      $intersection = A \bigcap B$
7:      $overlap\_score = |intersection|$
8:      $symmetric\_difference\_A = (A - B)$
9:      $symmetric\_difference\_B = (B - B)$
10:     $difference\_score = 0$
11:     for Each word_graph_type $i \in symmetric\_difference\_A$ do
12:         for Each word_graph_type $j \in symmetric\_difference\_B$ do
13:             $V_i = Wordgraph2vec(i)$
14:             $V_j = Wordgraph2vec(j)$
15:             $difference\_score+ = cosine\_similarity(V_i, V_j) \cdot K$
16:         end for
17:     end for
18:     $final\_score+ = (overlap\_score + difference\_score) \cdot V$
19: end for
20: return $final\_score$
```

4 Experiments

In this Section we will benchmark WordGraph2Vec (WG2Vec) against the averaged variant of Word2vec (AvgW2Vec). We will start with describing our experimental use-case: matching skills in the labor market. Next we will explain our experimental methodology and finally the results.

4.1 Labour market use-case: from jobs to skills

At the start of this project, funded by the Interreg 'Werkinzicht' project, there was growing tension in the Dutch labor market. Tension is defined by SN as the number of open vacancies per 100 unemployed people. Mid 2019 the tension was 93. This means there were almost as much open vacancies as there were unemployed people. This raises the question why it was still so difficult to employ the remaining unemployed people. Apparently, the supply (unemployed people) and the demand (open vacancies) didn't match. The current COVID-19 crisis will have considerable impact on the labout market tension. However, the problem of matching supply and demand in the labour market might become even more relevant since this crisis may have a permanent effect on specific economic sectors and the associated job profiles. It is generally assumed that matching supply and demand can be better achieved on the more granular level of *skills* as opposed to matching on *job titles* alone. Gaining insight into supply and demand of (new) skills is valuable for national statistics, e.g., for reporting trends of new skills, revision of curricula, support people who are seeking jobs with educational programs.

For this project, SN is collaborating with the Dutch (UWV) and Flemish Public Employment Services (VDAB) to create an open source ontology for the Dutch labor market that describes all existing jobs and skills. This ontology could be used to make much more fine grained matches between vacancies and people seeking a job within the Netherlands and across the border in Flanders. It also allows to create cross-border statistics for the labor market. That is why this part of the project is developed with funds from the Interreg Werkinzicht project.

To keep the ontology up-to-date is a labor intensive process. SN, therefore, strives to help the domain experts of UWV to automatically process available big data sources and use the previously described WG2V algorithm to provide suggestions for potential new skills and jobs that should be added to the ontology. The big data in this case are job vacancies posted at online websites both for the Netherlands and Flanders.

The learning task then is to match skills extracted from vacancy texts with (validated) skills incorporated in the ontology. The main aim is to find synonyms in the ontology for our input, or to actually discover new skills currently not present in the ontology. We call this algorithm **BI**g **D**ata **O**ntology **E**nrichment, see Algorithm 2. Note that we currently use vacancy texts as input for potential skills, though in practice this can be any kind of document or text. BIDOE is given an input sentence containing a potential 'skill', extracted from vacancy texts. It then computes similarity scores with skills in the ontology. The matching can for example be done with WG2Vec computes similarity scores between two sentences. BIDOE finally returns the best *n* matches, i.e., those with the highest ranked similarity scores.

Algorithm 2 Outline of the Big-Data Ontology Enrichment algorithm (BIDOE).

Require: Big data skill S_1, Ontology skills O_S, #Suggestions n

```
1: BIDOE_MATCH(Sentence $S_1$):
2: $match\_scores = []$
3: for Each $O\_Sj \in O\_S$ do
4:     $match_score = SIMILARITY(S_1, OS\_j)$
5:     $match\_scores[j] = match\_score$
6: end for
7: return $best(match\_scores, n)$
```

The idea is to use WG2Vec to determine the similarity between two skills. However, before we can use WG2Vec, we first need to establish if it can successfully match skills. For that reason we conducted an experiment which is detailed in the next Section.

4.2 Experimental methodology and results

We compared three algorithms that create sentence embeddings in the experiments, namely (1) an algorithm that compares texts based on the occurrence of literal word graphs (WG), (2) Word2Vec extended to sentences (AvgW2Vec) and finally (3) our proposed algorithm WG2Vec which combines elements from the first two algorithms. Effectively, this means that we use three different similarity functions in the BIDOE algorithm (see line 4 in Algorithm 2). For the similarity function of WG2Vec we use Algorithm 1 which was explained in section 3.3. For AvgW2Vec we simply take the averaged cosine similarity for all words in the sentence as the similarity. WG only sums the overlap score in Algorithm 1.

To be able to compare these algorithms, we need a ground truth. In other words, we need correct bindings between input and output skills (in the ontology). We collaborated with the public employment service of the Netherlands (UWV) to produce such a ground truth. We presented UWV with a set of input skills. Based on an earlier version of BIDOE, we provided UWV with ten suggestions. UWV then mapped the best suggestion to the input skill, more specifically they labelled the best match as '1', the second-best as '2' and so-forth. If the suggestions did not contain valid match, UWV was able to browse the ontology for better matches. In total, UWV made 97 annotations. Based on this dataset, we can compute various metrics for our benchmark. The following metrics are used:

– **Top-n**: BIDOE presents the top *n* matches to domain experts. This metric measures the number of times that the skill ranked by UWV is in the top-*n*. A high score is better.
– **Rank difference**: the difference between annotation and BIDOE rank. For each skill (s) where there is an overlap between the BIDOE result *j* and an annotation *i* (it counts as a Top-n hit), a difference in ranking is computed. This basically measures the uncertainty on the predictions. A low Rank Difference is better. There are three different computations to give a good indication of the Rank difference.
  RD-AVG This computation gives an idea of the average offset of a match.
  RD-MIN To offset the average (RD-AVG), a min value is added to illustrate the average the best rank over all skill annotations.
  RD-SCR Both RD-AVG and RD-Min depend on the Top_n. If a different Top_n is used the results are no longer comparable. This score RD-SCR normalises between 0 and 100 where a value of 100 indicates a perfect score.

$$\begin{aligned}
RD_{avg} &= (\sum_{s}^{S} \frac{\sum_{i}^{s_a} Min(|i-j|, Top_n)}{||s_a||})/(||S||) \\
RD_{min} &= (\sum_{s}^{S} i \in s_a Min(|i-j|, Top_n)/(||S||) \\
RD_{scr} &= 100 * (\sum_{s}^{S} 1 - (\frac{\sum_{i}^{s_a} Min(|i-j|, Top_n)}{||s_a||})/Top_n)/(||S||)
\end{aligned} \tag{1}$$

The Rank Difference scoring is shown above in the set of equations 1. The S is the set of skills and s_a is the set of annotations for a skill. The Top_n is the maximum number of matches returned by the matching algorithm. For the result table a Top_n of 10 was used. It should be noted that WG2Vec was given its parameters via a random search of the hyper-parameters (the word graph and vectors kernels) in the range of 0 to 20. It was found that the **specific** part was most important (with a kernel of 19), followed by **activity** (14), **noun** (5), **verb** (2) and lastly the **leading** (1). We set the vector kernels to 15 and 19 for the **activity** (i.e., the noun is made more important) and 16 and 14 for the **specific**. AvgW2Vec does not require any parameter tuning.

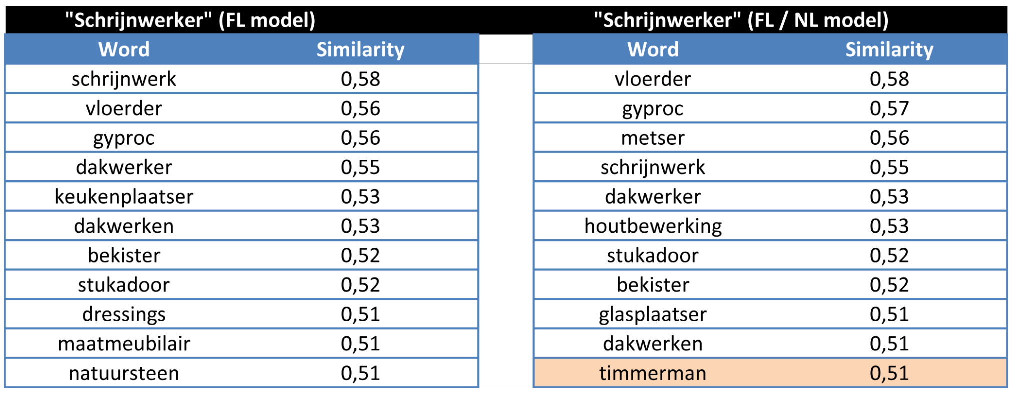
The results of the our benchmark algorithms are given in Table 4.1. From the results we clearly see that a literal match of word graphs (in other words, only doing a literal match on syntactic understanding of the text) does not work at all. We also see that WG2Vec outperforms AvgW2Vec for the TN metric, the other metrics show similar scores. In particular the high Top-n score is important. Out of 90 annotations, 23 percent of expert annotations are retrieved. We realize that these results may be biased to some extent. Experts were given a list of suggestions (made by earlier version of the WG2Vec algorithm) to speed up the annotation-making process. However, they were also allowed to manually browse skills in the ontology and annotate those. Still, it is possible that the suggestions by a same type of algorithm introduced a bias in the experimental results. However, a (expert validated) synonym is valid, regardless how it was acquired.

4.1 Empirical results of the experiments with BIDOE.

Algorithm	TN	RD-AVG	RD-LOW	RD-SCR
WG	1% (1)	0.00	0.00	0
AvgW2Vec	11% (11)	2.58	2.00	74
WG2Vec	23% (22)	2.67	1.88	73

4.3 Cross-border language

One important aspect of our particular use-case is the fact that we are dealing with vacancies in different languages (i.e., Dutch and Flemish). Both languages have strong similarities, but some concepts are described differently. The fact that vanilla Word2Vec is used to capture semantically similar concepts helps in this particular use-case. We did several tests with one Word2Vec model trained on a text corpus of Flemish vacancies (FL) and one model trained on both Flemish and Dutch vacancies (FL/NL). We then queried the word *schrijnwerker*, which is the Flemish word for *carpenter*. The Dutch synonym for *schrijnwerker* is *timmerman*. Image 4.1 shows the strength of using Word2Vec as the *engine* for capturing semantics between two similar languages: if we ask the Flemish model for similar concepts to *schrijnwerker*, the Dutch synonym *timmerman* is not among the top results. It appears that this concept wasn't often used in the Flemish text corpus. If we use the model that was trained on vacancies for both languages, the Dutch synonym was among the top results. This has consequences for cross-border matching of supply and demand. More specifically, a resume of a *timmerman* might not be matched with a *schrijnwerker* vacancy, because the semantic similarity is lost. Using the full model (FL/NL) a match was possible. This is an interesting property of Word2vec, that can be a strong benefit in our specific (cross-border labour market) use-case.



"Schrijnwerker" (FL model)			"Schrijnwerker" (FL / NL model)	
Word	Similarity		Word	Similarity
schrijnwerk	0,58		vloerder	0,58
vloerder	0,56		gyproc	0,57
gyproc	0,56		metser	0,56
dakwerker	0,55		schrijnwerk	0,55
keukenplaatser	0,53		dakwerker	0,53
dakwerken	0,53		houtbewerking	0,53
bekister	0,52		stukadoor	0,52
stukadoor	0,52		bekister	0,52
dressings	0,51		glasplaatser	0,51
maatmeubilair	0,51		dakwerken	0,51
natuursteen	0,51		timmerman	0,51

4.1 Two Word2vec models were trained, one on Flemish vacancies only (FL) and one on both Flemish and Dutch vacancies (FL/NL) on the labour market. The Dutch synonym for 'Schrijnwerker' is lost in the FL model, but can be found in the FL/NL model.

5 Conclusions and future work

In this paper, we proposed a novel algorithm **WordGraph2Vec** (WG2Vec) that combines different types of natural language processing (NLP) methods. First, the grammatical understanding of texts is used to extract relevant concepts in a text. We extract concepts from text, referred to in the paper as **word graphs**. An example are *activities* in skill texts. At this point WG2Vec has recognized these concepts to be relevant - based on a grammatical understanding of the sentence only - but it still has no idea what they mean, semantically. As a next step the semantics are obtained for these word graphs using wordembeddings models. In other words, the phrases will be converted into a vector of numbers, where the semantic meaning is captured in the vector (i.e., the phrases with vectors in close proximity to each other hold the same semantic meaning).

We present experimental results wherein we show that WG2Vec outperforms a straightforward extension of Word2Vec and an algorithm using only part-of-speech tagging in a labour market-use case. In this use-case we match skills obtained from vacancy texts with validated skills in an expert system, and compare the matches from either algorithm with expert annotations made by the Dutch Employee Insurance Agency. Although more experimentation is required (in particular on a larger, unbiased dataset), we conclude that WG2Vec's grammatical and semantic understanding helps in truly grasping the meaning of a piece of text.

We want to strengthen these claims in future research. For one, we want to benchmark WG2Vec on a larger test set and against more advanced algorithms, such as paragraph2vec (Le and Mikolov, 2014). Furthermore, WG2Vec can be further improved by making more elaborate word graphs. Also, the 'semantic engine' which is now Word2Vec could be replaced by more advanced algorithms.

The above are suggestions for improvements for the WG2Vec algorithm. Another crucial research track for SN is utilizing the techniques discussed in the paper for making timely and detailed statistics on the labour market, both from the perspective of demand and supply. By reliably linking skills found in vacancies to validated skills in the labour market ontology, SN can count the occurrence of skills in a given time period. With these counts of validated skills, SN can compute timely and detailed labour market statistics. Which skills are currently in demand? Which jobs are increasing in popularity? Which new jobs emerge, and what are its associated skills? These questions can in principle be answered, given a thorough understanding of vacancies and open (and validated) labour market ontology.

Acknowledgements

Part of this research was supported by the 'Werkinzicht' Interreg project. We thank the partners in this project, especially UWV, for providing feedback on our research approach and results. The reviewers are gratefully acknowledged for their comments and suggestions.

References

Buchholz, Sabine and Erwin Marsi (2006). “CoNLL-X shared task on multilingual dependency parsing”. In: *Proceedings of the tenth conference on computational natural language learning (CoNLL-X)*, pages 149–164.

Cutting, Douglass et al. (1992). “A practical part-of-speech tagger”. In: *Third Conference on Applied Natural Language Processing*, pages 133–140.

Gentzkow Kelly, Taddy (2019). “Text as Data”. In: *Journal of Economic Literature*, pages 535–574.

Headden III, William P, Mark Johnson, and David McClosky (2009). “Improving unsupervised dependency parsing with richer contexts and smoothing”. In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 101–109.

Kiperwasser, Eliyahu and Yoav Goldberg (2016). “Simple and accurate dependency parsing using bidirectional LSTM feature representations”. In: *Transactions of the Association for Computational Linguistics* 4, pages 313–327.

Le, Quoc V. and Tomas Mikolov (2014). “Distributed Representations of Sentences and Documents.” In: *ICML*. Volume 32. JMLR Workshop and Conference Proceedings. JMLR.org, pages 1188–1196. URL: `http://dblp.uni-trier.de/db/conf/icml/icml2014.html#LeM14`.

Mikolov, Tomas, Wen-tau Yih, and Geoffrey Zweig (2013). “Linguistic regularities in continuous space word representations”. In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751.

Pennington, Jeffrey, Richard Socher, and Christopher Manning (2014). “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Colophon

Publisher
Statistics Netherlands
Henri Faasdreef 312, 2492 JP The Hague
www.cbs.nl

Prepress
Statistics Netherlands

Design
Edenspiekermann

Enquiries
Telephone: +31 88 570 70 70
Via contact form: www.cbs.nl/infoservice